Comparison of Decision Making Strategies for Self-Optimization in Autonomic Computing Systems

Martina Maggio, Department of Automatic Control, Lund University, Sweden
Henry Hoffmann, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA
Alessandro V. Papadopoulos, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy
Jacopo Panerati, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy
Marco D. Santambrogio, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy
Anant Agarwal, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA
Alberto Leva, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy

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. 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. Our results indicate that the most suitable decision mechanism can vary depending on the specific test case but adaptive and model predictive control systems tend to produce good performance and may work best in a priori unknown situations.

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

Additional Key Words and Phrases: Decision mechanisms, Comparison, Design approaches

ACM Reference Format:
Maggio, M., Hoffmann, H., Papadopoulos, A.V., Panerati, J., Santambrogio, M.D., Agarwal, A., Leva, A. 2011. Comparison of Decision Making Strategies for Self-Optimization in Autonomic Computing Systems. ACM Trans. Autonom. Adapt. Syst. 9, 4, Article 39 (March 2010), 32 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

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 [Kephart

This work was partially funded by the U.S. Government under the DARPA UHPC program. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government. Also, this work was partially supported by the Swedish Research Council through the LCCC Linnaeus Center.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

2005]. 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 [IBM 2006]. 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 [Kephart and Chess 2003].

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 [Hellerstein et al. 2009]. 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. Specifically, we examine a problem of resource allocation and create an experimental framework where the available observation and action mechanisms are fixed, but the decision mechanism can be changed. This framework allows us to use different decision making techniques and compare their ability to achieve a desired performance target. 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 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 [Bienia et al. 2008], and the presentation of 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 [Lu et al. 2006], the corresponding engineering tools and processes are far from being 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, stability and convergence time guarantees for the *online* system. 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*.

This paper significantly extends previous results on the same topic [Maggio et al. 2011] and its remainder is organized as follows. Section 2 describes the problem of self-optimization applied to resource allocation, and shows the techniques qualitatively compared for that challenge. In Section 3 the mentioned solutions are described in detail, from the basic idea to the implementation, and some experimental results are provided in Section 4. Finally, Section 5 concludes the paper.

2. DESIGN FOR SELF-OPTIMIZATION OF RESOURCE ALLOCATION

An autonomic computing system may be built with different goals, but its essence is self-management [Kephart and Chess 2003]. Four main aspects of self-management emerge in the literature: self-configuration [Wildstrom et al. 2005], self-protection [Tsai et al. 2009], self-healing [Breitgand et al. 2005], and self-optimization, which is the focus of this paper.

Several techniques have been used to synthesize decision mechanisms for self-optimizing computing systems. As a notable example, [Ramirez et al. 2009] 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. Assigning a resource to an application is a cost for the system: it consumes power, it is consumed and it cannot be used by other applications at the very same time. To clarify, this case is different for example from the one in which an algorithm has to be selected (see e.g. [Ansel et al. 2011]), that optimize for performance given an architecture. In that case, the algorithm choice could be seen by the cost angle (it may consume more time and power) but also simply from the performance point of view. In this work, we want to balance the use of resources and their cost for the system, therefore constraining the system to use only the necessary resources to accomplish the applications goals.

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 [IBM 2006]. 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 [Bitirgen et al. 2008], the authors 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.

Section 2.1 introduces some commonly used techniques, that will be here used as the basis for the comparison, while Section 2.2 discusses a qualitative comparison between the mentioned solutions. The rest of the paper develops and implements the proposed strategies and compares them.

2.1. 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 [Chase et al. 2001] 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.

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 or 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.* [Pan et al. 2008] 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 designed 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, [Liu et al. 2005] 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 with exogenous input 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. Artificial Neural Networks (ANN) are often useful to build a model of the world for control purposes, see again [Bitirgen et al. 2008]. ANN 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. [Ramirez et al. 2009] 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. Also, 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 [Bodík et al. 2009].

Model-free machine learning solutions do not require a model of the system. Notable examples are some Reinforcement Learning (RL) algorithms like Q-Learning and SARSA. It is worth noticing that a recent research trend is to complement RL solution with a model definition [Tesauro 2007; Tesauro et al. 2006]. According to [Martinez and Ipek 2009], 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 [Sutton and Barto 1998] and complementing them with a model of the solution space may decrease it [Ulam et al. 2005].

2.2. 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: (i) the presence or absence of a model, (ii) the extent to which they provide analytic performance guarantees, (iii) their ability to handle unexpected conditions, and (iv) 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 an ANN, 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 3.4.1.

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 [Sutton and Barto 1998].

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 ANN, 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.

Table I. Summary of qualitative comparison

	Performance Guarantees	**Low Overhead**	**Model not Needed**	**Reaction to Unseen**
Heuristic		✓	✓	
Standard Control	✓	✓		
Advanced Control	✓			✓
Neural Network			✓[1]	✓
Reinforcement Learning			✓	✓

[1] It is worth stressing that in this case we say the neural network does not need a model, since it is not necessary to specify the dynamic model of the system, but to choose the structure of the network, usually easier and connected to intuition more than to formal analysis.

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. These characteristics are summarized in Table I.

3. IMPLEMENTATION FOR DECISION MAKING IN SELF-OPTIMIZING SYSTEMS

To compare different decision mechanisms, we focus on a single problem: that of creating a runtime system that allocates resources to allow an application to achieve a target performance. This autonomic runtime system must be capable of observation, action, and decision. Since we are interested in comparing decision mechanisms only, we fix the observation and action phases and develop the framework in flexible way such that different decision mechanisms may be "swapped in." This section describes the experimental framework and the decision mechanisms under test, starting with the observation phase, then the action phase, and finally each of the decision mechanisms is detailed in turn.

3.1. Observation

In our runtime framework, feedback is provided directly by applications using the Application Heartbeats API [Hoffmann et al. 2010]. This API provides a general mechanism for observing application performance and goals using the abstraction of a heart beat. The application emits heart beats at important parts of the code (e.g., once a frame for a video application) performance goals are then expressed in terms of a desired heart rate per second or a desired latency between specially tagged heart beats.

The Heartbeats framework has been applied to applications from the PARSEC benchmark suite [Bienia et al. 2008] and these applications have been found to divide into two groups based on the heart beat signal [Hoffmann et al. 2011]. The first group is applications with low variance in the signal (i.e., their performance is steady from heart beat to heart beat) and those with high variance (i.e., the performance varies tremendously from heart beat to heart beat). The varying needs of different benchmarks mean these applications provide a good set of tests for our resource allocation system.

3.2. Action

In this specific work two different knobs, or actions, are used, although the same techniques described hereafter apply to different configurations. In the present configuration it is possible to suggest to the operating system how many cores an application can use and to change the frequency of those cores. By "suggesting to the operating system" we mean that we can modify the affinity mask of the processes composing an application, using the `taskset` Unix command, forcing it to run on a subset of the hardware capabilities. Being the application not parallel, however, could invalidate our actuation mechanism, since we could specify a number of cores that would be kept idle or be run other processes onto. The frequency change is supported by the `cpufrequtils` package.

In the following we refer to c as number of cores currently allocated to the running application and to f as the actual frequency of these cores, hr as the current heart rate provided by the application, hr_{min} as the minimum threshold and hr_{max} as the maximum heart rate value the application should provide. We also define the desired heart rate $hr^{\circ} = 0.5(hr_{max}+hr_{min})$. Moreover, c_{min} and c_{max} are the minimum and maximum number of cores that can be given to the application and f_{min} and f_{max} the minimum and the maximum frequency that can be set for those cores. In the remainder of this section, we describe each of the evaluated decision mechanisms.

3.3. Heuristic

Heuristic methods are often used to speed up the search for a solution whenever an exhaustive search of the solution space is not affordable. The technique ignores the possible incorrectness of the solution.

The values of every knob can be numerically ordered for increased capabilities. Whenever it is not possible to provide a sort function, the different values could be just listed. An heuristic method could measure the actual performance and compare it with the target levels. If the actual performance value is below the minimum threshold the heuristic chooses a dimension, corresponding to a single knob value, and performs a move increasing the amount of resources allocated to the application. With the same *rationale*, if the actual performance value is above the maximum threshold, the heuristic selects a knob and diminishes its value.

The knob selection can be provided with a priority based mechanism that changes actuator values that are considered more effective on the application first and moves to less effective actuators after or a random selection or other techniques. Intuitively,

if the knob values can not be ordered, the search could take much more time with respect to the amount of time it would take in the opposite case. According to the *heuristic* nature of this solution, no guarantees are given that the system will enter the best state to attain the performance goal with the minimum amount of resources possible.

3.3.1. Single actuator heuristic. First, we suppose there is a single knob on the machine, being this the number of cores allotted to the single application processes. In this case the heuristic setup is straightforward and the system can be found in $1 + c_{max} - c_{min}$ possible states. Formally, the number of cores to be allotted at the k-th step of the algorithm execution $c(k)$ is given by

$$c(k) = \begin{cases} \min(c_{max}, c(k-1)+1) & hr(k) < hr_{min} \\ c(k-1) & hr_{min} \leq hr(k) \leq hr_{max} \\ \max(c_{min}, c(k-1)-1) & hr(k) > hr_{max} \end{cases} \quad (1)$$

Informally, whenever the heart rate signal we use as a sensor of the application progresses is below the minimum threshold, the number of cores is increased, limiting this value to c_{max}. If the signal is above the maximum threshold, the value is decreased, to a minimum of c_{min} cores allocated to the running application.

3.3.2. Multiple actuators heuristic. Second, we add to the previous system the frequency knob. In so doing, we need to select how to explore the solution space. We define a priority on the two actuators, assuming that frequency changes are more invasive on the system in its entirety, and should be done less frequently and we use the heuristic formalized in Algorithm 1.

ALGORITHM 1: Multiple actuators (cores and frequency) heuristic solution

Require: hr_{min}, hr_{max}, c_{old}, c_{min}, c_{max}, f_{old} f_{min}, f_{max}, f_{step}
Ensure: c, f
  $hr \leftarrow$ measured application heart rate
  if $hr_{min} \leq hr \leq hr_{max}$ **then**
    $c = c_{old}$
    $f = f_{old}$
  else if $hr > hr_{max}$ **then**
    if $c = c_{min}$ **then**
      $f = \max(f_{min}, f_{old} - f_{step})$
      $c = c_{max}$
    else
      $c = c_{old} - 1$
      $f = f_{old}$
    end if
  else if $hr < hr_{min}$ **then**
    if $c = c_{max}$ **then**
      $f = \min(f_{max}, f_{old} + f_{step})$
      $c = c_{min}$
    else
      $c = c_{old} + 1$
      $f = f_{old}$
    end if
  end if

This means that whenever the number of cores reaches c_{min} and the system capabilities need to be diminished, the clock speed of the cores is diminished. If the system

resources are to be augmented and the number of cores is c_{max} the frequency of those cores is augmented. Another possibility could be to randomly choose the actuator to act on, which probably makes more sense when a large number of knobs are available.

3.4. Control-theoretical

Autonomic and self-optimization capabilities have been added to computing systems via feedback control [Hellerstein et al. 2004]. Different approaches were developed in the literature up to date to solve a variety of specific problems, although a general lack of generalization could be noticed. An attempt to generalize these solutions was proposed [Zhang et al. 2002], however its success was limited, probably due to some of the limitations discussed in [Hellerstein 2010].

For the proposed problem, a preliminary control result was published [Maggio et al. 2010], that addresses the number of cores as a single knob and uses a basic control scheme. In the following, we extend these results proposing a more flexible model and we synthesize a controller that is much more general and could in principle manage any kind and combination of resources.

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 nonlinear systems, discrete event models, Petri nets, 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 and the set point is feasible.

3.4.1. 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 heart beat is given as

$$hr(k+1) = \frac{s(k)}{w(k)} + \delta hr(k) \tag{2}$$

where $s(k)$ 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 heart beats when the system is in the state that provides the lowest possible speedup. In this model, a speedup is applied to the system and it 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 $\delta hr(k)$ 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.

3.4.2. The control synthesis. 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 [Åström and Hägglund 2005; O'Dwyer 2006].

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

$$\frac{R(z)}{\bar{R}(z)} = \mu \frac{z - z_1}{(z - p_1)(z - p_2)} \tag{3}$$

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

$$\frac{R(z)}{\bar{R}(z)} = \frac{R(z)C(z)}{1 + R(z)C(z)} \tag{4}$$

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 k:

$$\begin{aligned} s(k) \;=\; & F \cdot [As(k-1) + Bs(k-2) + \\ & + Ce(k)w(k) + De(k-1)w(k-1)] \end{aligned} \tag{5}$$

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 and are

$$\begin{aligned} A &= - \; [-p_1 z_1 - p_2 z_1 + p_1 p_2] \\ B &= - \; [p_2 z_1 + p_1 z_1 - z_1 - p_1 p_2] \\ C &= + \; [p_2 - p_1 p_2 + p_1 - 1] \\ D &= + \; [p_1 p_2 - p_2 - p_1 + 1]\, z_1 \\ F &= + \; [z_1 - 1]^{-1}. \end{aligned} \tag{6}$$

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 the technical report [Hoffmann et al. 2011].

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.

3.4.3. Single actuator basic control. The speedup signal $s(k)$ can be translated into the value of a single actuator, defining a relationship between the value of the knob and the speedup. In the case of the number of cores we could for example assume that the application speeds up linearly with $c(k)$. Therefore, in the test hardware, the maximum speedup that can be applied is c_{max} while the minimum one is c_{min}.

3.4.4. Multiple actuators basic control. Different maps from the computed $s(k)$ to be applied on the system and the couple $c(k), f(k)$ can be defined. We choose to set

$$\hat{s}(k) = c(k)\frac{f(k)}{f_{min}} \tag{7}$$

where $\hat{s}$ is the estimated speedup given in the state $c(k), f(k)$. The minimum speedup applicable in the system is therefore c_{min} while the maximum one is $c_{max}f_{max}/f_{min}$. Therefore, given the speedup value computed by the controller, we map that value into the couple of values for our knobs.

3.4.5. Single actuator adaptive control. 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 a Recursive Least Squares (RLS) filter to estimate the workload value [Ljung 1998] and turn the standard Deadbeat controller into an adaptive one. The adaptive controller computes the desired speedup value, which is subsequently translated in the control signal with the same rationale of the basic methodology.

3.4.6. Multiple actuators adaptive control. Adaptability is not influenced in this case by the number of actuators. 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 and their clock speed) and the performance metric, to build a more sophisticated controller. One could for example identify regions of the solution space where adding a single core does not influence the application performance and more than one computing unit is needed to speedup the application. At the same time, one may see that adding cores and changing their frequencies does not influence the application, therefore discovering it can be for example memory bounded.

3.4.7. A more complex model. The considered model, however, does not capture all the behavior of the application to be controlled. An example of this non captured behavior could be the presence of nonlinearities in the actuation mechanism. For example, usually, the assumption that the number of cores linearly increase the speedup, and therefore the heart rate of the application, does not hold. Hence, a more complex model can be designed in order to extend the potentialities of the control approach.

The model $\mathcal{M}$ of a generic application is still very simple and can be expressed as

$$\mathcal{M} : \begin{cases} hr(k) = p \cdot hr(k-1) + (1-p) \cdot \hat{s}(k-1) \\ 0 \leq p < 1 \end{cases} \tag{8}$$

with the parameter p that, in general, tends to be 0 because the dynamics of the application are very fast, leading to a "pure-delay" system.

As already stated, the behavior of the system in terms of heart rate can be affected by assigning to the application very different resources that we can, generically, call as actuator. The model we propose considers a set a of N_a different knobs. Therefore, to consider a variable number of knobs, the relation (7) is generalized. Denoting with a_i a generic actuator the effective speedup given to the application is expressed as

$$\hat{s}(k) = \prod_{i=1}^{N_a} \left[k_i \left[a_i(k) \right]^{\alpha_i} + o_i \right]. \tag{9}$$

where each actuator contributes according to some parameters, namely k_i, α_i, and o_i. We can thus define the vector $\vartheta \in \Re^{3N_a}$ which contains all these parameters for every i in the interval $1 \ldots N_a$, describing the "profile" of the application. In general, those parameters are time-varying depending, for instance, on the portion of code the application is executing. This problem is strictly linked to the time-varying workload discussed in Section 3.4.5, and will be addressed also in this solution with an adaptive mechanism. From equation (9), a block $\mathcal{A}_\vartheta$ can be defined. This block stands for a nonlinear and time-varying relation that maps the manipulated variables (the knobs values) to the real speedup experienced by the application as a consequence of those values.

This new model, more complex, is able to capture dynamics that were impossible to be described with the previously introduced one. On the other hand, it is harder to design an effective control law for it.

3.4.8. Single actuator model predictive control. Model Predictive Control (MPC) can be used to provide a control strategy for the mentioned problem. In MPC, a model of the system is available, and the control system keeps that model updated. At each control step, the controller chooses the next action to be performed so that the discrepancy between the desired behavior and that forecast with the model be minimized. Since this is done at every step, quite loose hypotheses on the model accuracy are enough to ensure that the iterative process just sketched out converges and drives the system to the desired behavior. MPC is thus a very widely used technique exactly when complex and/or time-varying dynamics are encountered and only simple models can be reliably devised, theoretical explanation and many examples can be found in [Camacho and Bordons 2004], to which the interested reader is referred.

The control scheme devised for the more complex model is depicted in Figure 1. First, consider the case in which there is only the number of cores as a knob, the speedup signal becomes

$$\mathcal{A}_\vartheta : \hat{s}(k) = k_c \left[c(k) \right]^{\alpha_c} + o_c \tag{10}$$

However, ϑ is unknown and must be estimated in order to design an actuation block $\mathcal{A}_{\hat{\vartheta}}^{-1}$ which is able to map the control signal to the number of cores to be allotted to the application. Notice that if $\hat{\vartheta} = \vartheta$ then $s = \hat{s}$, and the speedup computed from the controller is the actual speedup given to the application.

The predictive control law is to be designed now. Denoting with y the output of the controlled system, i.e., the heart rate, the r-step ahead predictor of the model $\mathcal{M}$, which

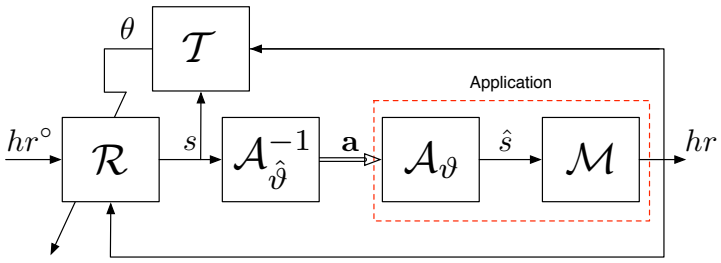


Fig. 1. Adaptive Predictive control loop. The application is evidenced with the dashed red line.

is an Auto Regressive with eXogenous input [Bittanti and Picci 1996], $ARX(n_a, n_b)$, is

$$\hat{y}(k+r|k) = W_r(z)y(k) + H_r(z)\Delta s(k). \tag{11}$$

Hence, denoting with N_p the prediction horizon – in our experiments we used a value of 2 – we can define the vector of r-step-ahead predictors of the output, with $r = 1, \ldots, N_p$, and the vector that contains the last n_a outputs, i.e., respectively

$$\begin{aligned} \hat{\mathcal{Y}}' &= [\hat{y}(k+1|k) \;\; \hat{y}(k+2|k) \;\; \ldots \;\; \hat{y}(k+N_p|k)]\,, \\ \mathcal{Y}' &= [y(k) \;\; y(k-1) \;\; \ldots \;\; y(k-n_a+1)]\,. \end{aligned} \tag{12}$$

Denoting with W the coefficient matrix of the N_p predictors from the past outputs to the predicted output, of dimensions $N_p \times n_a$

$$W' = \left[W'_1 \;\; W'_2 \;\; \ldots \;\; W'_{N_p}\right] \tag{13}$$

with ΔS_o the vector containing the n_b past values of the speedup signal and with ΔS the $N_p - 1$ future (unknown) values of the speedup signal:

$$\begin{aligned} \Delta\mathcal{S}'_o &= [\Delta s(k-n_b) \;\; \Delta s(k-(n_b-1)) \;\; \ldots \;\; \Delta s(k-1)]\,, \\ \Delta\mathcal{S}' &= [\Delta s(k) \;\; \Delta s(k+1) \;\; \ldots \;\; \Delta s(k+Np-2)]\,. \end{aligned} \tag{14}$$

Finally, defining with Φ_o, of dimensions $N_p \times n_b$, and Φ, of dimensions $N_p \times N_p - 1$, the coefficient matrices of the predictors from the speedup to the predicted heart rate become

$$\Phi_o = \begin{bmatrix} h_{n_b} & h_{n_b-1} & \ldots & h_1 \\ h_{n_b+1} & h_{n_b} & \ldots & h_2 \\ \vdots & \vdots & \ddots & \vdots \\ h_{n_b+N_p-1} & h_{n_b+N_p-2} & \ldots & h_{N_p} \end{bmatrix}, \; \Phi = \begin{bmatrix} 0 & 0 & \ldots & 0 \\ h_1 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ h_{N_p-1} & h_{N_p-2} & \ldots & h_1 \end{bmatrix}. \tag{15}$$

We can write the predicted output as $\hat{\mathcal{Y}} = W\mathcal{Y} + \Phi_o\Delta\mathcal{S}_o + \Phi\Delta\mathcal{S}$. The optimal control signal is computed minimizing the cost function J defined as

$$J = (\mathcal{Y}^\circ - \hat{\mathcal{Y}})'\mathcal{Q}(\mathcal{Y}^\circ - \hat{\mathcal{Y}}) + \Delta\mathcal{S}'\overline{\mathcal{R}}\Delta\mathcal{S} \tag{16}$$

where $\mathcal{Q}$ and $\overline{\mathcal{R}}$ are diagonal matrix of weights that must be chosen when designing the control law. Similar applications would benefit similarly from setting similar weights. Roughly speaking, see again [Camacho and Bordons 2004] for details, the elements of $\mathcal{Q}$ are proportional to the relative importance of the error components, thus comparatively weighing each objective versus the others. $\overline{\mathcal{R}}$ is conversely a control penalty: the higher its components, the more reluctant the system will be to exert a control action on the corresponding component (if $\overline{\mathcal{R}}$ the so called cost-less control MPC is obtained

since the controller assumes that control actions cost nothing). $\mathcal{Y}^\circ$ is a column vector of length N_p where each element is equal to the set point. Therefore

$$\Delta\mathcal{S} = (\Phi'\mathcal{Q}\Phi + \overline{\mathcal{R}})^{-1}\Phi'\mathcal{Q}(\mathcal{Y}^\circ - W\mathcal{Y} - \Phi_o\Delta\mathcal{S}_o). \tag{17}$$

The Receding-Horizon principle is applied and the first element of $\Delta\mathcal{S}$ is taken to compute the control value as

$$s(k) = s(k-1) + \Delta s(k). \tag{18}$$

It is worth stressing that the quadratic optimization problem is solved offline in closed form and results to be very simple. Subsequently, the described control law is used. This is very efficient from a computational standpoint and does not overload the calculations. As a consequence, just some linear algebra involving matrices operations is in order to compute the control signal.

The tuning block $\mathcal{T}$, shown in Figure 1, performs the adaptation mechanism implements the classical RLS identification technique [Ljung 1998], taking s and y signals as inputs and online providing an estimate of $\theta \in \Re^{n_a+n_b}$, containing the parameters of the $ARX(n_a, n_b)$ model. In the following, $n_a = 1$ and $n_b = 1$.

3.4.9. Multiple actuators model predictive control. Extending the results of the single actuator case means essentially designing a suited $\mathcal{A}_{\hat{\theta}}^{-1}$ in order to compute the actual values of the manipulated knobs. In this case, the number of cores is coupled with the frequency and thus, equation (9) becomes

$$\mathcal{A}_\vartheta : \hat{s}(k) = [k_c\ c(k)^{\alpha_c} + o_c]\left[k_f\ \frac{f(k)^{\alpha_f}}{f_0} + o_f\right] \tag{19}$$

and rules can be defined to invert it, providing the number of cores and frequency from the speedup signal. The same rule used for the adaptive controller are here used.

3.5. Machine learning

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.

3.5.1. Genetic algorithms. Genetic Algorithms (GA) 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 tuple of knob values, in our case 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. More in general, one could select part of the knob values from the first candidate and part from the second. A mutation operator could be an increase or decrease in one of the knob values. 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 en-

coded solution, for example $1/d(hr^{\circ}, hr(k))$ where $d(\cdot, \cdot)$ represents the distance of the two points, calculated with a generic norm.

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 is very low (unless we have a variety of different knobs) 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, especially when a lot of different actuators are available.

3.5.2. Artificial Neural Networks. Artificial Neural Networks (ANNs) are the AI attempt to simulate brain activity. ANNs are networks composed by many instances of the same mathematical model representing a neuron. Each neuron is a "multiple inputs, single output" element and it is characterized by a vector of weights and an activation function. ANNs are usually organized as a hierarchy of layers of neurons and have algorithms to learn from classified examples how to bind inputs of the first layer to outputs of the last layer. In our approach the learning procedure is implemented by very common back-propagation algorithm.

ANNs need examples to be able to learn. But this is open contrast with our online/unseen-before system. To cope with that we implement a very simple layer above the ANN, drawing from classical planning and rule-based classification. We use a single measure to monitor the system, taking values in the interval $[0, +\infty)$. This measure is assumed to represent the "state" of the application and, through hr_{min}, hr_{max}, hr°, we can partition the whole state space in 4 macro states. The state is coded in an input vector of length 4, the action is the output of the feed-forward ANN.

Our actuators in this case have total ordering in the values they can assume, so, for each of them, three possible actions can be taken: increase, decrease or do nothing. We therefore need two output neurons for each actuator as given by $min\ x|2^x \geq 3$.

Given this problem formulation, the ANN needs to decide whether an action was good if taken in a certain state, i.e. whether or not to learn the state-action couple. To this extent, a set of rules is defined, to decide if the ANN should learn and how many iteration of the learning algorithm should be run. The main drawback in this hybrid approach is that, at the beginning, random choices are necessary to decide which is the right "direction".

The network has four layers. The first is the input one, that encodes the current state with four neurons, specifying the current heart rate position against thresholds. The two hidden layers have twenty neurons each, the number of neurons of the output layer depends on the number of actuators used. In general, the number of neurons to be used in the hidden layers is a critical parameter and would deserve further investigations, that are deferred to future works due to space limitations. The software library the implementation relies on [Nissen 2005] has few options to prevent over-fitting as much as possible, however, different configurations were tested resulting in similar results.

3.5.3. Single actuator neural network. In this case, we have two output neurons. If the first is greater than the second, an additional core is added to the application. If the second is greater than the first, a single core is deallocated from the application. When the output neurons are equal, the neural network is communicating to keep the current action and leave the number of cores unchanged.

3.5.4. Multiple actuators neural network. In this second case, four output neurons are used, to encode the couple of changes in cores and frequency, as done for the single actuator case.

3.5.5. Reinforcement learning. As for reinforcement learning solutions, we implement a SARSA (State-Action-Reward-State-Action) algorithm for the problem at hand [Sutton and Barto 1998]. 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, choosing an action, being this action a set of values (or a single value) for the knobs.

ALGORITHM 2: SARSA algorithm

Initialize $Q(s, a)$ for all possible s, a pairs
for each step k **do**
  $hr \leftarrow$ measured application heart rate
  Define the actual state s_k based on hr
  Use a policy π to select an action a_k, in this case the one with highest $Q(s_k, a_k)$
  Perform a_k
  Wait for some time and observe s_{k+1}
  Compute r_{k+1}
  Update $Q(s_k, a_k)$ according to Equation (20)
end for

Using the standard formulation of the algorithm, presented in Algorithm 2, the Q-value for a state-action is updated by an error, adjusted by the learning rate α, in the experiments this parameter is 0.5. The second algorithm parameter is the reward discount factor γ, 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.

We define a maximum value for the reward r_{k+1}. If the measured heart rate is in the correct range we reward the pair s_k, a_k with the maximum value. Otherwise, we attribute a reward based on the difference between the distance from hr° at time k and at time $k + 1$, therefore accounting for the evolution. The new value for the Q-value $Q(s_k, a_k)$ is the following

$$Q(s_k, a_k) \leftarrow Q(s_k, a_k) + \alpha[r_{k+1} + \gamma Q(s_{k+1}, a_{k+1}) - Q(s_k, a_k)] \tag{20}$$

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.

3.5.6. Single actuator reinforcement learning. In this case only the number of cores is considered as a possible actuator. The system has $c_{max} - c_{min}$ possible actions and three states. The exploration of the solution space should be very fast and the algorithm should reach a stable point easily, whenever this is possible.

3.5.7. Multiple actuator reinforcement learning. The algorithm has no conceptual difference from the previous case, the number of possible actions being all the possible set of configurations. Intuitively, when the number of actuators increases, the number of actions is affected exponentially. In this case there are $(c_{max} - c_{min})(f_{max} - f_{min})/f_{step}$ possible actions to be taken.

4. EXPERIMENTAL RESULTS

The case studies presented in this paper are application taken from the PARSEC benchmark suite [Bienia et al. 2008] and instrumented with the Application Heartbeats framework [Hoffmann et al. 2010]. 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.

4.1. Extended case studies

In this section we show extended case studies, with benchmarks taken from the PARSEC benchmark suite [Bienia 2011].

4.1.1. Swaptions. The `swaptions` application is a financial software that prices a portfolio of swaptions. It employs the Monte Carlo simulation method to compute the prices. The program uses an array to store the prices. As for parallelism, the application partitions the array into a number of blocks equal to the number of threads and assigns one block to every thread. Each thread then iterate over its block.

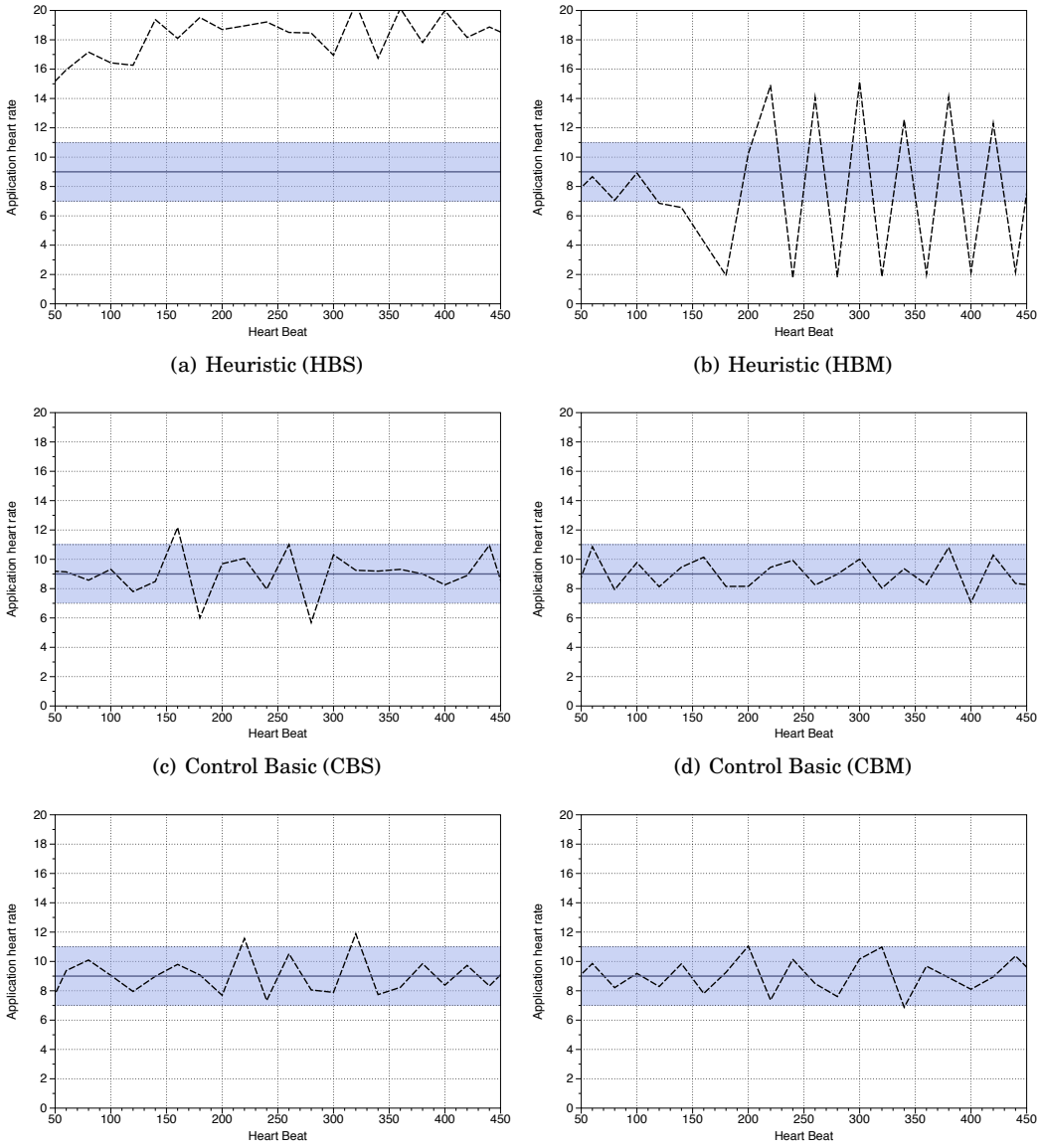
For our test, we decided to price 500 swaptions with 1000000 simulations each. This corresponds to the application emitting 500 heart beats over time. As reported in Table III we set a desired heart rate of 9 beats per second, specifying hr_{min} equal to 7 and hr_{max} equal to 11. Figures 2 and 3 report the test results. In each of the plots (and also in the following ones for other applications), the x-axis represents the application progress, expressed in Heart Beats (which could correspond to a measure of time). Each application is emitting a series of heart beats at non-regular intervals, and the controller strategies are acting when the application completes a prescribed number of heart beats, trying to adjust the progress rate of the software, depicted in the y-axis through the heart rate signal.

Figures 2(a) and 2(b) shows the results of the Heuristic approach. The Heuristic Based Single actuator (HBS) technique shown in Figure 2(a) is initialized outside the desired area and is not able to drive the performance signal while the Heuristic Based Multiple actuators (HBM) approach performs better, according to Figure 2(b), it starts when the heart rate is in the desired performance range and is able to attain an average heart rate, however bouncing between the upper region and the lower one.

Figures 2(c) shows the application of the Control Basic Single actuator (CBS) technique, while Figure 2(d) depicts the results of the Control Basic Multiple actuators (CBM). As can be seen both these techniques are able to attain the set point with few oscillations, while the sole workload adaptation fails in obtaining stable performances as can be noticed in Figure 2(e) for the Control Adaptive Single actuator (CAS) and Figure 2(f) for the Control Adaptive Multiple actuators (CAM).

Figures 3(a) and 3(b) report the execution of the Model Predictive Controller both for the Single (CPS) and for the Multiple (CPM) actuation mechanism. As can be seen, the use of a more complex control technique allows to attain the set point with much more precision than with simpler control structures.

In figure 3(c) the Machine learning Neural network with a Single actuator (MNS) results are shown, while Figure 3(d) reports the test with the same mechanism and multiple actuators (MNM). In the first case, the Neural Network is still learning the



(a) Heuristic (HBS)

(b) Heuristic (HBM)

(c) Control Basic (CBS)

(d) Control Basic (CBM)

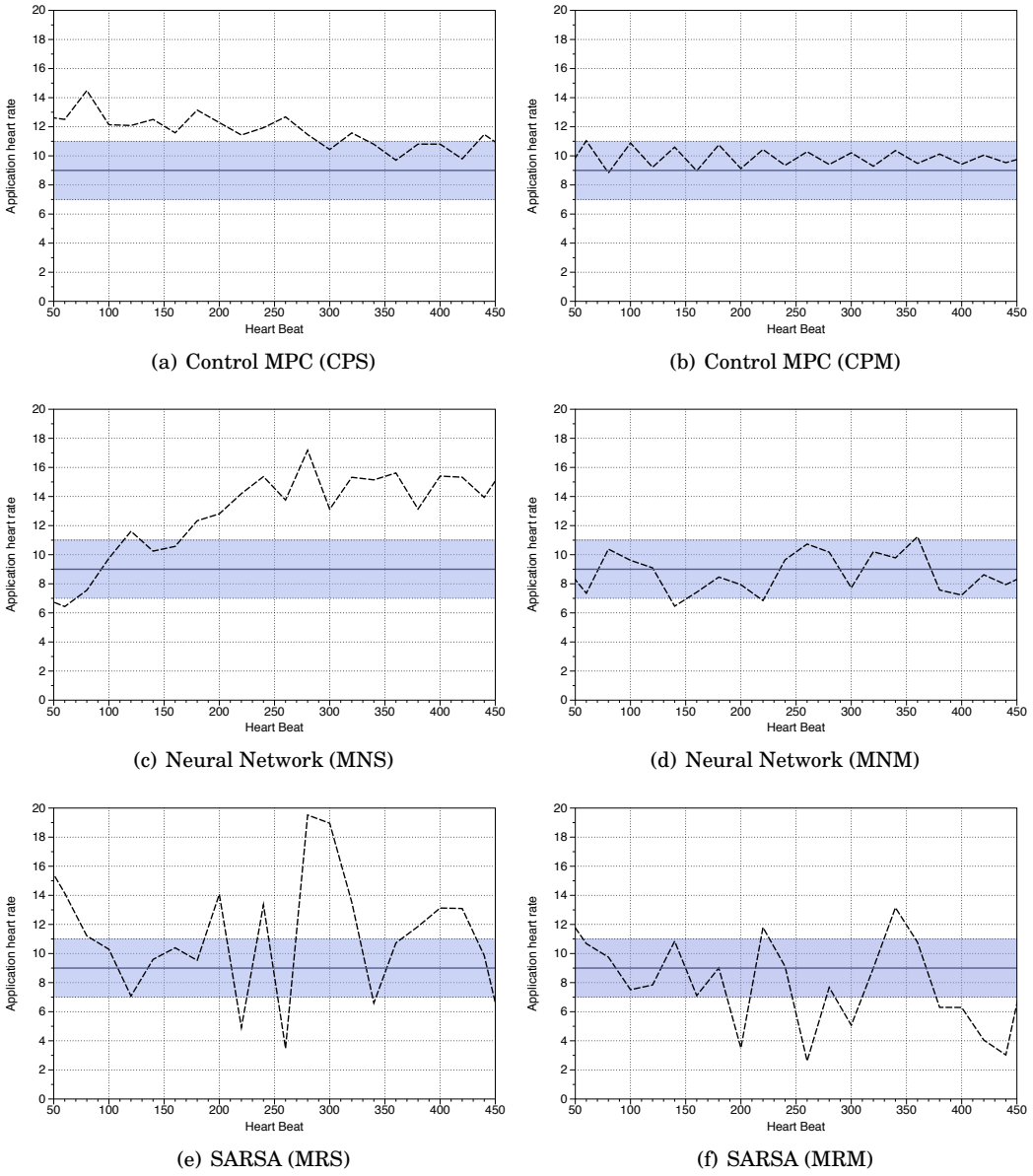
(e) Control Adaptive (CAS)

(f) Control Adaptive (CAM)

Fig. 2. Swaptions (part I)

best policy to be used while in the second case, with the frequency contribution, it is able to attain the set point more precisely.

Figures 3(e) and 3(f) show the SARSA Reinforcement Learning algorithm in the Single (MRS) and Multiple actuators (MRM) case. The reinforcement learning algorithm is still learning the best policy to attain the set point, therefore the results are not completely satisfactory. However, longer tests usually demonstrate the ability of the algorithm to learn, the disadvantage being slow convergence.



(a) Control MPC (CPS)

(b) Control MPC (CPM)

(c) Neural Network (MNS)

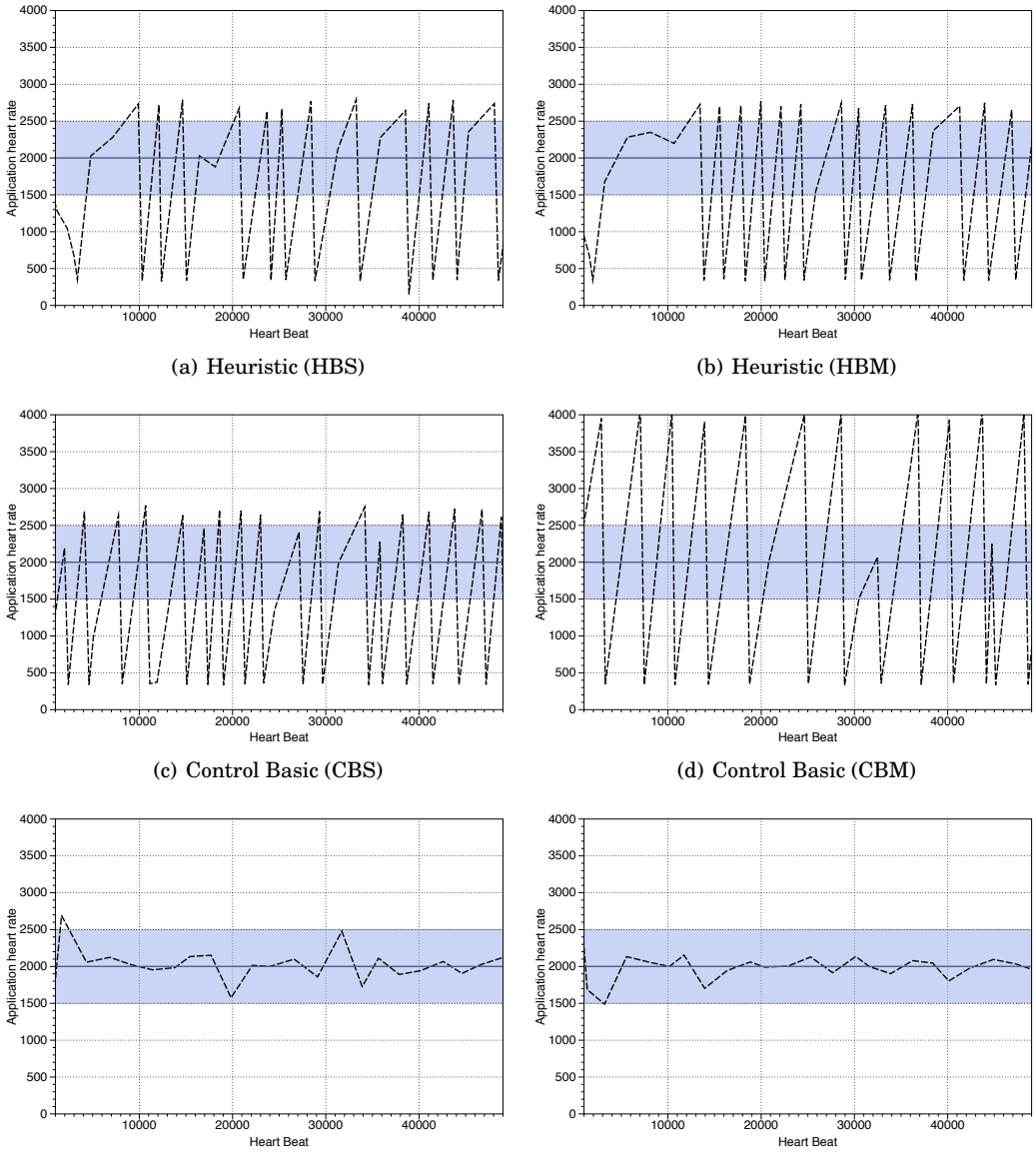
(d) Neural Network (MNM)

(e) SARSA (MRS)

(f) SARSA (MRM)

Fig. 3. Swaptions (part II)

4.1.2. Vips. The `vips` benchmark is a software application based on the VASARI image processing system. It includes fundamental image operations such as an affine transformation and a convolution and constructs multithreaded image processing pipelines transparently on the fly. The pipeline used for this test has 18 different stages. In our tests the `vips` application is supposed to emit 70000 heart beats and we desire to attain a set point of 2000 heart beats per second, being the acceptable range between 1500 and 2500 beats. Moreover, in the test, we use the `native` input configurations, acting on a 18000×18000 pixels image.



(a) Heuristic (HBS)

(b) Heuristic (HBM)

(c) Control Basic (CBS)

(d) Control Basic (CBM)

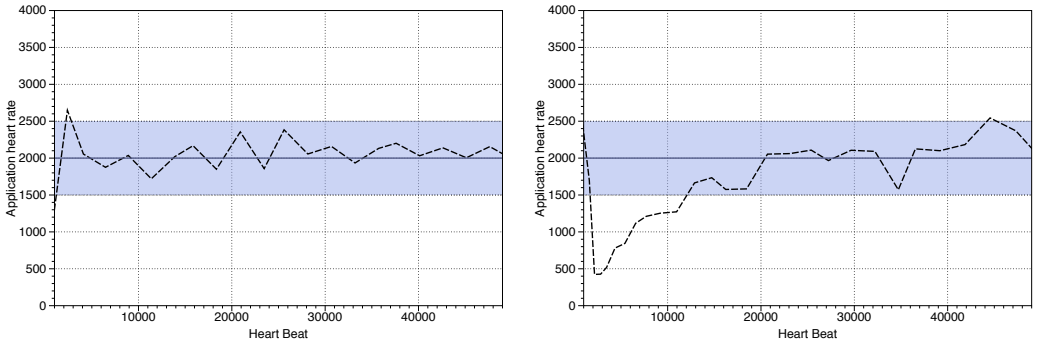
(e) Control Adaptive (CAS)

(f) Control Adaptive (CAM)

Fig. 4. Vips (part I)

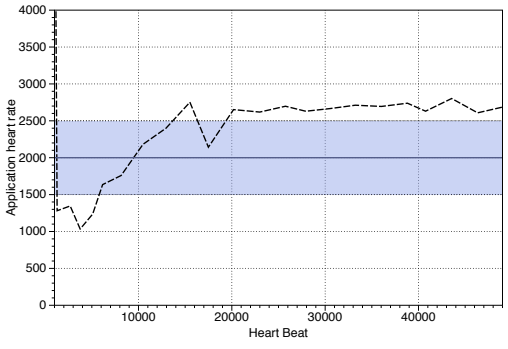
Notice that `vips` is way harder to control an application with respect to `swaptions`, because the parallelism model is much more complex. In fact, the introduction of a pipeline and of worker threads makes the control much more difficult. Not all the threads, in fact, are experiencing the same amount of work and the asymmetry of the workload affects the application performance.

Figures 4 and 5 show the results of this test and are organized as the `swaptions` test case ones. It is noticeable that the heuristic solutions, as well as simple control ones, fails in stabilizing the application performances. On the contrary, the Adaptive, Predic-

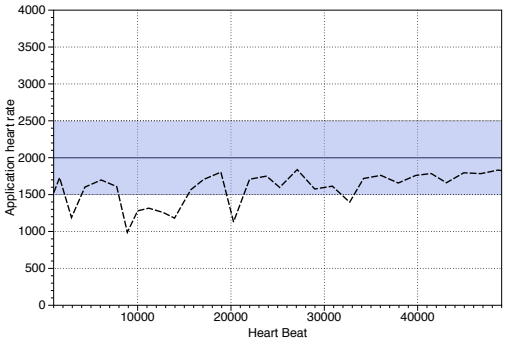


(a) Control MPC (CPS)

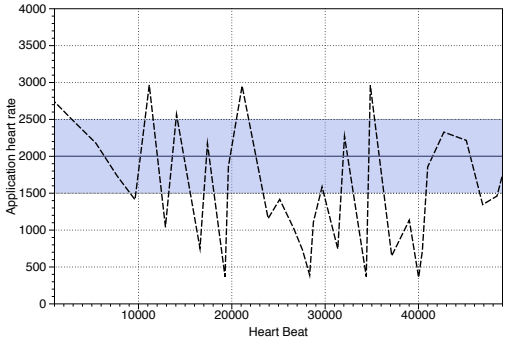
(b) Control MPC (CPM)



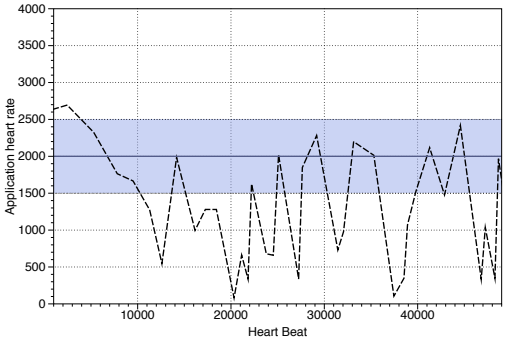
(c) Neural Network (MNS)



(d) Neural Network (MNM)



(e) SARSA (MRS)

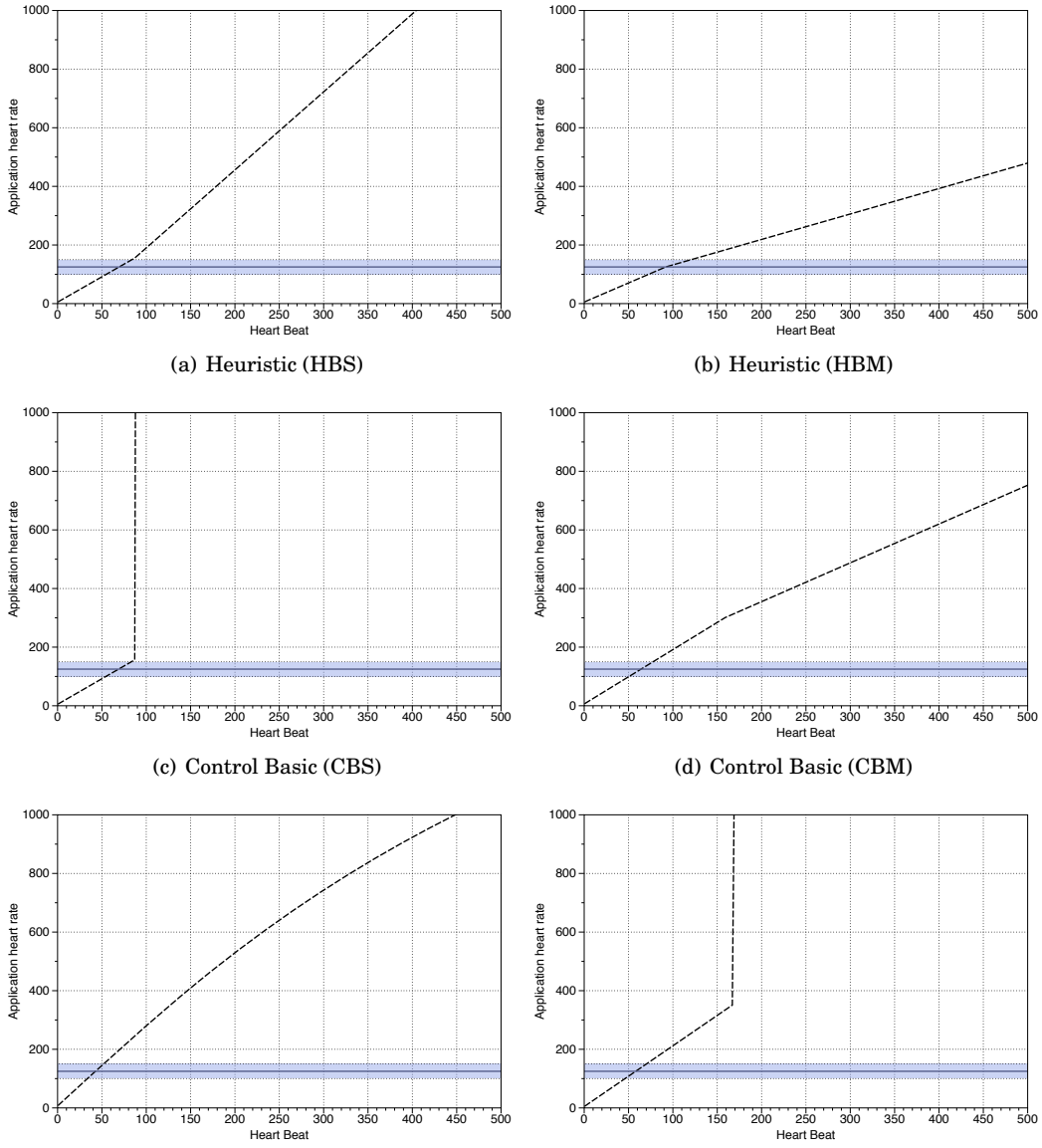


(f) SARSA (MRM)

Fig. 5. Vips (part II)

tive controllers and the Neural Networks are able to attain the desired set point, with some oscillations. The SARSA algorithms, even if they perform better than heuristic and basic control solutions, are not able to stabilize the heart rate signal, probably due to changes in the application behavior and therefore to the inability to learn the best action to be taken for all the possible configurations.

4.1.3. Freqmine. The `freqmine` application implements an array-based version of the frequent pattern-growth method for frequent itemset mining. During the first phase, the application build data structures, it subsequently performs operations on the data



(a) Heuristic (HBS)

(b) Heuristic (HBM)

(c) Control Basic (CBS)

(d) Control Basic (CBM)

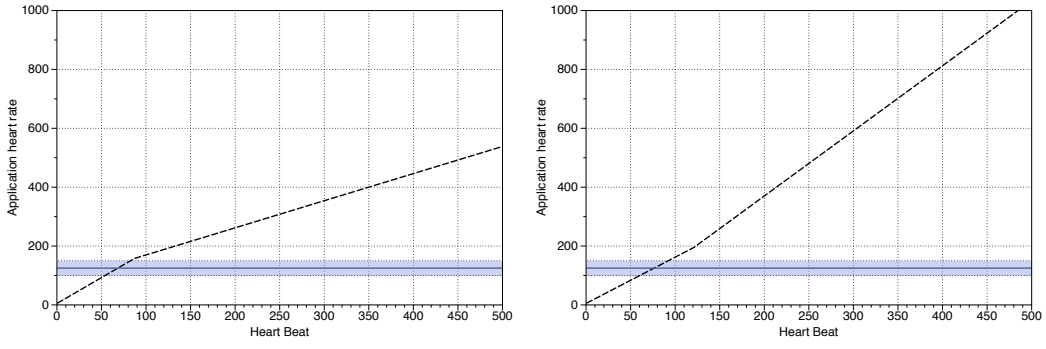
(e) Control Adaptive (CAS)

(f) Control Adaptive (CAM)

Fig. 6. Freqmine (part I)

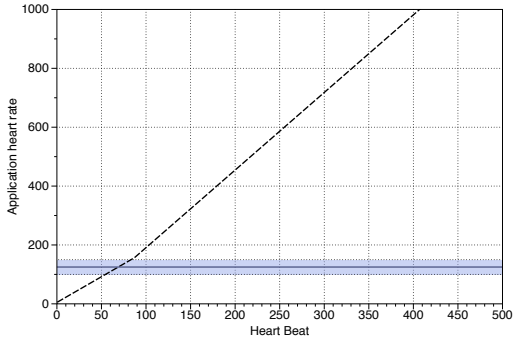
structure and mine the data. In our test case, the application is supposed to use a database with a collection of 250000 web html documents. The application is instrumented to emit a heart beat after processing 500 documents, therefore it emits 500 heart beats.

During the first phase, while data structures are still built, no matter how many computational resources are given to the application, it will not be able to match the set point. During the second phase, instead, the application will outperform the requirements even when the least amount of resources. We chose to set the desired value

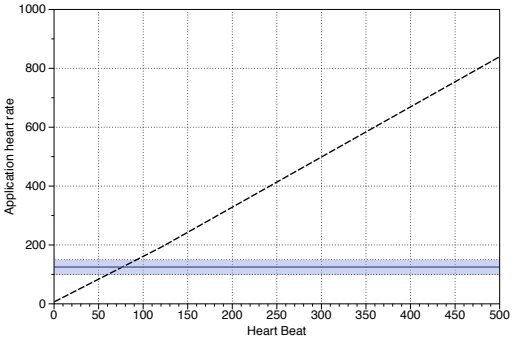


(a) Control MPC (CPS)

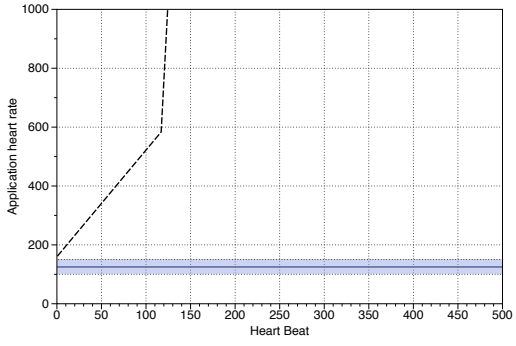
(b) Control MPC (CPM)



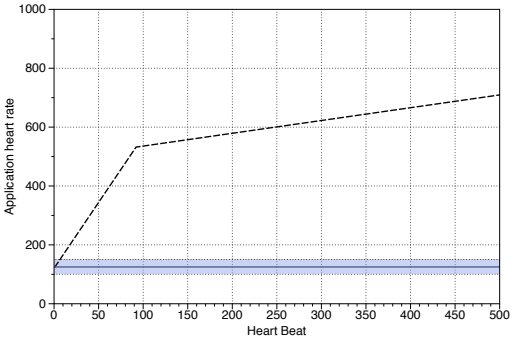
(c) Neural Network (MNS)



(d) Neural Network (MNM)



(e) SARSA (MRS)



(f) SARSA (MRM)

Fig. 7. Freqmine (part II)

to the average during a free execution with half of the available resources and, experiencing different phases, this application could not match the requirements in any of its portions. Figures 6 and 7 depict the results of this test and are organized as the `swaptions` and `vips` ones. Every control strategy behaves similarly and fails in assigning resources to the application. To obtain better results with this application one should specify two different set points, corresponding to two feasible values, one per each phase.

4.2. Overhead evaluation

The computational overhead of the different techniques was evaluated in the preliminary version of this work [Maggio et al. 2011], however, new techniques were added in this version of the paper and also the code for previously developed techniques was greatly improved.

Table II. Overhead results, seconds to execute the decision function 1000000 times.

Method	Overhead
HBS	0.00
HBM	0.00
CBS	0.02
CBM	0.01
CAS	0.20
CAM	0.21
CPS	1.88
CPM	1.71
MNS	0.01
MNM	0.01
MRS	0.01
MRM	0.03

Therefore Table II reports the seconds necessary to execute 1000000 times the decision function within the code, retrieved with `gprof`, the heuristic value being zero since the computation time is lower than one hundredth of a second. Notice, particularly, that the ANN code was improved dramatically by the use of convenient libraries. Also, as it could have been imagined, the model predictive code requires more time to be executed, however the code could still produce the required results within each step.

4.3. Evaluation metrics

To evaluate the goodness of the proposed solutions, we introduce different metrics. As previously stated, we use $hr(k)$ to refer to the heart rate signal, hr° to refer to the desired value for this signal, being it the average of the two threshold values, i.e.,

$$hr^\circ = \frac{hr_{max} + hr_{min}}{2}. \tag{21}$$

Suppose the application emits n heart beats in the controlled interval and denote with hr_{mean} the mean value of the heart rate experienced,

$$hr_{mean} = \frac{1}{n}\sum_{i=1}^{n} hr(i). \tag{22}$$

The first metric we define is W_{hr}, the difference between the mean of the heart rate signal and the desired value divided by the desired heart rate. As a second metric we also use the variance of the controlled heart rate V_{hr}.

$$W_{hr} = \frac{hr^\circ - hr_{mean}}{hr^\circ} \qquad V_{hr} = \frac{1}{n}\sum_{i=1}^{n} \left[hr(i) - hr_{mean}\right]^2 \tag{23}$$

Intuitively, one wants the absolute value of W_{hr} as close as possible to zero as well as the reduction of the variance V_{hr}.

Another metric we define is the Integral of the Squared Error (ISE), calcluated as

$$ISE = \frac{1}{n}\sum_{i=1}^{n}[hr^{\circ} - hr(i)]^2 \tag{24}$$

This metric defines in some sense a measure of the error of the corresponding heart rate signal. However, for different benchmarks, a high or low number of this value may mean different things. For example, if the set point hr° is less the unity, having an ISE of two or three means being very far from controlling the application behavior. If the same ISE is experienced when the set point is a thousand, the same value means controlling the application perfectly. Therefore we propose also a weighted value (ISE_w), defined as

$$ISE_w = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{hr^{\circ} - hr(i)}{hr^{\circ}}\right]^2. \tag{25}$$

This two measures combined gives a metric of how much the heart rate signal is close to its set point. One could also introduce a timed variant of the ISE, namely the ISTE, weighting the error over time, therefore lowering the pressure of the initial transient phase. This metric, although it could be useful, would result in better values for control theoretical solution, where the transient phase is the price to pay for modeling and controlling the system. However, this metric would not result in different information, therefore we omit these values in the following.

Another important metric is the percentage of recorded points that are in the undesired area, denoted by $\%_{wdp}$ and defined as

$$\%_{wdp} = \frac{1}{n}\left[n - \sum_{i=1}^{n} I_{hr_{min}<hr(i)<hr_{max}}\right] \tag{26}$$

where $I_{hr_{min}<hr(i)<hr_{max}}$ is the Borel set for the considered area, i.e., a function that equals to 1 if $hr(i)$ is in the chosen interval and to 0 otherwise.

The last metric, e_{wdp}, shows the sum of the absolute values of the distances from the closest treshold for the data points that are outside the valid range

$$e_{wdp} = \frac{1}{n}\sum_{i=1}^{n}\left[I_{hr(i)<hr_{min}}|hr(i) - hr_{min}| + I_{hr(i)>hr_{max}}|hr(i) - hr_{max}|\right] \tag{27}$$

showing therefore an idea of the experienced distances between the thresholds.

4.4. PARSECs aggregate

Similar tests with respect to the extended ones just proposed are conducted with all the PARSEC benchmarks. The applications set point and threshold values can be found in Table III and were obtained by analyzing the maximum performance value obtained on the target machine and the minimum one. An average between the two was chosen so the decision making system should choose to use an average value of resources with respect to the machine capabilities.

Instead of reporting plots for all the conducted tests, the performance metrics proposed in Section 4.3 are calculated after 30 executions of each test and averaged. The complete result set is reported in Appendix A, while some synthetic comments follow. Basically, applications (except one) can be divided in two groups.

— **Group 1** (bodytrack, dedup, facesim, fluidanimate, raytrace, streamcluster and swaptions) is composed by applications for which all the considered methods produce more or less comparable results. In fact, their response to resource allocation is

Table III. Experimental setup

benchmark	hr°	hr_{min}	hr_{max}
blackscholes	35	30	40
bodytrack	3	2.5	3.5
canneal	1500	1000	2000
dedup	35	30	40
facesim	0.3	0.2	0.4
ferret	30	25	35
fluidanimate	3	2.5	3.5
freqmine	125	100	150
raytrace	1.75	1.5	2
streamcluster	0.4	0.3	0.5
swaptions	9	7	11
vips	2000	1500	2500
x264	7.5	5	10

quite uniform within their run. In this case simpler the decision making method are preferable because of their computational lightness.

— **Group 2** (blackscholes, canneal, ferret, vips, x264) on the contrary requires adaptation owing to application response variability. In some of this cases, any adaptive technique is more or less equivalent, while others (canneal, ferret, vips) exhibit a particularly vast or complex search space, resulting in generally better performance of control-based methods.

We take the results of `bodytrack`, in *Group 1*, as a representative example. Table IV shows that the almost all the techniques are able to guarantee that the average heart rate of the application is close to the desired value, in most cases with a small variance. At the same time, the percentage of data points that are outside the prescribed region reveals that techniques that do not resort to exploration converge faster to the desired value. Similar results can be observed for other Group 1 applications in Appendix A.1.

Table IV. Bodytrack results

	W_{hr}	V_{hr}	ISE	ISE_w	$\%_{wdp}$	e_{wdp}
HBS	0.43	0.34	2.24	0.25	56	0.78
HBM	0.16	0.72	0.86	0.10	32	0.28
CBS	−0.03	0.01	0.01	0.00	00	0.00
CBM	0.06	0.09	0.11	0.01	00	0.00
CAS	0.05	0.46	0.41	0.05	16	0.15
CAM	0.00	0.07	0.06	0.01	00	0.00
CPS	0.08	1.80	1.58	0.18	71	0.63
CPM	0.25	1.62	1.89	0.21	58	0.68
MNS	0.31	0.24	1.27	0.14	68	0.51
MNM	0.35	0.45	1.49	0.17	62	0.61
MRS	−0.14	0.46	0.55	0.06	71	0.23
MRM	−0.10	0.18	0.25	0.03	39	0.07

Correspondingly, we show the results for `blackscholes` as an example of a *Group 2* application. Table V depicts the situation. Also in this case, the average heart rate is close to its desired value, however the signal presents much more variance, and this reflects also in performance indices like ISE and ISE_w. Each of the techniques has a percentage of data points outside the desired interval that is way higher than in easier application cases, however some techniques achieve smaller errors than others. Similar data can be noticed for other applications in Appendix A.2.

Table VI shows the data for the only exception, `freqmine`. None of the considered techniques is satisfactory in this case. A reason for that can however be identified in the different phases which this application experiences. There is a phase that rapidly

Table V. Blackscholes results

	W_{hr}	V_{hr}	ISE	ISE_w	$\%_{wdp}$	e_{wdp}
HBS	−0.54	135.01	852.74	0.70	81	20.50
HBM	−0.07	254.60	240.58	0.20	52	8.06
CBS	−0.09	487.98	454.00	0.37	88	14.91
CBM	−0.24	809.23	787.03	0.64	84	19.22
CAS	0.01	62.79	57.56	0.05	14	1.55
CAM	0.18	159.43	242.46	0.20	34	6.51
CPS	−0.24	287.39	318.69	0.26	70	10.14
CPM	−0.25	138.64	207.43	0.17	70	7.59
MNS	0.41	43.38	326.83	0.27	82	10.84
MNM	0.31	54.75	187.65	0.15	74	7.25
MRS	−0.44	485.98	645.47	0.53	83	16.68
MRM	−0.11	236.91	229.03	0.19	80	8.68

produces heart beats and one which is extremely slow. In the slow phase the application is significantly slower than the average heart rate with maximum resources, while in the fast phase it is above the requirements.

Table VI. Freqmine results

	W_{hr}	V_{hr}	ISE	ISE_w	$\%_{wdp}$	e_{wdp}
HBS	−53.22	1564679751.32	1173213595.19	75085.67	37	3316.41
HBM	−0.02	6617.95	9.12	0.00	00	0.00
CBS	−3.81	373958.30	256142.17	16.39	42	227.07
CBM	−1.34	40851.17	9726.01	0.62	32	47.92
CAS	−2.70	107492.15	41321.61	2.64	32	104.55
CAM	−1.44	46672.34	12953.25	0.83	29	52.80
CPS	−3.33	326798.73	222229.47	14.22	39	196.55
CPM	−0.50	16583.49	1395.50	0.09	31	12.94
MNS	−52.86	1536282954.83	1152012087.25	73728.77	43	3292.11
MNM	−0.48	15656.18	1316.77	0.08	32	12.07
MRS	−4.12	131459.58	101024.14	6.47	33	163.50
MRM	−5.12	295051.91	199100.27	12.74	33	204.88

5. CONCLUSION AND FUTURE WORKS

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 Heartbeats 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.

Some conclusions can be drawn from the experiments. First, instrumentation is crucial and whenever an application cannot be instrumented in a sensible manner, there is simple no way out. Second, the relative performance of the various techniques is influenced by application variability in quite predictable a way. In other words, forecasts on the application variability can easily be turn into choices of the most appropriate techniques. Third, the same remark above applies to different extents depending on how many and which actuators have the most significant influence on the application.

When these are more than one, techniques that in some sense resort to a space search, suffer significantly if not properly initialized. Fourth, and last, the control-based methods deserve being considered as viable alternatives to more often used ones. In fact, they invariantly perform comparably with all the other techniques, sometimes slightly worse but usually better. To summarize, our results indicate that the best decision method can vary depending on the specific application to be optimized; however, adaptive and model predictive control systems tend to produce good performance and may work best when the applications to be controlled are a priori unknown.

REFERENCES

ANSEL, J., ANS CY CHAN, Y. L. W., OLSZEWSKI, M., EDELMAN, A., AND AMARASINGHE, S. 2011. Language and compiler support for auto-tuning variable-accuracy algorithms. In *The International Symposium on Code Generation and Optimization*. Chamonix, France.

ÅSTRÖM, K. AND HÄGGLUND, T. 2005. *Advanced PID Control*. ISA - The Instrumentation, Systems, and Automation Society, Research Triangle Park, NC.

BIENIA, C. 2011. Benchmarking modern multiprocessors. Ph.D. thesis, Princeton University.

BIENIA, C., KUMAR, S., SINGH, J. P., AND LI, K. 2008. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*.

BITIRGEN, R., IPEK, E., AND MARTINEZ, J. 2008. 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. IEEE Computer Society, Washington, DC, USA, 318–329.

BITTANTI, S. AND PICCI, G. 1996. *Identification, adaptation, learning: the science of learning models from data*. Vol. 153. Springer Verlag.

BODÍK, P., GRIFFITH, R., SUTTON, C., FOX, A., JORDAN, M., AND PATTERSON, D. 2009. Statistical machine learning makes automatic control practical for internet datacenters. In *Proceedings of the 2009 conference on Hot topics in cloud computing*. HotCloud'09. USENIX Association, Berkeley, CA, USA, 12–12.

BREITGAND, D., HENIS, E., AND SHEHORY, O. 2005. Automated and adaptive threshold setting: Enabling technology for autonomy and self-management. In *Proceedings of the Second International Conference on Automatic Computing*. IEEE Computer Society, Washington, DC, USA, 204–215.

CAMACHO, E. AND BORDONS, C. 2004. *Model predictive control*. Springer Verlag.

CHASE, J., ANDERSON, D., THAKAR, P., VAHDAT, A., AND DOYLE, R. 2001. Managing energy and server resources in hosting centers. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*. SOSP '01. ACM, New York, NY, USA, 103–116.

HELLERSTEIN, J., DIAO, Y., PAREKH, S., AND TILBURY, D. 2004. *Feedback Control of Computing Systems*. Wiley.

HELLERSTEIN, J., MORRISON, V., AND EILEBRECHT, E. 2009. 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.

HELLERSTEIN, J. L. 2010. Why feedback implementations fail: the importance of systematic testing. In *Proceedings of the Fifth International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*. FeBiD '10. ACM, New York, NY, USA, 25–26.

HOFFMANN, H., EASTEP, J., SANTAMBROGIO, M., MILLER, J., AND AGARWAL, A. 2010. 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. ACM, New York, NY, USA, 79–88.

HOFFMANN, H., MAGGIO, M., SANTAMBROGIO, M. D., LEVA, A., AND AGARWAL, A. 2011. SEEC: a general and extensible framework for self-aware computing. Tech. Rep. MIT-CSAIL-TR-2011-046, Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Laboratory, Cambridge. November.

IBM. 2006. An architectural blueprint for autonomic computing. Tech. rep. June.

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

KEPHART, J. AND CHESS, D. 2003. The vision of autonomic computing. *Computer 36*, 41–50.

LIU, X., ZHU, X., SINGHAL, S., AND ARLITT, M. 2005. Adaptive entitlement control of resource containers on shared servers. In *Proceeding of the 9th IFIP/IEEE International Symposium on Integrated Network Management*. 163–176.

LJUNG, L. 1998. *System Identification: Theory for the User*. Prentice Hall PTR.

LU, C., LU, Y., ABDELZAHER, T., STANKOVIC, J., AND SON, S. 2006. Feedback control architecture and design methodology for service delay guarantees in web servers. *IEEE Transactions on Parallel and Distributed Systems 17,* 7.

MAGGIO, M., HOFFMANN, H., SANTAMBROGIO, M. D., AGARWAL, A., AND LEVA, A. 2010. Controlling software applications via resource allocation within the heartbeats framework. In *Proceeding of the 49th international conference on decision and control*. IEEE Control, Atlanta, USA.

MAGGIO, M., HOFFMANN, H., SANTAMBROGIO, M. D., AGARWAL, A., AND LEVA, A. 2011. Decision making in autonomic computing systems: comparison of approaches and techniques. In *Proceedings of the 8th ACM international conference on Autonomic computing*. ICAC '11. ACM, New York, NY, USA, 201–204.

MARTINEZ, J. AND IPEK, E. 2009. Dynamic multicore resource management: A machine learning approach. *IEEE Micro 29*, 8–17.

NISSEN, S. 2005. Neural Networks made simple. *Software 2.0 2*, 14–19.

O'DWYER, A. 2006. *Handbook of PI And PID Controller Tuning Rules* Second Ed. Imperial College Press.

PAN, W., MU, D., WU, H., AND YAO, L. 2008. Feedback Control-Based QoS Guarantees in Web Application Servers. In *Proceedings of the 10th International Conference on High Performance Computing and Communications*. 328–334.

RAMIREZ, A., KNOESTER, D., CHENG, B., AND MCKINLEY, P. 2009. Applying genetic algorithms to decision making in autonomic computing systems. In *Proceedings of the 6th international conference on Autonomic computing*. ICAC '09. ACM, New York, NY, USA, 97–106.

SUTTON, R. AND BARTO, A. 1998. Reinforcement learning: an introduction. 322.

TESAURO, G. 2007. Reinforcement learning in autonomic computing: A manifesto and case studies. *IEEE Internet Computing 11*, 22–30.

TESAURO, G., JONG, N., DAS, R., AND BENNANI, M. 2006. A hybrid reinforcement learning approach to autonomic resource allocation. In *Proceedings of the 2006 IEEE International Conference on Autonomic Computing*. IEEE Computer Society, Washington, DC, USA, 65–73.

TSAI, C., HSU, Y., LIN, C., AND LIN, W. 2009. Review: Intrusion detection by machine learning: A review. *Expert Syst. Appl. 36*, 11994–12000.

ULAM, P., GOEL, A., JONES, J., AND MURDOCH, W. 2005. Using model-based reflection to guide reinforcement learning. In *Proceedings of the 2005 IJCAI Workshop on Reasoning, Representation and Learning in Computer Games*. 1–6.

WILDSTROM, J., WITCHEL, E., AND MOONEY, R. 2005. Towards self-configuring hardware for distributed computer systems. In *Proceedings of the Second International Conference on Automatic Computing*. IEEE Computer Society, Washington, DC, USA, 241–249.

ZHANG, R., LU, C., ABDELZAHER, T., AND STANKOVIC, J. 2002. Controlware: A middleware architecture for feedback control of software performance. In *Proceedings of the 22nd International conference on Distributed Computing Systems*. IEEE computer society.

Received May 2011; revised Whenever; accepted Whenever

A. FURTHER EXPERIMENTAL RESULTS

A.1. Group 1 applications

Table VII. Dedup results

	W_{hr}	V_{hr}	ISE	ISE_w	$\%_{wdp}$	e_{wdp}
HBS	0.05	31.01	29.47	0.02	34	1.11
HBM	0.09	23.59	32.83	0.03	37	1.32
CBS	0.02	47.19	40.48	0.03	30	1.55
CBM	0.06	41.69	39.84	0.03	38	1.52
CAS	0.20	23.35	75.76	0.06	62	3.37
CAM	0.17	22.80	90.52	0.07	50	3.82
CPS	−0.07	68.63	63.56	0.05	43	2.17
CPM	−0.01	42.13	46.05	0.04	48	1.85
MNS	0.15	29.75	54.95	0.04	51	2.35
MNM	0.05	52.88	49.10	0.04	48	2.04
MRS	0.02	52.62	44.46	0.04	44	1.77
MRM	0.15	38.40	58.23	0.05	46	2.30

Table VIII. Facesim results

	W_{hr}	V_{hr}	ISE	ISE_w	$\%_{wdp}$	e_{wdp}
HBS	0.17	0.00	0.00	0.03	00	0.00
HBM	0.17	0.00	0.00	0.03	00	0.00
CBS	0.04	0.00	0.00	0.00	00	0.00
CBM	0.07	0.00	0.00	0.01	00	0.00
CAS	−0.00	0.00	0.00	0.01	02	0.00
CAM	0.00	0.00	0.00	0.00	00	0.00
CPS	0.03	0.02	0.02	0.22	66	0.05
CPM	0.14	0.01	0.01	0.13	33	0.02
MNS	−0.06	0.00	0.01	0.11	33	0.02
MNM	−0.04	0.01	0.01	0.17	29	0.03
MRS	0.03	0.01	0.01	0.06	10	0.01
MRM	0.14	0.01	0.02	0.18	49	0.03

Table IX. Fluidanimate results

	W_{hr}	V_{hr}	ISE	ISE_w	$\%_{wdp}$	e_{wdp}
HBS	−0.01	0.01	0.05	0.01	00	0.00
HBM	0.09	0.30	0.37	0.04	10	0.13
CBS	−0.01	0.20	0.18	0.02	23	0.01
CBM	0.02	0.06	0.06	0.01	07	0.03
CAS	0.03	0.01	0.01	0.00	00	0.00
CAM	0.01	0.02	0.02	0.00	03	0.01
CPS	0.07	0.18	0.21	0.02	22	0.10
CPM	0.12	0.37	0.47	0.05	27	0.21
MNS	0.33	0.12	1.57	0.17	78	0.52
MNM	0.28	0.22	1.01	0.11	68	0.45
MRS	0.21	0.52	0.88	0.10	37	0.36
MRM	0.24	0.33	0.82	0.09	49	0.28

Table X. Raytrace results

	W_{hr}	V_{hr}	ISE	ISE_w	$\%_{wdp}$	e_{wdp}
HBS	0.05	0.00	0.01	0.00	00	0.00
HBM	0.22	0.76	0.88	0.29	81	0.56
CBS	0.02	0.00	0.00	0.00	00	0.00
CBM	0.04	0.01	0.02	0.01	11	0.01
CAS	−0.01	0.02	0.02	0.01	03	0.01
CAM	0.00	0.00	0.00	0.00	01	0.00
CPS	0.04	0.00	0.01	0.00	00	0.00
CPM	0.12	0.03	0.07	0.02	37	0.06
MNS	0.25	0.04	0.57	0.18	86	0.38
MNM	0.30	0.15	0.59	0.19	84	0.42
MRS	−0.22	0.70	0.82	0.27	87	0.55
MRM	0.06	0.27	0.29	0.09	56	0.21

Table XI. Streamcluster results

	W_{hr}	V_{hr}	ISE	ISE_w	$\%_{wdp}$	e_{wdp}
HBS	0.08	0.00	0.00	0.02	03	0.00
HBM	−0.06	0.00	0.00	0.01	02	0.00
CBS	−0.00	0.00	0.00	0.01	02	0.00
CBM	−0.01	0.00	0.00	0.00	02	0.00
CAS	0.01	0.00	0.00	0.02	05	0.00
CAM	−0.01	0.00	0.00	0.01	02	0.00
CPS	0.05	0.01	0.01	0.06	30	0.02
CPM	0.11	0.01	0.01	0.07	27	0.02
MNS	0.12	0.00	0.01	0.05	18	0.02
MNM	0.08	0.01	0.01	0.06	35	0.01
MRS	−0.14	0.01	0.01	0.06	44	0.01
MRM	0.10	0.01	0.01	0.06	27	0.02

Table XII. Swaptions results

	W_{hr}	V_{hr}	ISE	ISE_w	$\%_{wdp}$	e_{wdp}
HBS	−0.05	1.32	1.53	0.02	11	0.03
HBM	0.03	11.28	11.62	0.14	37	1.18
CBS	0.01	2.77	2.58	0.03	21	0.22
CBM	0.02	2.29	2.17	0.03	07	0.14
CAS	0.02	2.32	2.21	0.03	08	0.17
CAM	0.04	2.90	3.06	0.04	11	0.24
CPS	0.17	2.89	14.95	0.18	63	1.59
CPM	0.27	3.20	15.04	0.19	68	1.60
MNS	0.22	4.33	13.10	0.16	82	1.42
MNM	0.39	5.44	17.95	0.22	75	2.01
MRS	−0.32	14.60	21.43	0.26	70	2.08
MRM	−0.01	9.40	9.03	0.11	50	0.94

A.2. Group 2 applications

Table XIII. Canneal results

	W_{hr}	V_{hr}	ISE	ISE_w	$\%_{wdp}$	e_{wdp}
HBS	0.34	83906.90	355386.15	0.16	51	155.00
HBM	0.58	34864.17	838724.26	0.37	81	403.98
CBS	0.34	224284.40	464799.07	0.21	44	204.61
CBM	0.32	194318.58	433495.87	0.19	43	192.13
CAS	0.32	163993.19	351597.04	0.16	33	145.79
CAM	0.38	118444.15	461262.29	0.21	61	206.05
CPS	0.38	131509.65	426132.72	0.19	59	184.02
CPM	0.54	7100.69	611676.49	0.27	91	287.40
MNS	0.64	29260.14	960332.67	0.43	87	458.74
MNM	0.50	9268.43	664617.59	0.30	74	326.64
MRS	0.56	75129.34	742520.11	0.33	83	345.66
MRM	0.63	61458.83	881605.82	0.39	89	421.25

Table XIV. Ferret results

	W_{hr}	V_{hr}	ISE	ISE_w	$\%_{wdp}$	e_{wdp}
HBS	0.01	29.54	28.86	0.03	34	1.05
HBM	0.17	75.07	102.63	0.11	50	3.89
CBS	0.01	158.81	153.90	0.17	67	5.83
CBM	0.00	130.94	126.69	0.14	62	4.75
CAS	0.03	34.02	33.86	0.04	33	1.15
CAM	0.06	77.49	82.07	0.09	45	2.85
CPS	0.35	26.90	231.66	0.26	71	8.40
CPM	0.33	27.59	211.76	0.24	75	7.82
MNS	0.38	21.91	236.84	0.26	83	8.75
MNM	0.25	43.54	151.88	0.17	72	6.03
MRS	0.05	130.89	133.63	0.15	64	5.09
MRM	0.32	72.92	164.51	0.18	77	6.39

Table XV. Vips results

	W_{hr}	V_{hr}	ISE	ISE_w	$\%_{wdp}$	e_{wdp}
HBS	0.02	43180.78	52376.11	0.01	03	11.76
HBM	0.17	873194.48	969351.91	0.24	58	397.94
CBS	−0.09	2617141.79	2484345.27	0.62	83	1000.66
CBM	−0.07	3181852.63	2994589.71	0.75	92	1185.84
CAS	−0.00	185821.37	174458.05	0.04	12	57.89
CAM	0.02	240614.75	234980.99	0.06	15	80.35
CPS	−0.01	256104.65	262467.32	0.07	22	98.58
CPM	0.03	212973.40	237492.31	0.06	20	81.94
MNS	0.26	104406.37	780818.20	0.20	61	322.32
MNM	0.39	140940.37	953911.96	0.24	68	409.97
MRS	−0.11	1216227.14	1177390.02	0.29	66	490.34
MRM	0.25	560755.47	781699.95	0.20	55	330.31

Table XVI. X264 results

	W_{hr}	V_{hr}	ISE	ISE_w	$\%_{wdp}$	e_{wdp}
HBS	−0.00	30.27	30.33	0.54	19	0.76
HBM	−0.05	29.90	29.61	0.53	20	1.04
CBS	−0.04	7.70	7.81	0.14	20	0.52
CBM	−0.02	11.52	11.32	0.20	19	0.65
CAS	0.05	7.38	7.44	0.13	17	0.46
CAM	0.00	10.42	10.20	0.18	21	0.62
CPS	−0.01	24.15	24.77	0.44	29	0.93
CPM	−0.01	440.74	441.31	7.85	28	1.39
MNS	−0.05	28.28	28.94	0.51	29	1.36
MNM	−0.01	14.07	14.21	0.25	26	0.70
MRS	−0.10	19.41	19.66	0.35	24	1.15
MRM	−0.10	21.55	21.59	0.38	25	1.16