HAL open science

On the Combined Behavior of Autonomous Resource Management Agents

Siri Fagernes, Alva L. Couch

▶ To cite this version:

Siri Fagernes, Alva L. Couch. On the Combined Behavior of Autonomous Resource Management Agents. 4th International Conference on Autonomous Infrastructure, Management and Security (AIMS), Jun 2010, Zurich, Switzerland. pp.38-49, 10.1007/978-3-642-13986-4_5 . hal-01056625

HAL Id: hal-01056625

https://inria.hal.science/hal-01056625v1

Submitted on 20 Aug 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

On the Combined Behavior of Autonomous Resource Management Agents

Siri Fagernes[1] and Alva L. Couch[2]

[1]Faculty of Engineering
Oslo University College
Oslo, Norway
siri.fagernes@iu.hio.no

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

Abstract. A central issue in autonomic management is how to coordinate several autonomic management processes, which is assumed to require significant knowledge exchange. This study investigates whether two autonomic control units, which control the same system, can achieve some level of self-coordination with minimal knowledge exchange between them. We present the results from simulations on a model of two autonomous resource controllers influencing the same system. Each of the controllers tries to balance system utility with the cost of resource usage, without the knowledge of the second controller. Simulations indicate that coordination of autonomic management processes is possible, as the system performs close to optimally with minimal knowledge exchange between the two resource controllers.

Key words: self-organization, resource management, autonomic computing, agent management, distributed agents

1 Introduction

The vision of autonomic computing is to build systems that are capable of self-management, adapting to changes by making their own decisions, based on status information sensed by the systems themselves[1, 2]. Current solutions to autonomic computing rely upon "control loops" in which the managed system is monitored, changes are planned and implemented, and the effectiveness of changes is evaluated [3]. For many authors, the definition of control loops as provided in IBM's autonomic computing manifesto[1] is equated with the definition of autonomic computing.

However, there are some drawbacks to IBM's control architecture. One drawback is that distinct control loops are difficult to combine into a coherent management strategy. Another drawback is that the effectiveness of a single control loop depends upon the accuracy of its model of system behavior during the

planning phase[4]. Vendors deal with these two drawbacks by constructing autonomic controllers that gather and act upon global knowledge of system behavior. This knowledge is exchanged with other controllers via what might be called a "knowledge plane" [5, 6]. As a result, vendors pursue large monolithic approaches to autonomic control, in which an overarching controller gathers and acts upon data concerning all aspects of performance. The net result of this approach is that it is not practical for two non-communicating control approaches to collaborate or even co-exist.

In this paper, we study an alternative approach to autonomic control that is designed to foster collaboration between controllers and minimize necessary knowledge between them. In an initial AIMS paper last year [7], Couch demonstrated that an autonomic controller could achieve near-optimal behavior with an incomplete model of system behavior. Couch claimed that this model of control will allow controllers to be composed with minimal knowledge exchange. In this paper, we test that claim, by simulating the results of composing two controllers for front-end and back-end services under a variety of conditions. We demonstrate that composition is possible, and that reasonable results can be achieved from two controllers, provided that each controller is aware of when the other is operating.

The remainder of this paper is organized as follows. Section 2 gives a brief overview of related works. In section 3, we introduce the extended theoretical model which forms the basis of the simulations in the experiments, which are described in section 4. Section 5 gives an overview of our observations and results, which we discuss in section 6.

2 Related Work

There is a wealth of literature on the control-theoretic approach to autonomic management. Traditional approaches to resource control rely upon control theory to predict system response in the presence of changes[8]. These authors use the system model to predict overall response. Therefore, the usefulness of the technique varies with the accuracy of available models.

Another technique for gaining optimal control of application resources involves using optimized sampling strategies to compensate for lag times from unrestrained sampling[9]. Rather than controlling a short-term sampling window, these authors sample based upon experimental design principles and a concept of which samples are more "important" than others.

Other alternatives that do not rely on control theory are still dependent on constructing accurate models[10]. In these studies, an optimizer uses a model to predict the relationship between resources and behavior, and plans without control-theoretic assistance.

3 Model

As a sample case for our composition problem we have used a system consisting of a back-end server and a front-end server which –when combined –provides a service. The performance of the system is determined by the total *response time* P incurred in providing the service. The response time consists of two main parts:

- P_1: the response time at the front-end server.
- P_2: the response time at the back-end.

The response time will be influenced by several factors, but in this study we will isolate two such variables in the resource domain, so that R_1 is a adjustable resource variable on the front-end server, and R_2 on the back-end.

For the purposes of this paper, in like manner to Couch's simulations[7], we assume ideal scalability, so that $P_1 = \frac{L_1}{R_1}$ and $P_2 = \frac{L_2}{R_2}$. The total response time or *performance* P is then expressed by

$$P = \frac{L_1}{R_1} + \frac{L_2}{R_2}, \tag{1}$$

where L_1 and L_2 is the current load at each of the servers. An illustration of the scenario is shown in Figure 1.

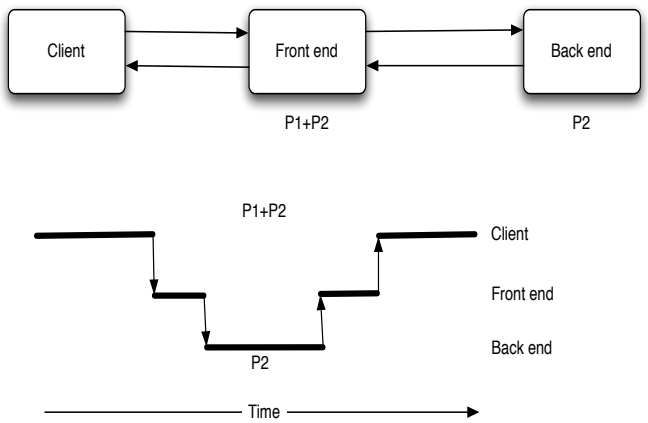

Fig. 1. The total system response time. Transmission time is ignored.

The associated *cost* S of providing the service is the sum of the costs of the current resource usage, which is defined as

$$C(R_1, R_2) = C(R_1) + C(R_2) = R_1 + R_2. \tag{2}$$

The associated value of receiving the service, which is given notation $V(P)$ is defined (in like manner to Couch's initial paper) according to the following expression:

$$V(P) = 200 - P = 200 - \frac{L_1}{R_1} + \frac{L_2}{R_2} \tag{3}$$

illustrating how the perceived value decreases with increasing response time, and increases with increases in R_1 and/or R_2. This cost function is chosen so that the overall reward function $V(P) - C(R_1, R_2)$ has a theoretical optimum value.

3.1 Feedback and control loops

The core of this study is to analyze how the closure operators control/adjust their individual resource parameters. Both operators receive feedback on the *perceived value* V of the delivered service S, and based on local knowledge of the cost of the current resource usage, each operator can make its local estimate of the derivative $\frac{dV}{dR}$. The dynamics of the model is simple; if $\frac{dV}{dR}$ is positive it will be beneficial to increase R, and if it is negative the resource usage should be decreased.

An obstacle lies in the fact that the change in value dV is a result of recent changes in both R_1 and R_2. It is challenging to estimate $\frac{dV}{dR_1}$ and $\frac{dV}{dR_1}$ correctly, since the operators controlling R_1 and R_2 do not know about each other's presence.

3.2 Theoretical Optimum and False Optimum

The choice of this model makes calculating the theoretical optimum of the operators' choices both simple and straightforward. The goal of the system management is to balance cost and value, i.e. to maximize the net value (value minus cost). This is equivalent to maximizing

$$V(P) - C(R_1) - C(R_2) = 200 - \frac{L}{R_1} - \frac{L}{R_2} - R_1 - R_2, \tag{4}$$

which trivially gives the following values for optimal choice of R_1 and R_2

$$R_1^{Opt} = R_2^{Opt} = \sqrt{(L)}, \tag{5}$$

where L represents the common load $L = L_1 = L_2$.

Based on the fact that these two operators do not know about each other, it is reasonable to expect them to make a wrong estimate of the optimal choice of resource usage. Each operator receives feedback of the overall value in the system, but does not take into consideration the cost of the other resource variable.

In a scenario where the resource operators work concurrently with equal resource values R, the perceived net value function would look like this:

$$V(R) - C(R) = 200 - \frac{2L}{R} - R \tag{6}$$

which gives the optimal value $R_{\text{false}} = \sqrt{2L}$, which we refer to as the *false optimum* in the remainder of this paper, because some management designs achieve this optimum instead of the true optimum.

4 Experiment

We implemented a simulator based on the two-operator model, with the aim of simulating how the system behaves, and of determining whether the independent resource controllers are able to adjust the resource level to a level that is close to the optimal values. The system was tested for two different types of load L:

- *Constant load*, where $L = 1500$.
- *Sinusoidal load*, where $L(t) = 1000sin((t/p) * 2\pi) + 2000$, which makes the load varying periodically between 1000 and 3000.

We developed and tested two methods for updating the resource variable. Under our first method, both controllers receive system feedback and simultaneously update their resource variable. We designate this method as the *concurrent* algorithm in the remainder of the paper.

Under the second approach, each controller adjusts its variable while the other remains constant through a specific number of iterations (cycles). During each cycle, one update of the system performance is fed back to the resource controllers. In this approach, which we designate as the *alternating* algorithm, each controller tries to converge to its optimal resource usage level, without interference of the other controller's updates.

5 Results

This section presents and explains the main findings from the simulations. The metrics for evaluating model performance are

1. The actual *net value* produced by the system throughout each simulation, compared with the theoretically highest value.
2. The actual *resource usage* (actual values of R_1 and R_2) compared with the theoretical optimal value for R_1 and R_2.

The simulator was run for a range of values of R_1 and R_2, but due to space limitations only a small selection is presented here. However, the presented findings occurred for all tested values.

True and false optima

As computed earlier in this paper, the theoretically optimal behavior of both resource controllers is a resource level of $R_1 = R_2 = \sqrt{L}$. In other words, a desirable result would be to have both variables R_1 and R_2 converge to $\sqrt{L}$. The plots we computed of the actual behavior of R_1 and R_2 all indicate some kind of convergence, but this convergence does not necessarily come to the true optimum behavior.

Concurrency leads to false optima

Throughout the simulation results, concurrent updates of the resource variables R_1 and R_2 by both controllers acting independently resulted in convergence to false optima for at least one of the variables. The value of the false optima depended on the *initial* resource variables.

If the initial values of R_1 and R_2 were *identical*, both variables appeared to converge to the same false optimum $\sqrt{2L}$ as seen in Figure 2.

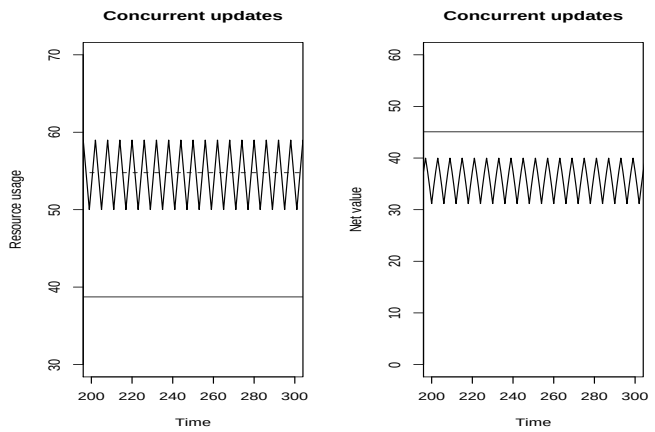

Fig. 2. Actual resource usage (left) and net value (right), under constant load. Initial resource values: $R_1 = R_2 = 50$. In the left figure, the dashed straight line represents the false optimum $\sqrt{2L}$ while the lower solid line represents the actual optimum level $\sqrt{L}$. Net value (right) is compared to the theoretical maximum net value (represented by the upper straight line).

However, R_1 and R_2 have different initial values, the lowest one ends up "dominating" by converging close to the optimal value ($\sqrt{(L)}$). The other gets "stuck "at a higher level, and sometimes even increases from its initial level. For initial values far from the optimal this trend wreaks a devastating impact on the efficiency of the system (Figure 3).

Concurrent, uncoordinated resource adjustments result in poor performance based on achieved net value, particularly when there is a difference in initial values of R_1 and R_2. If the difference between these is above a certain threshold, the system produces *negative* net value (Figure 4).

Alternating between processes leads to true optima and thrashing

Our next approach was to *alternate* the two processes. We constrained one of them from acting while the other adjusted its resource parameter. As illustrated

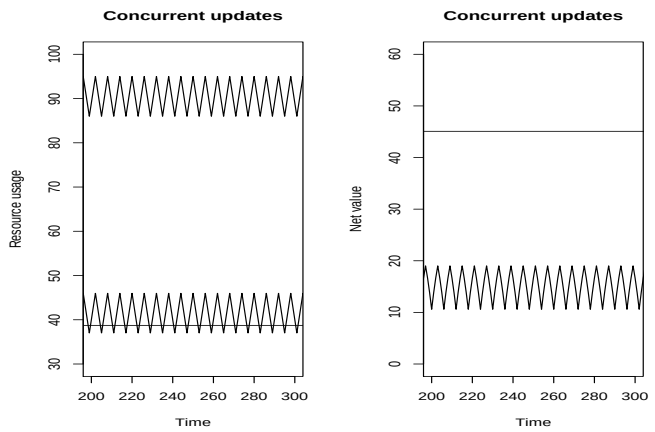

Fig. 3. Actual resource usage (left) and net value (right), under constant load. Initial resource values: $R_1 = 1, R_2 = 50$. In the left figure, the lower oscillating curve represents the resource variable R_1, while the upper represents R_2. The straight line in the left figure represents the actual optimum level $\sqrt{L}$. In the right figure, the straight line represents the theoretical maximum net value.

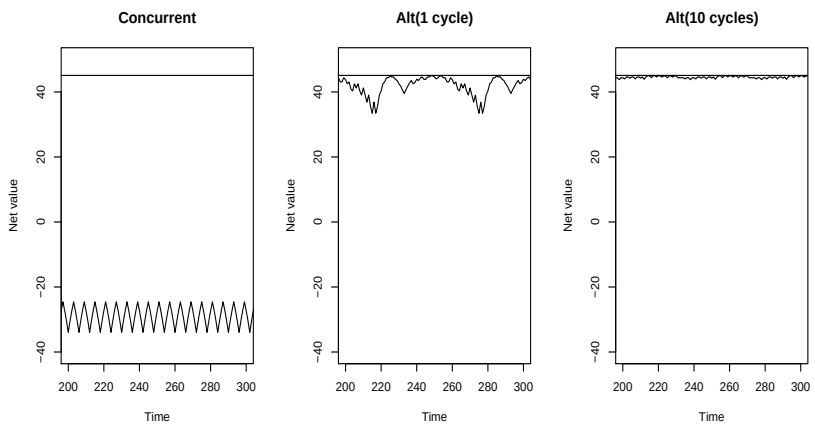

Fig. 4. Achieved net system value, constant load. Initial resource values: $R_1 = 1, R_2 = 100$.

in Figure 5, the apparent effect is that the resource values chosen by both processes now oscillates around the theoretical optimum R^{opt}. In our view, this algorithm has managed to "escape" the false optimum. The downside is that the oscillations are "larger", which leads to worse performance with respect to total net value, as illustrated in Figure 6.

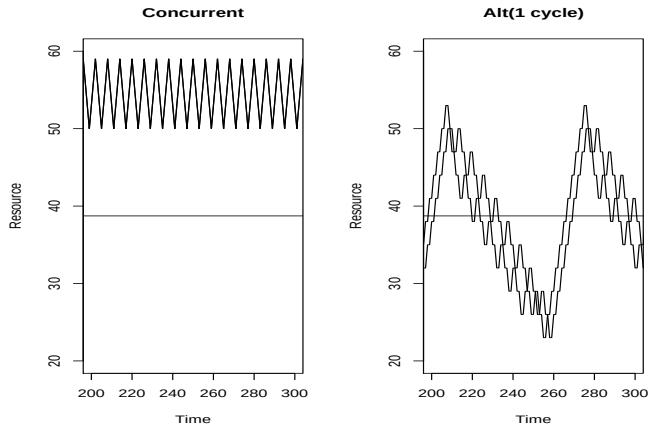

Fig. 5. Actual resource usage, constant load. Initial resource values: $R_1 = R_2 = 50$.

Another feature of this experiment, is that the system seems to be less vulnerable to choices for initial values of R_1 and R_2, compared to when the processes are run concurrently (Figure 4).

The best-case situation.

The next approach was then to increase the time that each process was able to adjust while the other was kept constant. The idea was to let each process converge to its optimal value without interference from the other variable.

We have run the algorithms for varying measurement window and number of cycles. As seen in Figure 7, by keeping each resource variable constant for 10 cycles, the resulting net value has improved significantly (compared to the "1 cycle"-case).

However, increasing the number of cycles beyond 10 does not seem to improve things, as the net value is actually lower for cycles=25 and 50.

Keeping the number of cycles at 10 (which seemed to give the best results in some of the experiments), and varying the measurement window size for the input data, showed that increasing the window generates more chaotic behavior and worse performance with respect to achieved net value, as seen in Figure 8.

However, the pattern of less efficient management when the number of cycles increased, proved not to be consistent throughout the simulations, as seen in

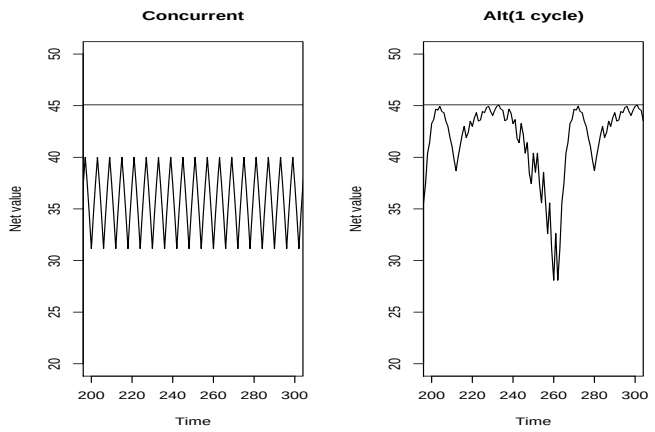

Fig. 6. Achieved net system value, constant load. Initial resource values: $R_1 = R_2 = 50$.

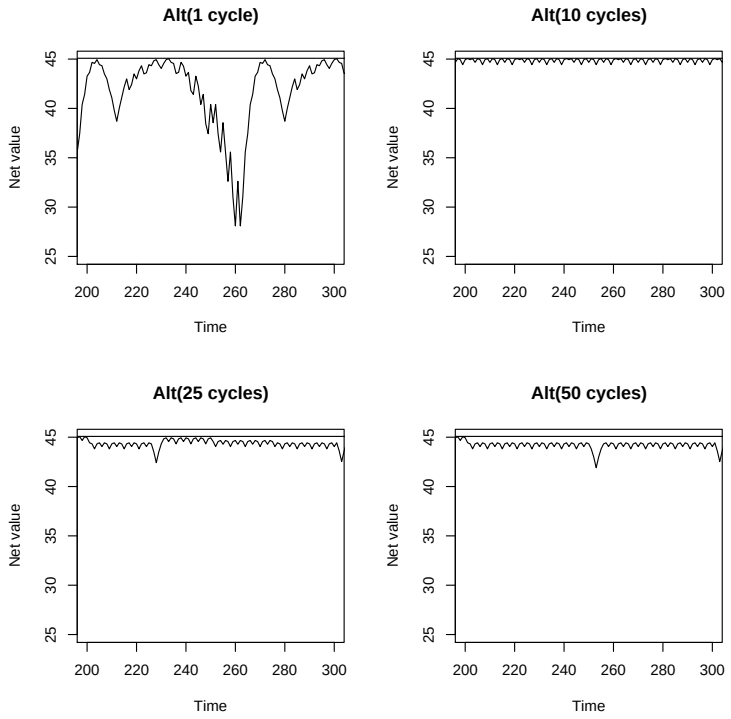

Fig. 7. Achieved net system value, constant load. Initial resource values: $R_1 = R_2 = 50$.

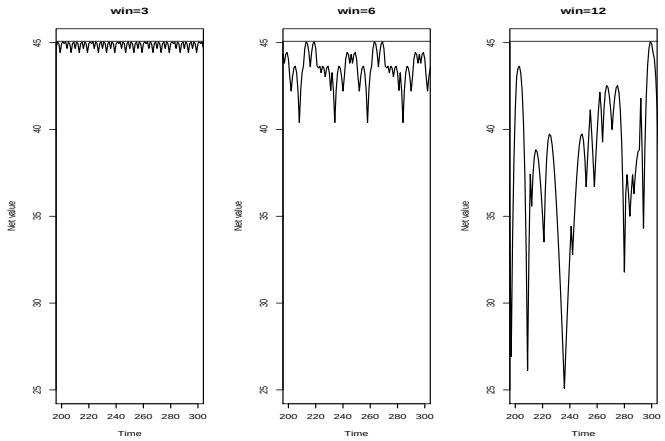

Fig. 8. Achieved net system value, constant load. Initial resource values: $R_1 = R_2 = 50$. Alternating for 10 cycles.

Table 1. Here we have systematically computed the *theoretical efficiency* as introduced in [7]. Theoretical efficiency was defined as the ratio of sums of achieved net value to the theoretically best:

$$E = \frac{\sum_i (V_i - C_i)}{\sum_i (V_i^{\text{opt}} - C_i^{\text{opt}})} \tag{7}$$

where $V_i^{\text{opt}} - C_i^{\text{opt}}$ is the best theoretical net value the system can produce at time i. As seen in Table 1, keeping each variable constant for 10-75 cycles give significantly higher efficiency than just keeping them constant for one cycle (e.g. the values in column Alt(1) are much lower). However, the number of cycles that are optimal varies according to the choice of initial values of R_1 and R_2.

Win=3	Conc	Alt (1)	Alt(10)	Alt (25)	Alt(50)	Alt (75)	Alt (200)
R1=R2=1	0.815	0.934	0.977	0.993	0.988	0.993	0.911
R1=R2=50	0.794	0.911	0.995	0.987	0.986	0.986	0.990
R1=1, R2=50	0.335	0.936	0.995	0.993	0.988	0.990	0.909
R1=1, R2=100	-0.643	0.934	0.988	0.987	0.991	0.989	0.914
R1=R2=100	0.815	0.934	0.987	0.987	0.993	0.993	0.979

Table 1. Theoretical efficiencies (constant load).

Increasing the measurement window size from 3 seems to have undesirable effects, as seen in Figure 9, which is consistent with Couch's claim that more data sometimes hurts performance [7, 11, 12].

The results discussed here were obtained while running the simulator under constant load. We also ran the simulator under sinusoidal load, to confirm that the algorithms still retain convergence effects as reported by Couch[7]. One example is shown in Figure 9.

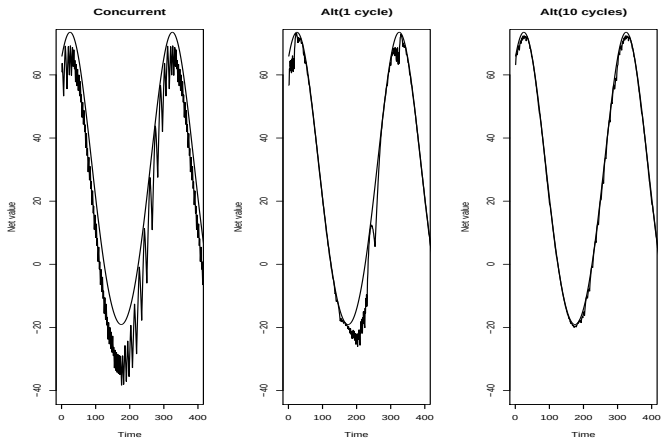

Fig. 9. Achieved net system value, sinusoidal load. Initial resource values: $R_1 = R_2 = 50$.

6 Conclusion

In this paper, our goal was to understand the mechanisms that are required in order to achieve self-organization of independent autonomic components or processes.

The findings indicate that controllers can collaborate effectively when they are constrained under a simple process of taking turns, and without requiring any other information exchange. This finding challenges the popular belief that a knowledge plane and a concept of global knowledge are necessary to allow multiple processes to co-exist.

For this kind of self-organization to be possible, our experiments indicate that it is sufficient that the autonomic entities adjust their behavior at different times, i.e. completely concurrent execution of controllers behavior is not sufficient for the system as a whole to achieve its goals. An algorithm that lets each entity affect the common environment or system (and receive system feedback) for several iterations without the interference of other components seems to help the components to achieve close-to-optimal behavior.

We found that while the time window in which each resource controller operates alone must be of some size, the measurement window is constrained to be quite small. This corresponds well with earlier results [7, 11] which concluded

that high reactivity can be more efficient than requiring large numbers of measurements.

At the outset, we challenged the idea that management agents must be tightly coupled to coordinate their actions. Our initial hypothesis was that this coupling could be much less than is present in other proposed and actual agents. This study showed that the minimal level of coupling needed was temporal coordination, which is much less than the total knowledge exchange proposed in competing solutions.

References

1. J.O.Kephart and D.M.Chess. The vision of autonomic computing. *IEEE Computer*, 2003.
2. M. C. Huebscher and J. A. McCann. A survey of autonomic computing - degrees, models, and applications. *ACM Comput. Surv.*, 2008.
3. J.L. Hellerstein, Y. Diao, S. Parekh, and D.M. Tilbury. *Feedback Control of Computing Systems.* John Wiley & Sons, 2004.
4. J.O.Kephart. Research challenges of autonomic computing. *Proceedings of the 27th international conference on Software engineering.*, 2005.
5. D. F. Macedo, A. L. dos Santos, J. M. S. Nogueira, and G. Pujolle. A knowledge plane for autonomic context-aware wireless mobile ad hoc networks. *Management of Converged Multimedia Networks and Services (LNCS)*, 2008.
6. M. Mbaye and F. Krief. A collaborative knowledge plane for autonomic networks. *Autonomic Communication*, 2009.
7. A. Couch and M. Chiarini. Dynamics of resource closure operators. *Proceedings of Autonomous Infrastructure Management and Security 2009, Twente, The Netherlands, June 30-July 2, 2009*, 2009.
8. P. Padala, K. G. Shin, X. Zhu, M. Uysal, S. Singhal Z. Wang, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. *EuroSys*, 2007.
9. B. Xi, Z. Liu, M. Raghavachari, C. H. Xia, and L. Zhang. A smart hill-climbing algorithm for application server configuration. *WWW*, pages 287–296, 2004.
10. P. Padala, K.-Y. Hou, K. G. Shin, M. Uysal, S. Singhal Z. Wang, and A. Merchant. Automated control of multiple virtualized resources. *EuroSys*, pages 13–26, 2009.
11. A. Couch and M. Chiarini. Combining learned and highly-reactive management. *Proceedings of Managing Autonomomic Communications Environments (MACE) 2009, Venice, Italy, Oct 27, 2009.*, 2009.
12. A. Couch, M. Burgess, and M. Chiarini. Management without (detailed) models. *Proceedings of Autonomic and Trusted Computing 2009, Brisbane, Australia, July 7-9, 2009.*, 2009.