MIT-CSAIL-TR-2011-019 April 1, 2011

A Comparison of Autonomic Decision Making Techniques

Martina Maggio, Henry Hoffmann, Marco D. Santambrogio, Anant Agarwal, and Alberto Leva

A Comparison of Autonomic Decision Making Techniques

Martina Maggio[1,2], Henry Hoffmann[2],
Marco D. Santambrogio[1,2], Anant Agarwal[2] and Alberto Leva[1]

[1]Dipartimento di Elettronica e Informazione
Politecnico di Milano, Italy
{maggio,santambr,leva}@elet.polimi.it

[2]Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology, Cambridge
{mmaggio,hank,santambr}@mit.edu, agarwal@csail.mit.edu

ABSTRACT

Autonomic computing systems are capable of adapting their behavior and resources thousands of times a second to automatically decide the best way to accomplish a given goal despite changing environmental conditions and demands. Different decision mechanisms are considered in the literature, but in the vast majority of the cases a single technique is applied to a given instance of the problem. This paper proposes a comparison of some state of the art approaches for decision making, applied to a self-optimizing autonomic system that allocates resources to a software application, which provides direct performance feedback at runtime. The Application Heartbeats framework is used to provide the sensor data (feedback), and a variety of decision mechanisms, from heuristics to control-theory and machine learning, are investigated. The results obtained with these solutions are compared by means of case studies using standard benchmarks.

Categories and Subject Descriptors

D.2.10 [**Design**]: Methodologies; F.1.1 [**Models of Computation**]: Self-modifying machines; I.2.8 [**Problem Solving, Control Methods, and Search**]: Control theory, Heuristic methods

General Terms

Algorithms, Design, Performance

Keywords

Decision mechanisms, Comparison, Design approaches

1. INTRODUCTION

Autonomic computing is a very promising research area for confronting the complexity of modern computing systems. Autonomic systems manage themselves without human intervention, and their development involves a variety of exciting challenges [16]. One of the most important of these challenges is the establishment of systematic and reproducible processes for the design of autonomic systems.

In the literature, the autonomic paradigm is characterized by the presence of three distinct phases: *sensing*, *deciding*, and *acting*. Notable examples of this division are the Monitor, Analyze, Plan and Execute (MAPE) or Observe, Decide, Act (ODA) loops [1]. In both cases, the "decide", or equivalently the "analyze and plan", phase is responsible for providing and enforcing the desired properties of the self-managing system. Thus, the design of the decision phase is essential for obtaining the desired self-configuring, self-healing, self-optimizing and self-protecting autonomic system [17].

It has been noted that the design of closed-loop autonomic systems shows impressive convergence with control engineering, which to date has been only marginally exploited in the design of computing systems [13]. In fact, modern control engineering may provide useful complements or alternatives to the heuristic and machine-learning decision methods used to date. In order to create systematic and reproducible processes for decision making, it is important to understand both the quantitative and qualitative differences between techniques.

This paper begins an investigation comparing decision making processes for self-optimizing autonomic computing systems, covering heuristic, control, and machine learning methods. In detail, the novel contributions of this paper are:

- a discussion of literature techniques (heuristic, control-theoretical and machine learning-based), analyzing their properties and guarantees, both theoretically and in practice;
- the proposal of a set of reference problems for comparing the performance and applicability of the mentioned solutions;
- the synthesis, development, implementation and testing of said techniques in the mentioned framework;
- the application of the proposed solutions to some benchmark test cases, taken from the PARSEC suite [4], and the presentation the results, both in detail for a single benchmark and in an aggregate to summarize the effects seen in multiple case studies.

Although autonomic systems based on the feedback control theory have been proposed [20], the corresponding engineering tools and processes are far from fully exploited to date. To give a brief example, controlling behavior of applications requires application-level sensors if we are to take full advantage of control theory's capabilities. Such a capability complements typical analysis, providing, for example, *online* time and convergence guarantees. A reason for the limited use of control is that concepts like those just mentioned are quite well assessed in the control engineering community, yet it is not immediately obvious how to extend them to the autonomic computing domain. Exploitation of a control-theoretical framework requires a modeling phase involving all system components prior to the algorithmic design of the computing system.

Having a model of the system, in the control-theoretical sense, means writing the equations that describe system behavior. In the case of autonomic computing systems, developing such a model may be difficult. In contrast, machine learning techniques may require little to no explicit modeling because they capture complex relationships online, automatically learning the interactions between components in a complex system. Thus, in practice, the best solutions will probably combine techniques, enhancing feedback control solutions with machine learning mechanisms, and *vice versa*.

The remainder of the paper is organized as follows. Section 2 presents some of the achievable properties and motivations for building a decision-making mechanism. Section 3 focuses on one of said properties, self-optimization, and shows the techniques qualitatively compared for that challenge. In Section 4 the mentioned solutions are described in detail, from the basic idea to the implementation, and some experimental results are provided in Section 5. Finally, Section 6 concludes the paper.

2. MOTIVATIONS

An autonomic computing system may be built with different goals, but its essence is self-management [17]. Four main aspects of self-management emerge in the literature.

Self-configuration: A self-configuring system is able to configure itself according to high-level policies and objectives, thereby improving its effectiveness [31]. One of the most important goals of self-configuration is the ability of a system to reconfigure itself online, seamlessly incorporating new components while existing ones adapt to these new features. Self-configuration is beyond the scope of this work, in that it involves the way the system is *designed*, while here the focus is on how the system decides the actions to take.

Self-protection: A self-protecting system is capable of defending itself from malicious attacks or cascading failures. Self-protection is potentially related to the decision making process, albeit the main issue is the attack detection mechanism. Machine learning techniques are often used to build intrusion detection systems: reviewing the matter is impossible here due to space limitations, the interested reader can refer to the recent survey [29]. In this context, some control strategies were proven useful in building a system that autonomously controls a worm spread [10].

Self-healing: A self-healing system detects, diagnoses, and repairs localized problems resulting from bugs or failures in software and hardware. The problem detection is usually considered part of sensing, and although including the "correct" sensors may be difficult, this does not affect decision making. Breitgand *et al.* proposed a method for automatic computation and adaptation of performance thresholds for system components to detect failures [7], in a view to improving the detection efficacy.

Indeed, the diagnosis and reaction mechanism is an important part of the decision and plan phase of the loop. A notable example is described in [9], where the authors assume that the failure has already been detected by the underlying system and use aggressive logging to trace request paths through the entire system, recording the components used by each request. Using machine learning algorithms (decision trees) and querying a database that contains data on many independent requests, the authors simultaneously examine many potential causes. Another work exploiting machine learning for self-healing is [11], where a solution is proposed to increase the throughput of TCP over wireless links, i.e.,to distinguish between different classes of failures so as to prevent the system from reducing its rate when the packet loss is due to a link error. Supervised machine learning techniques are used to automatically derive a classification model for loss causes. Each observation is an input/output pair where the inputs are variables describing the state of the network when a loss occurs and the output is a label to denote a loss due to a congestion or due to a link error. The paper presents a comparison of different machine learning techniques (decision trees, neural networks, k-th nearest neighbor) and shows possible machine learning approaches to the diagnose of the network problems.

The two examples just mentioned show how machine learning techniques were proven useful in the classification of failures. It is unlikely that heuristic and/or control-theoretical approaches can be as effective for this classification problem. In fact, to the best of the authors' knowledge, there is no documented research on the matter.

Self-optimization: A self-optimizing system is capable of monitoring and tuning itself according to performance analysis. Performance-based tuning strategies play a key role in the autonomic computing systems definition and are strictly related to the decision making process. An autonomic computing system is supposed to seek ways to improve its operation, identifying and seizing opportunities to be more efficient in performance or cost.

The focus of this paper is the comparison of different techniques to design a self-optimizing system, covering heuristic, machine learning and control-theoretical approaches. A representative problem will be used as a reference example. The goal is to manage the performance of software applications which have been instrumented to emit their performance level via the Application Heartbeat framework [14]. By making calls to the Heartbeat API, applications signal "heartbeats" at some important places in the code (for example, a video encoder may emit a heartbeat for every encoded frame). Additional functions in the Heartbeat interface allow applications to specify their goals in terms of a desired heart rate. The decision making process should assign operating system resources to each instrumented application in order for the application to match the specified performance level, i.e., as envisioned and discussed in [2].

3. DESIGN FOR SELF-OPTIMIZATION

Several techniques have been used to synthesize decision mechanisms for self-optimizing computing systems. As a notable example, [25] addresses the problem of how to apply a genetic algorithm. The case study proposed therein is the dynamic reconfiguration of an overlay network for distributing data to a collection of remote data mirrors. The developed algorithm is able to balance the competing goals of minimizing costs and maximizing data reliability and network performance.

In this work, we detail a single problem, whose generality allows us to draw some considerations about the decision making processes for self-optimizing systems. Suppose we want to build a self-optimizing operating system, and one of its tasks is the assignment of resources to running applications. Notice that the word *resources* may assume different meanings. In a single device, an application may receive computational units or memory, while in a cloud infrastructure, a resource can be a server devoted to responding to some requests. Each manageable resource is a touchpoint in the sense of [1]. Some proposals to address the management of a single resource have been published in the literature; however, proposals to manage multiple interacting resources are more rare. Intuitively, the number of ways the system capabilities can be assigned to different applications grows exponentially with the number of resources under control. Moreover, the optimal allocation of one resource type depends in part on the allocated amounts of other resources, requiring coordination.

In [5], Bitirgen *et al.* periodically redistribute shared system resources between applications at fixed decision-making intervals, allowing the system to respond to dynamic changes. Their infrastructure is based on an Artificial Neural Network that learns to approximate the application performance from sensor data. Their neural network takes inputs from the system (amount of cache space, off-chip bandwidth, and power budget allocated to the application). In addition, the neural network is given nine attributes describing recent program behavior and current cache state (number of reads and misses, and so on). The network is implemented in a separate hardware layer, to reduce its overhead. Training data can be gathered online, allowing an accurate model even when the operating conditions are changing.

3.1 Techniques

This section presents an overview of some common techniques for making decisions in a self-optimizing system.

Heuristic solutions start from a guess about application needs and adjust this guess. Heuristic solutions are designed for computational performance or simplicity at the potential cost of accuracy or precision. Such solutions generally cannot be proven to converge to the optimum or desired value. A notable example is the greedy approach in [8] that optimizes resource allocation and energy management in a hosting center. This system controls server allocation and routing requests based on an economic model, where customers bid for resources as a function of service volume and quality. Bodík *et al.* use linear regression to predict the workload based on previous data. This predicted workload is fed to a performance model that estimates the number of servers required to handle it; these servers are subsequently activated [6].

Standard control-based solutions employ canonical models – two examples being discrete-time linear models and discrete event systems – and apply standard control techniques such as Proportional Integral (PI) controllers, Proportional Integral and Derivative (PID) controllers, optimal controllers, Petri nets. Assuming the model to be correct, some properties may be enforced, among which stability and convergence time are probably the most important ones, thereby providing *formal performance guarantees*. As an example, Pan *et al.* [24] propose two PI controllers for guaranteeing proportional delay differentiation and absolute delay in the database connection pool for web application servers. The model used is a first order linear time-invariant system and the PI controllers are design with the Root Locus method. Such techniques may not however be enough in the case of heavily varying environment or workload conditions.

Advanced control-based solutions require complex models, with some unknown parameters (e.g., the machine workload) that may be estimated online, to provide Adaptive Control (AC). AC requires an identification mechanism and the ability to adjust controller parameters on the fly. Another advanced control strategy is Model Predictive Control (MPC) where the controller selects the next actions based on the prediction of the future system reactions. The overhead of sophisticated control solutions is greater than that of standard controls; however, one may still be able to formally analyze parameter-varying systems and prove stability, obtaining formal guarantees even in the case of unknown operating conditions. For example, [18] proposes an approach based on model identification, to adjust the CPU percentage dedicated to the execution of a web server. A first-order auto-regressive model is used for identification purposes. A PI control structure is presented together with an adaptive controller. The recursive least squares method is used to estimate the model parameters.

Model-based machine learning solutions require the definition of a framework in which to learn system behavior and adjust tuning points online. Neural Networks (NN) are often useful to build a model of the world for control purposes, see again [5]. NN solutions may be used to predict the system reaction to different inputs and, given some training samples, to build a model. The structure of the network and the quality of the training data are critical to performance. The accuracy of the results depend on these crucial choices, and thus no *a priori* guarantees can be enforced. Another model-based family of techniques is Genetic Algorithms (GA), see e.g. [25] for a discussion on their use in autonomic system. Using a genetic algorithm requires selecting a suitable representation for encoding candidate solutions (in other words, a model). In addition, some standard operators (crossover and mutation) must be defined and a mathematical function must be provided to rate candidate solutions and select among them. The overhead of both neural networks and genetic algorithms may in principle be very significant.

Model-free machine learning solutions do not require a model of the system. A notable example is Reinforcement Learning (RL), even if a recent research trend is to complement RL solution with a model definition [27, 28]. According to [22], RL agents face three major challenges. The first challenge is how to assign credits to actions, the second is how to balance exploration versus exploitation and the third is generalization. The convergence time of an RL algorithm is often critical [26] and complementing them with a model of the solution space may decrease it [30].

3.2 Qualitative comparison

Providing a meaningful comparison of the mentioned techniques is apparently hard. Part of the difficulty is due to the fact that literature works typically implement a single technique to solve a specific problem. Before proposing an experimental evaluation of different techniques, some qualitative points are thus worth briefly discussing. Specifically, we compare these techniques along four dimensions: the presence or absence of a model, the extent to which they provide analytic performance guarantees, their ability to handle unexpected conditions, and their ease of implementation.

One point that differentiates the techniques is the presence (or the absence) of a *model*. Intuitively, a heuristic solution in principle does not require a model, while RL is the only model-free machine-learning based mechanism mentioned herein. The difficulty of developing a model can vary depending on what that model is required to capture. In the case of a NN, for example, defining a model means structuring the dependency between input and output variables. Equivalently, setting up a GA means defining a way to encode the solution and to evaluate it, without necessarily capturing the relationship between the modeled quantities. On the contrary, building a model for control-theoretical purposes, means deriving the mathematical relationship between the involved entities. This crucial matter will be treated with an example in Section 4.2.1.

Another interesting comparison point is the presence of performance guarantees. Ideally, a decision methodology may be proven to converge to a predefined performance level with a known convergence time, at least in standard (or "nominal") conditions. This is the case, for example, of standard control techniques, where the design of the system may be carried out so the system is asymptotically stable, implying that the measured signal will approach the set point with the desired accuracy and timing. With a control-theoretical solution one is able to compute the convergence rate (i.e., the time the system needs to stabilize at the desired value) *analytically*. This is usually not the case with heuristic or machine learning solutions, although in the latter case some algorithms are proven to converge to the optimal solution if it exists and under some technical assumptions [26].

The third topic to be addressed is how the decision mechanism is able to handle unexpected (non-nominal) situations. Such situations may include unseen data, i.e., data that were not tested before and still depend only on the entities under control. Also there may be environmental fluctuations. Suppose for example that something is happening in the machine we are controlling, or in the network, and does not depend on the entities under control. This situation is different from the previous one, since the decision mechanism may not have the correct actuators to act on the system. The last thing to be handled are failures, distinguished as *soft* or *hard*. Soft failures limit the performance of the system, while hard failures completely eliminate the system's ability to respond to requests. A control-based solution is potentially able to handle any kind of situation, except for hard failures, especially if the system is augmented with identification and prediction mechanisms; however, the system needs to be designed with the correct actuators to handle performance degradations and environmental fluctuations. Suppose, for example, that a chip temperature control mechanism reduces the clock frequency when the temperature is too high. In this case, any decision mechanism should be able to increase the number of computation units (CPUs) to be allotted to a specific application in order for it to reach its goals. However, if the number of CPUs is not an actuator, any solution can fail. At the same time, machine learning solutions may handle unseen data and failures if designed carefully. However, it is usually stated that for example in NN, the test data that should be provided to train the network has to be representative of the entire space of solutions, therefore tendentiously reducing robustness. A decision mechanism's ability to handle unseen data is something that should be carefully analyzed.

The last consideration is on ease of programming. A heuristic solution usually does not require much effort to be implemented, while machine learning and control theory-based ones are often harder, as a system analysis (in the control engineering sense) is required.

4. IMPLEMENTATION

In the following we propose, discuss and implement some decision making solutions, and evaluate them with some experiments. The following sections will provide details about each solution implementation and synthesis.

4.1 Heuristic frameworks

Setting up a heuristic solution for the addressed problem is very easy, the main drawback being the lack of guarantees. In the following we sketch a possible solution and show its behavior in the proposed case study. Suppose the hardware device has some capabilities that can be tuned online, in this case the number of cores assigned to the application and the frequency of the machine. The sensing framework just introduced provides performance data from the application. A simple heuristic solution is to list system states in order from least to most powerful. The heuristic solution keeps track of the current state. It moves to a less powerful state if the measured performance is above the threshold and a more powerful state if performance is below threshold.

4.2 Control frameworks

The pioneering work by Hellerstein *et al.*, [12] contains a variety of examples of control-theoretical solutions for computing systems problems, generally with a view toward making said systems adaptive, robust, and stable. For our resource allocation problem, a preliminary control result is published in [21]. In the following we extend that result, limiting the scope to the comparison of different decision making techniques. In so doing, we will also explain the basic steps to build a control system for resource management.

A key point in building a control system is the choice of the modeling framework and control formalism. There are many possibilities including continuous or discrete time linear or non-linear systems, discrete event models, Petri networks, and a vast corpus of possible other choices. In the following, discrete-time linear systems are used. Once the formalism is chosen, the application of a fully control-theoretical approach requires that a model is written. In fact, the synthesis of a control system starts from the definition of what is called the "open loop behavior" of the object to be controlled. In the definition of the open loop behavior we should introduce a "control signal" that drives the system performance.

Probably one of the best reasons to use a control-theoretical framework for a decision mechanism is the performance guarantees control provides. In fact, when modeling and analyzing the closed-loop system, proving stability means proving that the heart rate value would reach the desired point, if the control system is well-designed.

4.2.1 The model

We define the model for the open loop behavior of our system as follows. The performance of the application at the k-th heartbeat is given as

$$r(k) = \frac{s(k)}{w(k)} + \delta r(k) \tag{1}$$

where $r(k)$ is the heart *rate* of the application at the k-th beat, $s(k)$ is the relative speedup applied to the application between time $k-1$ and time k, and $w(k)$ is the *workload* of the application. The workload is defined as the expected time between two subsequent heartbeats when the system is in the state that provides the lowest possible speedup. In this model, speedup is applied to the system and clearly controls the heart rate signal. This formulation is general so the source of speedup can vary and may include the assignment of resources, such as cores, servers and memory, or the online modification of the algorithms used in the application.

The simplicity of this model is both an advantage and a disadvantage. Obviously we are not modeling all the components that may interact with the application and change its performance value, but a model does not need to be complete to serve its control purposes (as decades of experience in other domains like process control have shown). We introduce the term $\delta r(k)$ as an exogenous disturbance that may vary the application behavior in unexpected ways to deal with the unknown. However, using a simple model to describe a much more complex behavior may be effective, if we are correctly describing the main components that interact with the modeled object.

4.2.2 Standard control techniques

The next step in order to exploit the capabilities of a control-theoretical framework is the choice of a controller that constrains the behavior of the closed-loop system, usually named the control synthesis. This build phase involves the specification of the desired behavior, in this case, maintaining a target heart rate value. Some simple controllers may be synthesized for the problem at hand.

Probably the easiest control solution would be a Proportional (P) controller, that would compute the relative speedup proportionally to the error between the desired heart rate and the actual measured one. Another viable choice are Proportional and Integral (PI) and Proportional, Integral and Derivative (PID) controllers. In the former case the speedup is computed as a function of the error and its integral over time, while in the latter a term based on the actual trend is added to these two. These controllers are usually very easy to build, the only action involved is the choice of their parameters (e.g., the degree of proportionality of the various terms). More details on how to build and parameterize such controllers may be found in [3, 23].

Another standard control solution is a Deadbeat controller. Its synthesis involves the specification of the Z-transfer function between the input data (the desired heart rate, $\bar{r}$) and the output (the measured heart rate r). In our case we specify that function as

$$\frac{R(z)}{\bar{R}(z)} = \mu \frac{z - z_1}{(z-p_1)(z-p_2)} \tag{2}$$

where z^{-1} is the delay operator and $\{z_1, p_1, p_2\}$ are a set of customizable parameters which alter the transient behavior. We want to shape the function so that the overall gain of the closed loop system is 1, meaning that the input signal is reproduced to the output one, therefore we choose $\mu = (1-p_1)(1-p_2)/(1-z_1)$.

The Deadbeat control is straightforward to synthesize, in that the closed loop transfer function

$$\frac{R(z)}{\bar{R}(z)} = \frac{R(z)C(z)}{1 + R(z)C(z)} \tag{3}$$

where $C(z)$ is the controller transfer function and can be obtained solving the equation. The control equation is then found by taking the inverse Z-transform of $C(z)$ to find the speedup $s(k)$ to apply at time t:

$$\begin{aligned} s(k) \quad = \quad & F \cdot [As(k-1) + Bs(k-2) + \\ & Ce(k)w(k) + De(k-1)w(k-1)] \end{aligned} \tag{4}$$

where $e(k)$ is the *error* between the current heart rate and the desired heart rate at time k and the values of the parameters $\{A, B, C, D, F\}$ come from the controller synthesis. The choice of the parameters $\{z_1, p_1, p_2\}$ allows customization of the transient response. A preliminary discussion on different viable ways to impose the desired speedup value, the parameters values and some words on how to shape different responses and therefore select the parameter values can be found in [15].

However, it is evident that the speedup equations depend on $w(k)$, the workload value. It is not always possible to have an off-line estimation of the workload value, so a robustness analysis is in order. Suppose to use w_o as a nominal value for the workload. Trivial computations show that if the actual workload w is expressed as $w_o(1+\Delta w)$, thereby introducing the unknown quantity Δw as a multiplicative error, then the eigenvalue of the closed loop system is $\frac{\Delta w}{(1+\Delta w)}$. Requiring the magnitude of said eigenvalue to be less than unity, one finds that closed loop stability is preserved for any Δw in the range $(-0.5, +\infty)$ hence if the workload is not excessively overestimated such a simple control law can effectively regulate the system despite its variations.

4.2.3 Advanced control techniques

A standard control solution may be sufficient for many systems and applications, but much more can be done by introducing more articulated (e.g., adaptive) techniques. Suppose that the proposed control system is augmented with an identification block, which provides an online estimation of the workload. Adding this capability to a standard control system turns it into an *adaptive* one. Different techniques can be used for identification; we implement both a Kalman filter and a Recursive Least Square algorithm to estimate the workload value [19] and turn the standard Deadbeat controller into an adaptive one.

However, even more complex solutions may be envisaged, where more parameters describe the relationship between the control entities (number of cores assigned to the application, frequency of the machine, and so forth) and the performance metric, to build a more sophisticated controller.

4.3 Machine learning frameworks

Machine learning techniques are often employed as decision mechanisms for a variety of systems. Roughly speaking, machine learning allows computers to evolve behaviors based on empirical data, for example from sensor data. During the years, a number of techniques have been proposed to address both specific and broad issues. As done for classical control-theoretical frameworks, we focus our exploration on well-established, standard decision mechanisms. Therefore, we will explore the implementation of a neural network and a reinforcement learning algorithm, as examples of model based and model free techniques, respectively. Some words are spent on the use of a genetic algorithm, without coding this solution.

4.3.1 Neural Networks

We implement an artificial neural network (NN), with the purpose of learning the best policy for control. This means the neural network has to produce the next step control outputs from the current situation, with the purpose of reducing the error between the measured heart rate and the desired one. Every time we have a new sample, we feed that into the network and update its weights according to the gradient of the error we are experiencing.

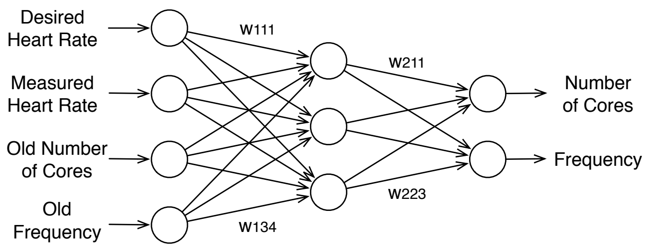


Figure 1: Neural Network topology.

The network is composed by four different input sources, corresponding to the desired heart rate, the actual heart rate and the two control inputs: number of cores and frequency. With three neurons in the (single) hidden layer and two output neurons we learn the relationship between the inputs and the (possibly optimal) control strategy. It is worth stressing that we didn't train the network before launching the experiments and the network itself is trained online, updating the weights according to the experienced error with a gradient descent method. The network topology is shown in Figure 1. The activation function used for the hidden layer is the `atan`, while the output layer uses a linear combination; the learning rate is set to 0.6. In general, to find the best solution, a study on the influence of these two parameters should be conducted; moreover, one may in principle train the network before updating the weights online. These two tasks are deferred to future investigations.

4.3.2 Genetic algorithms

Genetic algorithms are another class of popular and well assessed machine learning techniques. The employment of a genetic algorithm requires selecting a suitable representation scheme for encoding candidate solutions, to define some standard operators, crossover and mutation, and to encode a mathematical function to rate the obtained solutions and to select among them. A possible solution for the proposed case is to synthesize a solution as a couple `{cores, frequency}` and to use as crossover operator the choice of the number of cores from the first solution and the frequency from the second pair. A mutation operator could be an increase or decrease in one of the two quantities. The fitness function for the proposed problem could be some function of the desired heart rate and the expected speedup given to the application by the encoded solution.

Notice that this straightforward idea for GA is not however of particular interest for the proposed problem. Usually, GAs are used when it is hard to explore the space of possible solutions (either because that space is infinite or for other reasons). In the proposed scenario, the number of possible solutions would be 56 and the algorithm would just choose the most suitable solution, without evolution. In this case it makes no sense to implement a GA and pay its overhead. However, there may be different ways to encode the solution that would exploit the potential of the strategy.

4.3.3 Reinforcement learning

As for reinforcement learning solutions, we implement a SARSA (State-Action-Reward-State-Action) algorithm for the problem at hand [26]. The algorithm learns a decision policy for the system. A SARSA implementation interacts with the environment and updates the policy based on actions taken, known as an on-policy learning algorithm.

We define three different states in which the system can be found. In the first one, the heart rate of the monitored application is above the maximum performance threshold. In the second one, the heart rate is between the minimum and maximum levels. In the third one the heart rate is below the minimum. The algorithm automatically finds the optimal policy for these three states. An action is a pair (c, f) where c is the number of cores assigned to the application and f is the frequency set for the computing units of the machine, resulting therefore in 56 different actions.

Using the standard formulation of the algorithm, the Q-value for a state-action is updated by an error, adjusted by the learning rate α (chosen to be 0.5), while the reward discount factor γ is set to 0.2. Q-values represent the possible reward received in the next time step for taking action a in state s, plus the discounted future reward received from the next state-action observation. The new value for the Q-value $Q(s_k, a_k)$ is the following

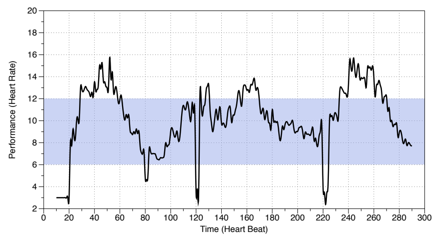
$$Q(s_k, a_k) + \alpha[r_{k+1} + \gamma Q(s_{k+1}, a_{k+1}) - Q(s_k, a_k)] \qquad (5)$$

and in the implementation the Q-values are randomly initialized to provide some variability and exploration of the solution space. It can be shown that under certain boundary conditions SARSA will converge to the optimal policy if all state-action pairs are visited infinitely often.

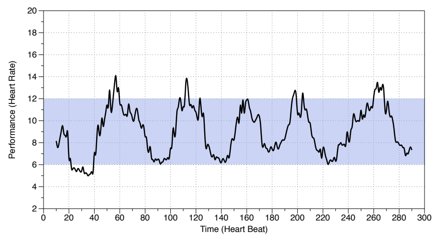
In this example, the algorithm is implemented in such a way that if two different actions provide the same expected reward for the same state, the one that guarantees a higher speedup is chosen.

5. EXPERIMENTAL RESULTS

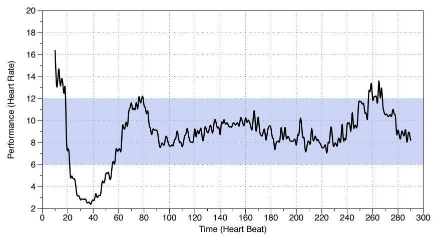
Due to space limitation, a single case study is shown in detail for all the proposed techniques. Then, some aggregate results are presented using five benchmarks from the PARSEC benchmark suite [4]. All the applications used are instrumented with the Application Heartbeat framework [14].



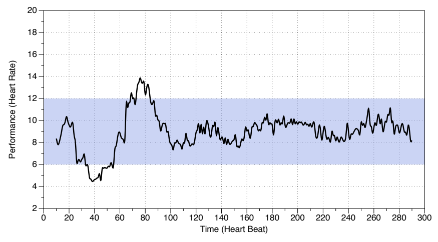
(a) Heuristic



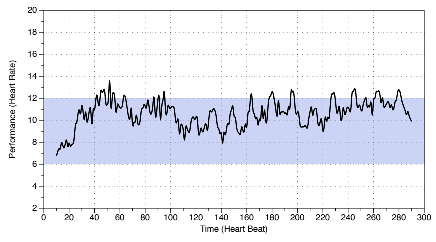
(b) Deadbeat



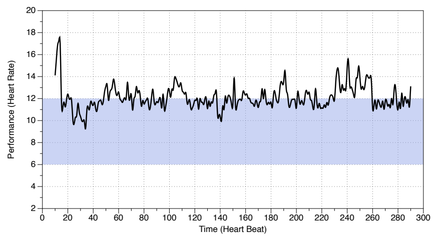
(c) RLS Identification



(d) Kalman Identification



(e) Reinforcement Learning



(f) Neural Network

Figure 2: The heart rate of the application (swaptions) with different solutions.

	ISE	WDP	ISEWDP	Overhead
Heuristic controller	9.82	0.39	4.40	$3.60 \cdot 10^{-9}$
Power controller (Deadbeat)	4.72	0.15	1.40	$20.09 \cdot 10^{-9}$
Power controller (Adaptive with RLS)	5.85	0.19	1.38	$37.10 \cdot 10^{-9}$
Power controller (Adaptive with Kalman)	2.81	0.12	1.86	$44.90 \cdot 10^{-9}$
Reinforcement learning	4.15	0.14	1.77	$89.80 \cdot 10^{-9}$
Neural network	10.81	0.42	7.27	$1410.00 \cdot 10^{-9}$

Table 1: Integral of the Squared Error (ISE), percentage of data point outside the specified thresholds (WDP), integral of the squared error for wrong data points (ISEWDP), technique overhead (expressed in seconds) for the swaptions test case (plots are shown in Figure 2); lower is better.

All experiments are run on a Dell PowerEdge R410 server with two quad-core Intel Xeon E5530 processors running Linux 2.6.26. The processors support seven power states with clock frequencies from 2.4 GHz to 1.6 GHz. The `cpufrequtils` package enables software control of the clock frequency, while the `taskset` Linux command allows to set the affinity mask of the controlled application to specific computing units (CPUs). The two actuators used are therefore the machine frequency and the number of cores to be assigned to the given instrumented application.

5.1 Swaptions

The in-depth case study manages performance of the `swaptions` application, which uses the Heath-Jarrow-Morton framework to price a portfolio of swaptions and employs Monte Carlo simulation to compute the prices. When a swaption is processed, the application makes an API call to signify a heartbeat. Over time, the intervals between heartbeats provide information about progress for the purpose of application auto-tuning and/or externally-driven optimization. Moreover the Application Heartbeat framework allows users to set minimum and maximum performance targets. For swaptions these targets are set to 6 and 12 heartbeats per second, respectively. The application processes 300 swaptions and performs 100000 simulations for each of them. `swaptions` is run multithreaded with 8 threads.

For each invocation of swaptions run we collect 300 data points (corresponding to the 300 heartbeats emitted) and disregard the first 10 and last 10 data points, as performance at these values is dominated by initialization and finalization. We present some plots to depict the application behavior using the different decision mechanisms. Moreover, Table 1 presents some useful comparison numbers. The first column of the table shows the Integral of the Squared Error (ISE). Suppose we define a target at $\mathcal{T}$ heartbeats per second (the average between the minimum and the maximum threshold, respectively $\mathcal{T}_{min}$ and $\mathcal{T}_{max}$), the ISE is defined as

$$ISE = \frac{1}{n}\sum_{k=1}^{n}[\mathcal{T} - hr(k)]^2 \tag{6}$$

where k represent a single heartbeat and n is the number of considered points, in this case 280. The second column presents the percentage of data points that lays outside the desired performance range, i.e. those with a wrong control. The third column contains the integral of the squared error considering just the points with wrong control

$$ISEWDP = \frac{1}{n}\sum_{k=1}^{n}[1 - I_{\mathcal{T}_{min}<hr(k)<\mathcal{T}_{max}}][\mathcal{T} - hr(k)]^2 \tag{7}$$

where $I_{\mathcal{T}_{min}<hr(k)<\mathcal{T}_{max}}$ a Borel set, i.e., a function that is 1 in the specified interval and 0 elsewhere (in this case 1 if

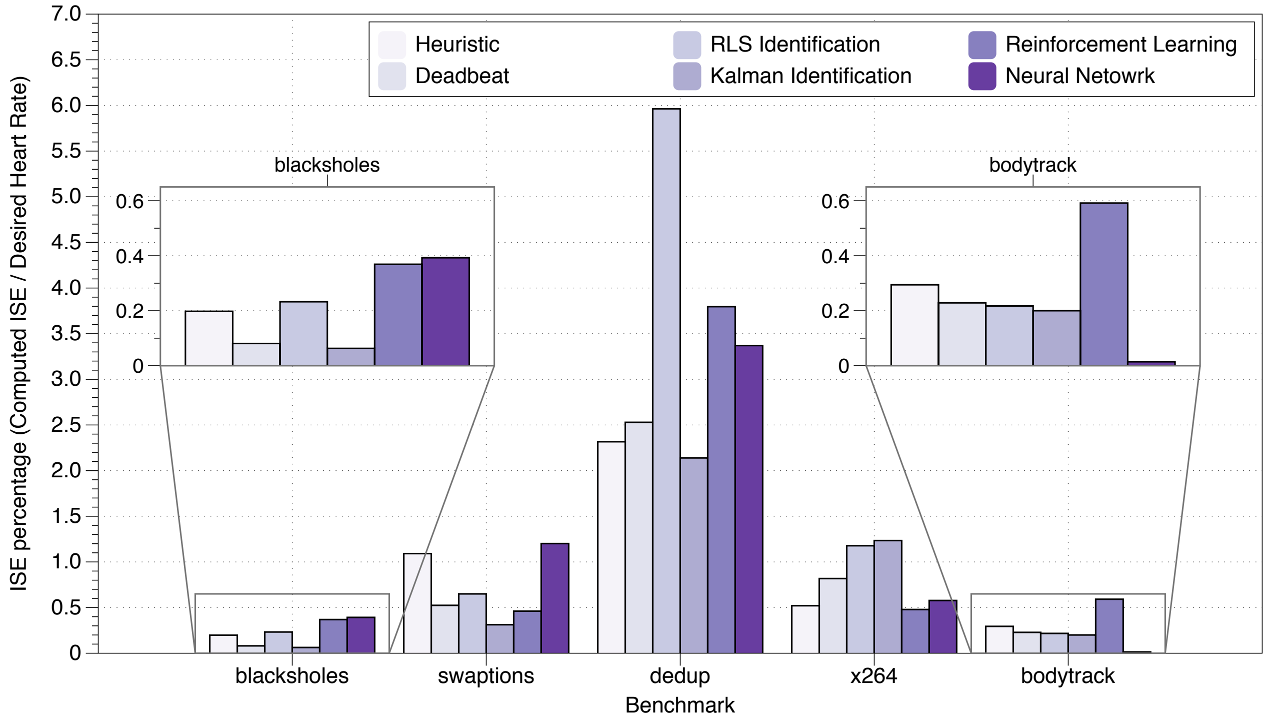


Figure 3: Summary ISE results (lower is better).

the measured heart rate belongs to the interval from $\mathcal{T}_{min}$ to $\mathcal{T}_{max}$). The last column proposes a measure of the technique overhead, i.e., an average of the number of seconds necessary to execute the decision mechanism at each iteration.

Figure 2 depicts the data for the proposed solution. Table 1 reports some numerical data. In Figure 2(a) the heuristic decision mechanism is shown at work. The power aware scheduler described in [15] obtains the results of figure 2(b). Figure 2(c) presents the results obtained with the same controller empowered with the Recursive Least Square identification mechanism while Figure 2(d) shows the Kalman filter solution. Figure 2(f) shows the results obtained using the neural network. Apparently, `swaptions` is poorly controlled with the proposed neural network. However, the more extensive benchmark study reported in the following shows that there are other applications where the NN outperforms other approaches. The tendency of the SARSA RL algorithm to choose the most powerful state is quite evident in Figure 2(e), as the measured heart rate is consistently in the upper part of the desired range.

5.2 Benchmark campaign results

This section presents aggregate results from an extensive test campaign. Five different PARSEC benchmark are used as applications, these being blacksholes (options pricing, 1600 heartbeats emitted), bodytrack (tracking of people in a video, 260 heartbeats emitted), dedup (compression with data deduplication, 300 heartbeats emitted), swaptions and x264 (video encoding, 500 heartbeats emitted). For these benchmarks, a tighter performance range is used compared to the previous example. The desired heart rate for blacksholes is set between 13 and 14 heartbeats per second (h/s), bodytrack is given a performance target between 3 and 4 h/s, dedup's thresholds are respectively 53 and 54 h/s and for this second swaptions run, the heart rate was constrained to be between 9 and 10 h/s. The desired range is wider for x264, since its modified version proposed in [15] is supposed to span from 25 and 35 beats, therefore maintaining a target of 30 frames per second.

For each benchmark, we again measure ISE, WDP, and ISEWDP. Figure 3 shows the ISE for these benchmarks. Every bar depicts a percentage of error, the meaning of a single bar being the percentage of average fluctuation with respect to the set point. For example, a value of 2 means that, on average, the ISE is twice as big as the desired heart rate value. Figure 4 depicts the percentage of data points that are not in the desired performance interval for each test case with each of the different decision methods.

Notice that `dedup` is especially hard to control as there is little regularity in the heartbeats emission and the feedback mechanism is difficult to exploit. In that case, apparently, almost all the points lay outside the desired range independently of the adopted decision method, but the use of an adaptive control system based on a Kalman filter seems to limit the errors as much as possible. The Kalman filter solution behaves better than the other proposals for `blacksholes` and `swaptions` as well, where both the percentage of wrong data points and the relative error are smaller than with all the other techniques.

On the contrary, the most powerful solution to control the performance of `bodytrack` seems to be the implemented neural network, for which all the points are inside the specified range, with a limited error. The reinforcement learning solution's performance is highly variable between benchmarks; this is probably partially due to its random initialization.

It is worth noticing, however, that for `x264`, the heuristic

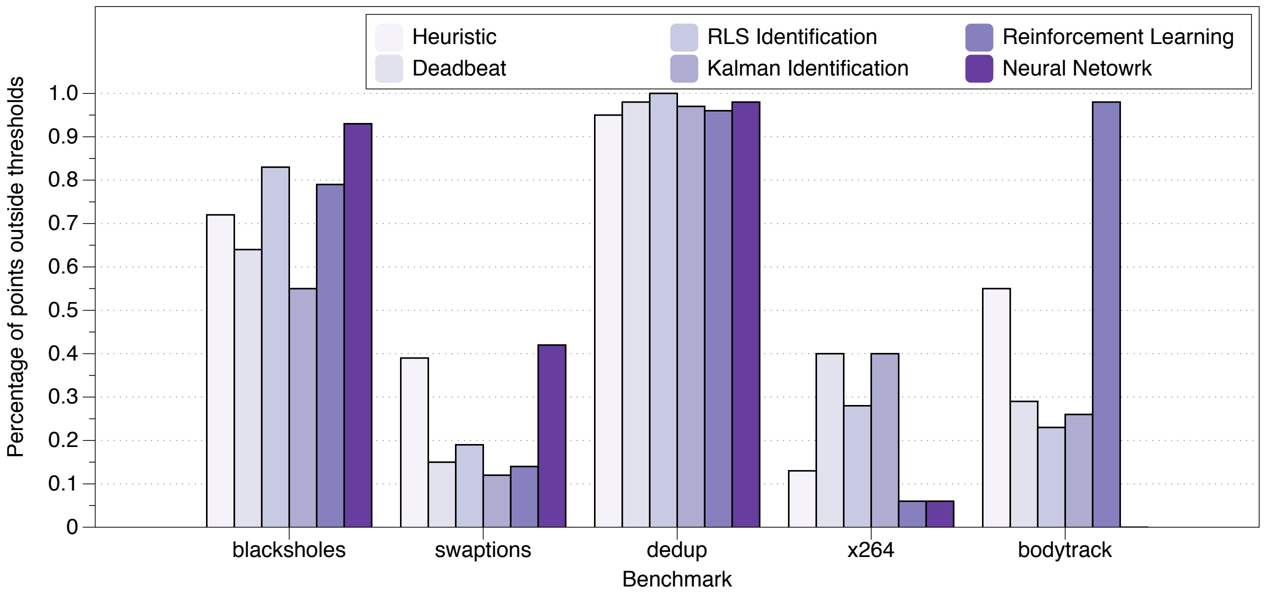


Figure 4: Summary WDP results (lower is better).

solution and the two machine learning-based ones outperform the control-theoretical methods. This seems to imply that the model assumption in this particular case are wrong. One reason for this may be that the dynamic between the measured heart rate and the past one is more complex than what is modeled for the control solutions, a matter to be further studied in the future.

6. CONCLUSIONS

In this paper, we proposed a comparison of some state-of-the-art techniques for building decision making mechanisms in autonomic computing systems. The Application Heartbeat framework was used to provide the necessary sensors to develop different solutions, spanning from machine learning techniques to heuristic and control-theoretical systems.

We focused on decisions related to the problem of self-optimization, where for an application a range of desired operating conditions is defined, and the decision making mechanism needs to provide the necessary resources (and possibly no more) to meet the requirements. A single case study was shown in detail with all the proposed techniques, to allow a meaningful comparison, and some numerical data are proposed as well as plots depicting the behavior of the system. This case study is chosen among the tests conducted with all the PARSEC benchmark applications, for which some aggregate results are provided.

Our results indicate that the best decision method can vary depending on the specific application to be optimized; however, we note that the adaptive control system based on a Kalman filter tends to produce good performance for a range of applications. Thus, we conclude that for specific systems with a narrow range of target applications developers are probably best off testing a number of decision mechanisms, but for systems with a broad or unknown range of target applications, adaptive control may provide the best general solution. Future works will consider more advanced methodologies and extend the present comparison. Moreover, we will summarize the results obtained with all the software applications, to the extent of generalization.

Acknowledgment

We'd like to thank the Rocca Foundation for the support through the Progetto Rocca.

7. REFERENCES

[1] An architectural blueprint for autonomic computing. Technical report, June 2006.

[2] D. Albonesi, R. Balasubramonian, S. Dropsho, S. Dwarkadas, E. Friedman, M. Huang, V. Kursun, G. Magklis, M. Scott, G. Semeraro, P. Bose, A. Buyuktosunoglu, P. Cook, and S. Schuster. Dynamically tuning processor resources with adaptive processing. *Computer*, 36:49–58, December 2003.

[3] K. Åström and T. Hägglund. *Advanced PID Control*. ISA - The Instrumentation, Systems, and Automation Society, Research Triangle Park, NC, 2005.

[4] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, October 2008.

[5] R. Bitirgen, E. Ipek, and J. Martinez. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 41, pages 318–329, Washington, DC, USA, 2008. IEEE Computer Society.

[6] P. Bodík, R. Griffith, C. Sutton, A. Fox, M. Jordan, and D. Patterson. Statistical machine learning makes automatic control practical for internet datacenters. In *Proceedings of the 2009 conference on Hot topics in*

cloud computing, HotCloud'09, pages 12–12, Berkeley, CA, USA, 2009. USENIX Association.

[7] D. Breitgand, E. Henis, and O. Shehory. Automated and adaptive threshold setting: Enabling technology for autonomy and self-management. In *Proceedings of the Second International Conference on Automatic Computing*, pages 204–215, Washington, DC, USA, 2005. IEEE Computer Society.

[8] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, SOSP '01, pages 103–116, New York, NY, USA, 2001. ACM.

[9] M. Chen, A. Zheng, J. Lloyd, M. Jordan, and E. Brewer. Failure diagnosis using decision trees. In *Proceedings of the International Conference on Autonomic Computing*, pages 36–43, May 2004.

[10] R. Dantu, J. Cangussu, and S. Patwardhan. Fast worm containment using feedback control. *IEEE Transactions on Dependable and Secure Computing*, 4(2):119–136, April-June 2007.

[11] P. Geurts, I. E. Khayat, and G. Leduc. A machine learning approach to improve congestion control over wireless computer networks. In *Proceedings of the 4th IEEE International Conference on Data Mining*, pages 383 – 386, November 2004.

[12] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury. *Feedback Control of Computing Systems*. Wiley, September 2004.

[13] J. Hellerstein, V. Morrison, and E. Eilebrecht. Applying control theory in the real world: experience with building a controller for the .net thread pool. *SIGMETRICS Performance Evaluation Review*, 37(3):38–42, 2009.

[14] H. Hoffmann, J. Eastep, M. Santambrogio, J. Miller, and A. Agarwal. Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments. In *Proceeding of the 7th international conference on Autonomic computing*, ICAC '10, pages 79–88, New York, NY, USA, 2010. ACM.

[15] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, and A. Agarwal. SEEC: A Framework for Self-aware Management of Multicore Resources. Technical Report MIT-CSAIL-TR-2011-016, CSAIL, MIT, March 2011.

[16] J. Kephart. Research challenges of autonomic computing. In *Proceedings of the 27th international conference on Software engineering*, ICSE '05, pages 15–22, New York, NY, USA, 2005. ACM.

[17] J. Kephart and D. Chess. The vision of autonomic computing. *Computer*, 36:41–50, January 2003.

[18] X. Liu, X. Zhu, S. Singhal, and M. Arlitt. Adaptive entitlement control of resource containers on shared servers. In *Proceeding of the 9th IFIP/IEEE International Symposium on Integrated Network Management*, pages 163–176, May 2005.

[19] L. Ljung. *System Identification: Theory for the User*. Prentice Hall PTR, December 1998.

[20] C. Lu, Y. Lu, T. Abdelzaher, J. Stankovic, and S. Son. Feedback control architecture and design methodology for service delay guarantees in web servers. *IEEE Transactions on Parallel and Distributed Systems*, 17(7), 2006.

[21] M. Maggio, H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva. Controlling software applications via resource allocation within the heartbeats framework. In *Proceeding of the 49th international conference on decision and control*, Atlanta, USA, 2010. IEEE Control.

[22] J. Martinez and E. Ipek. Dynamic multicore resource management: A machine learning approach. *IEEE Micro*, 29:8–17, September 2009.

[23] A. O'Dwyer. *Handbook of PI And PID Controller Tuning Rules*. Imperial College Press, second edition, February 2006.

[24] W. Pan, D. Mu, H. Wu, and L. Yao. Feedback Control-Based QoS Guarantees in Web Application Servers. In *Proceedings of the 10th International Conference on High Performance Computing and Communications*, pages 328–334, September 2008.

[25] A. Ramirez, D. Knoester, B. Cheng, and P. McKinley. Applying genetic algorithms to decision making in autonomic computing systems. In *Proceedings of the 6th international conference on Autonomic computing*, ICAC '09, pages 97–106, New York, NY, USA, 2009. ACM.

[26] R. Sutton and A. Barto. Reinforcement learning: an introduction. page 322, 1998.

[27] G. Tesauro. Reinforcement learning in autonomic computing: A manifesto and case studies. *IEEE Internet Computing*, 11:22–30, January 2007.

[28] G. Tesauro, N. Jong, R. Das, and M. Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *Proceedings of the 2006 IEEE International Conference on Autonomic Computing*, pages 65–73, Washington, DC, USA, 2006. IEEE Computer Society.

[29] C. Tsai, Y. Hsu, C. Lin, and W. Lin. Review: Intrusion detection by machine learning: A review. *Expert Syst. Appl.*, 36:11994–12000, December 2009.

[30] P. Ulam, A. Goel, J. Jones, and W. Murdoch. Using model-based reflection to guide reinforcement learning. In *Proceedings of the 2005 IJCAI Workshop on Reasoning, Representation and Learning in Computer Games*, pages 1–6, 2005.

[31] J. Wildstrom, E. Witchel, and R. Mooney. Towards self-configuring hardware for distributed computer systems. In *Proceedings of the Second International Conference on Automatic Computing*, pages 241–249, Washington, DC, USA, 2005. IEEE Computer Society.