Automated Control of Multiple Software Goals using Multiple Actuators

Maggio, Martina; Papadopoulos, Alessandro Vittorio; Filieri, Antonio; Hoffmann, Henry

Published in:
ESEC/FSE 2017 Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering

DOI:
10.1145/3106237.3106247

2017

Document Version:
Peer reviewed version (aka post-print)

Link to publication

Citation for published version (APA):
Maggio, M., Papadopoulos, A. V., Filieri, A., & Hoffmann, H. (2017). Automated Control of Multiple Software Goals using Multiple Actuators. In ESEC/FSE 2017 Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (pp. 373-384). Association for Computing Machinery, Inc. https://doi.org/10.1145/3106237.3106247

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Automated Control of Multiple Software Goals using Multiple Actuators

Martina Maggio
Lund University, Sweden
martina@control.lth.se

Alessandro Vittorio Papadopoulos
Mälardalen University, Sweden
alessandro.papadopoulos@mdh.se

Antonio Filieri
Imperial College London, UK
a.filieri@imperial.ac.uk

Henry Hoffmann
University of Chicago, US
hankhoffmann@cs.uchicago.edu

ABSTRACT

Modern software should satisfy multiple goals simultaneously: it should provide predictable performance, be robust to failures, handle peak loads and deal seamlessly with unexpected conditions and changes in the execution environment. For this to happen, software designs should account for the possibility of runtime changes and provide formal guarantees of the software's behavior. Control theory is one of the possible design drivers for runtime adaptation, but adopting control theoretic principles often requires additional, specialized knowledge. To overcome this limitation, automated methodologies have been proposed to extract the necessary information from experimental data and design a control system for runtime adaptation. These proposals, however, only process one goal at a time, creating a chain of controllers. In this paper, we propose and evaluate the first automated strategy that takes into account multiple goals without separating them into multiple control strategies. Avoiding the separation allows us to tackle a larger class of problems and provide stronger guarantees. We test our methodology's generality with three case studies that demonstrate its broad applicability in meeting performance, reliability, quality, security, and energy goals despite environmental or requirements changes.

CCS CONCEPTS

•**Computer systems organization** → **Embedded systems;** *Redundancy;* Robotics; •**Networks** → Network reliability;

KEYWORDS

Adaptive Software, Control Theory, Dynamic Systems, Non-Functional Requirements.

ACM Reference format:
Martina Maggio, Alessandro Vittorio Papadopoulos, Antonio Filieri, and Henry Hoffmann. 2017. Automated Control of Multiple Software Goals using Multiple Actuators. In *Proceedings of 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Paderborn, Germany, September 4–8, 2017 (ESEC/FSE'17),* 12 pages.
DOI: 10.1145/3106237.3106247

ESEC/FSE'17, Paderborn, Germany
© 2017 ACM. 978-1-4503-5105-8/17/09…$15.00
DOI: 10.1145/3106237.3106247

1 INTRODUCTION

The growing complexity and dynamic unpredictability of computer systems has motivated the design and implementation of a new class of *self-adaptive* software systems [49]. Such software automatically reacts to changes in both the operating environment and application behavior to ensure that certain high-level goals are met. The development of self-adaptive software creates, however, a huge challenge: designing software systems that are robust in the face of dynamic behaviors we are not aware of at design-time. To face the challenge, software systems are often highly configurable [58], and can often modify their behavior during runtime.

Control theory provides a vast array of tools for designing robust adaptive systems that operate with formal guarantees [3, 5]. This combination of robustness and formal grounding has led to increased interest in developing self-adaptive software based on control theoretic techniques [46]. While several *ad hoc* approaches to control design have arisen, recent work proposes a general, automated methodology for creating formally robust software control [15]. However, this first approach can handle only a single, measurable goal (*e.g.*, performance or reliability, but not both).

Modern software, however, must meet multiple, often conflicting, goals. For example, software for cyber-physical systems must meet performance, energy, and security requirements simultaneously. Very little research has addressed automating the design of self-adaptive software that meets multiple goals. A first step proposes a hierarchy of single-goal controllers [16, 24]. In this approach, goals are ordered. Higher priority goals are met first using one set of actuators (or tunable software parameters), and then those actuators are removed from consideration for the controllers that manage lower priority goals. Priorities can be set based on user preference. The actuators are partitioned into disjoint sets, with fewer actuators available to meet lower priority goals.

Despite being one of the first solutions to offer multidimensional control, this hierarchical approach has two limitations. First, due to the partitioning of actuators, the controller may not reach low-priority goals even when they are feasible. A combined approach that considers the effects of concurrent actuation, however, should reach any set of feasible goals. Second, the hierarchical approach only provides formal guarantees for the highest priority goal, others are not guaranteed. Thus, there is a need for a formally verifiable methodology to automate the design of self-adaptive software that meets multiple goals using multiple actuators. Finally, the approach in [16, 24] requires actuators to assume only a finite number of values. While continuous actuators can be automatically discretized [16], the complexity of control may grow exponentially, limiting its applicability when timely decisions are required or when the available computational capacity is limited (e.g., IoT).

We therefore propose a novel *automated* strategy—based on multivariable control—that simultaneously uses all available actuators to meet all goals. Unlike prior work, our approach allows users to:

- Reach sets of goals that are unreachable with the prior hierarchical approach, and in fact, to reach any set of feasible goals.
- Identify the largest subset of goals that are reachable with the available actuators. Additionally, users can express desires like controlling *the largest subset of goals that contains specific non-deniable objectives*; *e.g.*, finding the largest set of reachable goals where request latency is below 0.1 seconds.
- Shape the software's dynamic behavior by specifying a cost function to be optimized, while prioritizing the goals based on their relevance. For example, users may prefer upgrading an existing virtual machine instead of starting up a new one.
- Provide faster convergence to goals after system perturbation. The cascaded approach determines an actuator setting for each goal, waits for that goal to be reached, and then addresses the next goal. In contrast, the multivariable controller sets all actuators simultaneously, offering much faster convergence to the goals and response to environmental fluctuation.
- Model and exploit the mutual dependencies that exist among the actuators and goals. Instead of considering partitions of actuators one-by-one, we model all the combined effects and exploit them to obtain more precise and efficient control.
- Limit the identification time needed for the system design. Instead of testing all possible values for different actuators during the learning phase, we only need to test random switches from the minimum to maximum value for each actuator. The length of the initial model identification phase is then greatly reduced, becoming proportional to the number of actuators and not to the number of permutations of all the possible actuator values.
- Tune the tradeoff between control overhead and optimality of the actuator settings with respect to the cost function. This tunability allows users to construct solutions with acceptable overhead, whereas no prior control synthesis approach supports user adjustable overhead.

We demonstrate the advantages of the proposed approach through three case studies developing self-adaptive software including: a video encoder, a secure radar system, and a dynamic service binder. We use case studies from prior work to highlight the additional capabilities proposed in this paper. The remainder of this paper is organized as follows. Section 2 compares the approach presented in this paper to other automated methodologies for automated multi-concern control. Section 3 presents the technical details of the proposed approach and the Section 4 discusses the formal guarantees that can be given using the proposed strategy. Section 5 shows experimental evidence of how the proposed strategy works and Section 6 concludes the paper.

To foster future research and enable comparison with our approach, we published the source code for our experiments and to generate the control strategies used in the remaining of this paper[1].

2 RELATED WORK

Modern software systems must be robust to frequent, unpredictable changes to their execution environment, users, and requirements. *Self-adaptive* software adjusts its behavior at runtime, withstanding external changes as they are detected, or even proactively avoiding critical situations [5, 9, 33, 51]. One great challenge of self-adaptation is ensuring its effectiveness and dependability [13, 19, 57]. Control theory has defined a variety of techniques for controlling the behavior of physical plants, and its formal framework serves as a basis for a variety of software adaptation mechanisms [8, 12, 17, 18, 23].

Control of software systems. Recent surveys capture the current state-of-the-art applying control-theory to software applications [17, 46, 59]—from controlling web server delays [38], to data service management [10], resource allocation [2, 26, 27, 35, 47], operating systems tuning [30, 40, 45], and energy management [25, 41]. Some of these systems use automata-based formalisms to abstract software's behavior and temporal logic to specify some of its requirements [9, 51], while we focus here on discrete-time control, where equation-based models are used to satisfy quantitative software properties.

Most discrete-time control approaches satisfy quantitative, non-functional requirements: controlling tunable actuators identified either by the designer or automatically [22] and whose value affects the software behavior. The majority of software controllers belong to the family of Proportional-Integral-Derivative (PID) controllers [36]. PIDs are computationally inexpensive and support formal analysis of their dynamics. They are, however, limited to linear (or linearized) system models and mostly control a single goal (*e.g.*, the response time) using a single actuator (*e.g.*, the number of VMs). This approach is known as single-input, single-output (SISO). In contrast, multiple-input, multiple-output (MIMO) controllers are more complex, managing conflicting goals and contending actuators.

Model predictive control (MPC). MPC is an effective, flexible solution for MIMO problems [39]. MPC design incorporates the different actuators' higher-order dynamics; *i.e.*, it captures how each actuator affects each goal and interferes with the other actuators.

MPC controllers decide the control signals for the next time step by optimizing a utility function that accounts for both the current operating point and all possible trajectories up to a given horizon [20, 34]. Reasoning based on predictions of future behavior has proven effective in other self-adaptation approaches [12, 44, 61], though with reasoning techniques mostly ad-hoc and tailored to user-defined models. In contrast, MPC provides a more general analysis framework and the ability to refine and compensate for model inaccuracies by exploiting a feedback loop. MPC-based adaptation mechanisms have been used to define controllers for a class of goal models [1] and resource provisioning in cloud environments under uncertainty [54]. These MPC solutions, however, requires the developer to explicitly provide a system model and are tied to specific problems; in turn, they require the developer to master modeling techniques and do not generalize beyond the models manually defined by the developer.

Automated controller synthesis. Automated controller synthesis eases the integration of control-theoretical adaptation into software systems. Abstracting specific views of the software system

[1] http://www.martinamaggio.com/papers/fse17/

into equation-based models and defining adequate control strategies are open problems for current development processes [17, 18].

The first automated modeling and controller synthesis approach has been proposed in [15]. It builds a locally linearized model of a SISO system collecting input-output measurements during system execution. Then, a tunable controller is continuously adjusted around the current operation point to provide computationally efficient and robust adaptation decisions, under mild assumptions on the smoothness of the – possibly non-linear – system behavior. This approach cannot deal with general MIMO systems, however.

Recent work automatically synthesizes MIMO controllers for discrete systems by chaining multiple SISO controllers (one for each goal) together in a hierarchy [16]. The hierarchy reflects goal prioritization, and each controller produces a continuous reference signal that is converted into a mixture of the discrete input configurations using Pulse Width Modulation [36]. The two main limitations of this solution are the use of disjoint sets of actuators along the hierarchy of SISO controllers and the need for actuators to assume values from a finite domain. Actuators that are used to reach a higher priority goal cannot be changed to meet the lower priority ones, limiting the controller's ability to achieve all goals optimally at the same time (see also the experimental comparison in Section 5.3). The need for finite domains for the actuators requires the discretization of continuous control inputs; while this can be done automatically, as in [16], the complexity of the control law may grow exponentially, limiting the practical applicability of the approach when timely decisions are required or the controller has to run on low-power devices, like embedded systems. The approach in [50] extends [16], formulating the conversion of the continuous references into discrete settings as a linear optimization problem. This approach avoids partitioning actuators into disjoint sets and allows actuation to minimize a cost function (*e.g.*, considering the priority of different actuators), but it does not provide an explicit means for handling conflicting goals when the satisfaction of one makes others infeasible.

This paper's contribution. This paper proposes automated model construction and controller synthesis for MIMO controllers. Unlike prior work [16, 50], it does not require the input space to be finite, requires less observation to build a comprehensive equation-based model of the system, and produces optimal control decisions considering not only the current situation but also future predicted system evolutions.

3 METHODOLOGY

A MIMO system has multiple actuators that influence multiple goals. Our methodology makes two assumptions: 1) the user knows all available actuators and their limits; *i.e.*, the maximum and minimum values they can assume; and 2) design-time tests can be performed that measure the effects actuator changes have on goals. The proposed methodology minimizes the number of tests to be performed – a big improvement with respect to prior work [15, 16, 50] – but some experimental data collection is required to build a model and synthesize the controller.

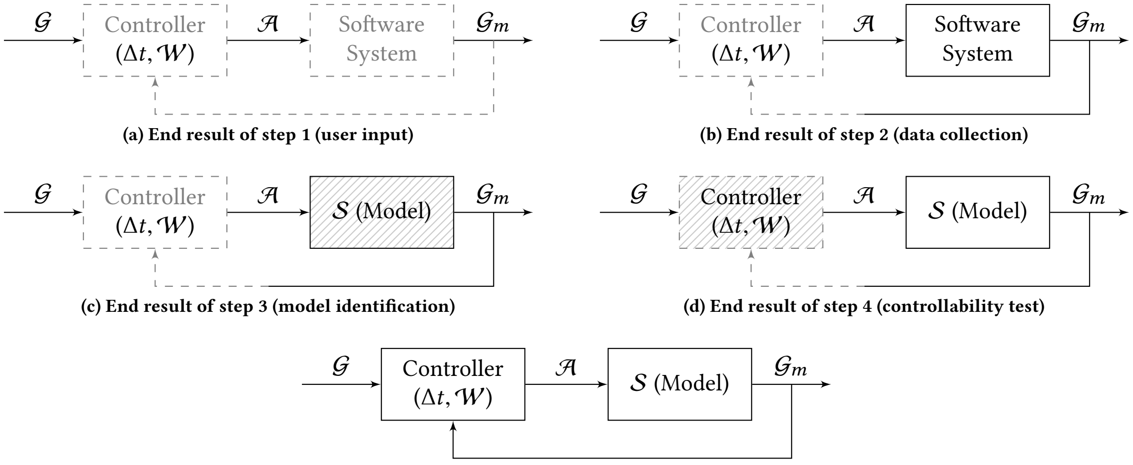
The methodology has five steps, illustrated in Figure 1. It requires several inputs from the user, but is otherwise completely automated, requiring no control expertise. The fifth step outputs code implementing the controller. The framework collects user inputs (Step 1) and then collects data by running experiments (Step 2). This data is used to estimate a model relating actuator changes to changes in system behavior (Step 3). A *controllability test* is then performed to ensure the provided actuators can reach the specified goals (Step 4). If the system is controllable, the methodology synthesizes a controller and generates the resulting code (Step 5). The user then selects the desired values for the goals and complements the generated code with calls to the sensors (to obtain the current values of the goals), and to the actuators (to effectively perform the action chosen by the controller). These user-defined functions for sensing and actuating are system-dependent and represent the interfaces to the rest of the software.

Step 1: user input. The user specifies a sampling time Δt, a set of v actuators $\mathcal{A} = \{a_1, a_2, \dots a_v\}$, and a set of p goals. For example, a_1 might be the clock speed of the execution environment, while a_2 might be the probability of using one service provider over another. The values assumed by the set $\mathcal{A}$ at time k are denoted with the vector $a(k)$. The user provides the values of $a_{i,\min}$ and $a_{i,\max}$, which are the minimum and maximum values for the i-th actuator.

The set of goals is $\mathcal{G} = \{g_1, g_2, \dots g_p\}$. For example, g_1 might be the 95-th percentile response latency, and g_2 might be power consumption. The goals' values at time k are denoted with the vector $g(k)$. $g(k)$ is a function of time as goals may change while the software is running. The user should also provide a way to measure the current value of the goals. We denote with the set $\mathcal{G}_m = \{g_{m_1}, g_{m_2}, \dots g_{m_p}\}$ the measured values corresponding to the goals — in the example g_{m_1} is the current 95-th percentile of the response time and g_{m_2} the current power consumed by the embedded device. Again, the vector $g_m(k)$ is the measurements at time k. We assume that $p \leq v$, *i.e.* there cannot be more goals than actuators, an inherent limitation of any control approach. For each goal g_j, the user specifies a weight w_j, resulting in a set $\mathcal{W} = \{w_1, w_2, \dots w_p\}$. Goal weighting specifies a proportional disparity between goals' importance when goals aren't simultaneously satisfiable. Equally weighted goals imply their errors are treated equally and the controller balances between them. Optionally, the user can specify additional weights for the actuators, $\mathcal{D} = \{d_1, d_2, \dots d_p\}$, where a lower weight indicates changing the corresponding actuator is preferred to a higher weight actuator. The controller tries to minimize the sum of the products of actuators and weights. Because of this, a lower weight would favor the use of the corresponding actuator, while a higher weight would make the controller try to avoid the use of the corresponding actuator unless it is really necessary. As seen in Figure 1a, the results of this phase are the input quantities that the controller synthesis needs.

Step 2: data collection. For $100 \cdot v$ uniformly spaced time intervals (each Δt apart) our methodology selects values for the vector $a(k)$ and records $g_m(k)$. For each $a_i \in \mathcal{A}$, we choose either the minimum value $a_{i,\min}$ or the maximum value $a_{i,\max}$. As seen in Figure 1b, this phase is the first step towards closing the feedback loop. Considering that k belongs to the interval $[1, 100 \cdot v]$, this data collection lasts for $100 \cdot v \cdot \Delta t$ time, which is $O(v)$, a strong improvement over previous methods that sweep the entire parameter space [15, 16]. Compared to prior work, the resulting models have less fidelity, but the synthesized MPC is robust to these modeling inaccuracies. As in prior work [15], we update the model at runtime to capture variations.

Step 3: model identification. We use subspace identification [52, 53] to build a linear model based on the data. We build the lowest



(a) End result of step 1 (user input)

(b) End result of step 2 (data collection)

(c) End result of step 3 (model identification)

(d) End result of step 4 (controllability test)

(e) End result of step 5 (MIMO controller synthesis)

Figure 1: Controller synthesis phases. Dashed elements are not yet introduced or exploited at the corresponding stage. For example, the measurement of $\mathcal{G}_m$ are used in Step 2 for the model building phase (Step 3). At the end of Step 3, the Software System is replaced with its corresponding model $\mathcal{S}$. In Step 4, the methodology verifies that the controller can be built and Step 5 produces the end result and allow the software engineer to close the loop.

possible order model that fits the data. Selecting a higher order increases the model's accuracy but also increases the chance of overfitting. On the contrary, a lower order model is less accurate, but also increases the probability that the resulting models capture fundamental behavior rather than noise. This noise derives, for example, from the presence of the operating system routines and from other applications running at the same time on the hardware.

At this point we have a model of order n. In such a model, the dynamic system has n states and a state vector $x = [x_1, x_2, \ldots x_n]$. Notice that x is not a measurable quantity, and not even some quantity that has a meaningful interpretation for the user, but an abstract variable that links the inputs to the outputs, and thus, describes the system's dynamics in a compact form. The values of x may not correspond to anything measurable in the system.

The subspace identification procedure returns a model $\mathcal{S}$ in the difference equation form

$$\mathcal{S}: \begin{cases} x(k+1) = A \cdot x(k) + B \cdot a(k) \\ g_m(k) = C \cdot x(k) \end{cases}, \tag{1}$$

using Δt as sampling interval (the distance in time between two subsequent measurements). The time is represented with the letter k, that denotes the sampling instants and assumes values in the set of integers where a number k is the instant $t = \Delta t \cdot k$ in time. The following steps use $\mathcal{S}$ as a model of the system we want to control. Figure 1c shows that from here on, in the controller synthesis procedure, the real software is substituted with its model. The identification procedure is completely automatic and no input is required from the user, if not for the collected data.

Step 4: controllability test. From control theory [21], a system is controllable if the $n \times (p \cdot n)$ matrix

$$C_o = \begin{bmatrix} B & A \cdot B & \cdots & A^{n-1} \cdot B \end{bmatrix} \tag{2}$$

has rank n, which means that C_o has n independent columns among the $n \cdot p$ total columns[2]. If this condition holds, a controller can drive all states to any feasible value, in the absence of actuator saturations. Recall that the output values are linked to the state values by the second equation in (Eq. 1). Thus, if the states can assume any desired value, then the output can assume any desired value in the feasible region. In practical terms, if the goal values in $\mathcal{G}$ can be reached, a controller can be constructed that will set the actuators in $\mathcal{A}$ to reach them. In the presence of saturations, the goals may not be reached, but the controller will drive the measurements as close as possible to the goal. For example, if the user specifies a maximum number of virtual machines that can be spawned to improve the response time of a cloud application, and more than the maximum number would be necessary to serve all the requests, the goal is not reached despite the system being *controllable*, but the controller will decrease the response time as much as possible with the given resources. If the system is controllable, the controller we synthesize in Step 5 will reach all the goals whenever possible [32]. If the system is not controllable, we are able to detect it and warn the user. To solve this problem, one may add other actuators or reduce the set of goals and re-run the previous steps.

Step 5: MIMO controller synthesis. Based on the model from Step 3 and on the controllability test from Step 4, we automatically synthesize a Model Predictive Controller (MPC) [4, 32, 39]. This controller is complemented with a Kalman Filter (KF) [37] to update the system model as the controller runs. MPC is a control technique that formulates an optimization problem to use the set $\mathcal{A}$ of actuators to achieve the set $\mathcal{G}$ of goals. At every control instant k, the problem becomes the minimization of a loss function ℓ_k, subject to given constraints. A common approach to guarantee the removal of the steady state error is to introduce integral action into the controller [32]. This can be done simply by rewriting the identified model (Eq. 1) in the *augmented velocity form*. Letting $\Delta x(k) := x(k) - x(k-1)$, $\xi(k) := \begin{bmatrix} \Delta x(k)^\top & g_m(k-1)^\top \end{bmatrix}^\top$, and

$$\overline{A} := \begin{bmatrix} A & 0_{n \times p} \\ C & I_{p \times p} \end{bmatrix}, \quad \overline{B} := \begin{bmatrix} B \\ 0_{p \times v} \end{bmatrix}, \quad \overline{C} := \begin{bmatrix} C & I_{p \times p} \end{bmatrix},$$

[2] Recall that n is the number of model states and p the number of goals.

the augmented velocity form is expressed as:

$$\mathcal{S}_a : \begin{cases} \xi(k+1) = \overline{A}\xi(k) + \overline{B}\Delta a(k) \\ g_m(k) = \overline{C}\xi(k) \end{cases} \tag{3}$$

The augmented velocity form is typically used for the formulation of the MPC, since it allows integral action in the loop; *i.e.*, in practical terms, it guarantees that the controlled system reaches all the goals when kept constant over time. The system output $g_m(k)$ is unchanged but now expressed with respect to the state variations $\Delta x(k)$ and not with respect to the state values $x(k)$. The new model (Eq. 3) predicts future state values with a time horizon of L steps; *i.e.*, L discrete time steps from now. The MPC then minimizes the following cost function

$$\ell_k = \sum_{i=1}^{L}\sum_{j=1}^{p} q_j \left(g_{m,j,k+i} - g_{j,k+i}\right)^2 + \sum_{j=1}^{m} r_j \left(\Delta a_{j,k+i-1}\right)^2, \tag{4}$$

where q_j and r_j are positive weights, that respectively represent the importance of the distance between the j-th goal and the current value, and the inertia to changing the j-th actuator. The values of the weights can be chosen as $\mathcal{W}$ given by the user in Step 1. When one or more goal is infeasible (for example because one conflicts with the other), the controller favors the goals with the higher weights. The values r_j indicate preferences on actuators, and is chosen either as one or the elements of $\mathcal{D}$.

The resulting MPC optimization problem can written as follows.

$$\begin{aligned} \text{minimize}_{\Delta a_{k+i-1}} \quad & \ell_k && (5) \\ \text{subject to} \quad & a_{\min} \le a_{k+i-1} \le a_{\max} \\ & \Delta a_{\min} \le \Delta a_{k+i-1} \le \Delta a_{\max} \\ & g_{m,\min} \le g_{m,k+i-1} \le g_{m,\max} \\ & \xi_{k+i} = \overline{A}\xi_{k+i-1} + \overline{B}\Delta a_{k+i-1} \\ & g_{m,k+i-1} = \overline{C}\xi_{k+i-1} \\ & i = 1, \ldots, L. \end{aligned}$$

This formulation is equivalent to a convex Quadratic Programming (QP) problem [32]. The solution of the QP problem has time complexity of $O(L^3 v^3)$ [55]. The solution is an optimal plan for the future $\Delta a^{\star}_{k+i-1}$, $i = 1, \ldots, L$, but typically a *receding horizon* approach is adopted, and only the first action of the plan, *i.e.* $\Delta a^{\star}_{k}$, is applied. The new control signal is then obtained as

$$a(k) = a(k-1) + \Delta a^{\star}_{k}. \tag{6}$$

The receding horizon principle is particularly important, since the model (Eq. 1) will never capture all environmental phenomena. Therefore, the plan needs to be recomputed every time new information is available. In case of real-time constraints on finding a solution, it is possible to store the past planned control trajectory that would have been disregarded, and use it if the solver does not converge in time.

The MPC strategy assumes that the process state is measurable, but in many cases this is not possible — recall that the system state has a non-trivial interpretation. Indeed, it is impossible to measure $x(k)$ directly and so it must be estimated based on $g_m(k)$. To accomplish this, we use a KF that computes an estimate $\hat{x}(k+1)$ of the state $x(k+1)$, as

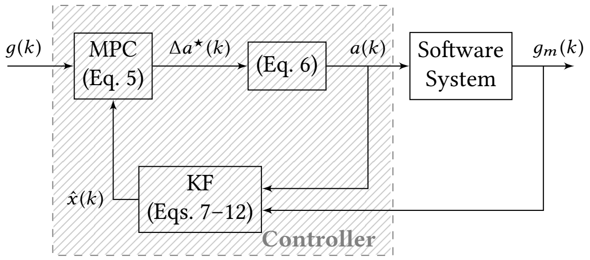


Figure 2: Control scheme.

$$M(k) = P(k)C^{\top}\left(CP(k)C^{\top} + R_n\right)^{-1} \tag{7}$$

$$e(k) = g_m(k) - C\hat{x}(k) \tag{8}$$

$$\overline{x}(k) = \hat{x}(k) + M(k)e(k) \tag{9}$$

$$\overline{P}(k) = (I - M(k)C)\,P(k) \tag{10}$$

$$\hat{x}(k+1) = A\overline{x}(k) + Ba(k) \tag{11}$$

$$P(k+1) = A\overline{P}(k)A^{\top} + BQ_nB^{\top} \tag{12}$$

where (Eqs.7–10) update the KF with the new information from the prediction error $e(k)$ in (Eq.8), and (Eqs.11–12) compute a prediction of the system's state and of the covariance matrix P. $M(k)$ is also called Kalman gain, and adapts over time depending on the magnitude of the prediction error e [37]. The estimate $\hat{x}(k)$ can be used, in place of $\xi(k)$, to solve the optimization problem in (Eq. 5).

Figure 2 shows the block diagram for the resulting control scheme. The controller is then executed every Δt time units. Summarizing, the control design is performed by using the identified matrices of the model (Eq. 1), and by choosing appropriate weights for the cost function, *i.e.*, q_j and r_j. The Kalman filter is designed on the basis of the identified model and keeps said model updated during runtime. Step 5 produces the python code for the MPC, which should be complemented with the code used to obtain the measured values of the goals and apply the actuators values.

Discussion. To apply this methodology, users must provide sensors that measure behavior for any goals the controller should meet. These sensors usually take the form of methods that return some system property; *e.g.*, performance or power. Users must also provide a list of actuators and their minimum and maximum values. These actuators are, again, usually methods that changes some key parameter; *e.g.*, the strength of a filter. For the methodology to work, the number of goals (and thus sensors) must, in general, be less than or equal to the number of actuators. Similarly, if the actuators cannot be used to meet the goals (*i.e.*, the controllability test fails), our methodology reports this to the users who must either add actuators or change goals. The methodology assumes that actuator settings are continuous between the minimum and maximum settings. If the actuators are discrete, then users can either choose the closest discrete setting or approximate the continuous value by time averaging different actuators settings. Our experiments include examples of both approaches (Sections 5.2 and 5.3). Importantly, we note that this methodology does not assume linear functions mapping actuator settings to goals. The Kalman filter is added specifically to account for complex actuation mechanisms: it continually computes an optimal estimate of the underlying system state, as measurable by the sensors. This process of continual state estimation makes the controller robust to inaccuracies in the system identification process. To demonstrate this robustness, all three case studies in the evaluation section include actuators with complex, non-linear interactions. Additional stress tests that evaluate more complex actuation functions (including cosine and higher-order polynomials, which we have never observed in real software) are available with our code release.

Implementation. We provide an implementation that automatically generates the controller code in Python and C++, which we use for our case studies. The user does not need to provide more than the weights, the bounds on the actuators and the actuate and sense functions that interact with the software. We implemented Step 3, 4 and 5 in Matlab. The subspace identification procedure relies on the Matlab function n4sid, using as parameters the given data and the keyword best for the model order. In this way, the model's order is the one that best approximate the data obtained during the experimental phase in the range between one and ten.

4 FORMAL ASSESSMENT

Applying control theory in software systems provides a set of formal guarantees about the software's response to dynamic changes [17]. The MPC presented in this work belongs to the class of *optimal controllers*, since control decisions are based on the solution of an optimization problem [32]. In particular, adopting the model predictive approach allows us to provide a number of formal guarantees.

Convergence to the objectives. MPCs generated by our methodology ensure that *all goals are reached, when they are reachable* [32]; *i.e.*, if there are actuator settings that achieve the goals within the given constraints (Eq. 5), then the MPC will find them. Convergence is proven by observing that whenever there exists a feasible actuator configuration, the MPC optimization problem is equivalent to the unconstrained optimization problem that minimizes ℓ_k [32]. The values for the actuators' variation are $\Delta a_k^\star = \operatorname{argmin}_a \ell_k$. Considering the gradient of (Eq. 4), the closed-form solution is $\nabla_a \ell_k = 2H\Delta a + 2F\xi$, where H and F are functions of q_j, r_j and the dynamic matrices of the system (Eq. 3). The gradient has a minimum for $\nabla_a \ell_k = 0$, which corresponds to $\Delta a^\star = -H^{-1}F\xi(k)$, where $\Delta a^\star$ is a vector containing the optimal plan for the future $\Delta a_{k+i-1}^\star$, $i = 1, \ldots, L$. In the MPC case, only the first element of the plan is applied.

The controller can thus be expressed as a matrix multiplied by the current state value $\xi(k)$ as $a(k) = \Gamma\xi(k)$. Thus, the closed-loop system dynamics (Eq. 3) can be rewritten as follows.

$$\xi(k+1) = (\overline{A} + \Gamma)\xi(k)$$
$$g_m(k) = \overline{C}\xi(k)$$

Assume, without loss of generality, that all goals are zero, $g(k) = 0$. A well known result says that the steady-state error converges to zero if and only if all eigenvalues of $\overline{A} + \Gamma$ have magnitudes less than unity [32]. If q_j in (Eq. 4) are all positive (required by our methodology), the eigenvalues of $\overline{A} + \Gamma$ always lie in the unit circle in the complex plane. This property guarantees stability and convergence to the objective when it is reachable with the actuators supplied by the user.

Minimum distance for infeasible cases. If the goals are not reachable, the MPC finds actuator settings that minimize the overall steady state error. The definition of "closeness" depends on the weights for each term of the cost function (Eq. 4) – a solution is closer to the desired one when it minimizes the cost function [32]. The minimum distance depends on q_j. Since different values of q_j yield different quantitative solutions, the choice of q_j is used to enforce the prioritization of the goals.

Minimum convergence time. The dynamic model in (Eq. 1) relates control parameters and outputs to time. The optimization problem finds the best trajectory converging to the goals, according to the selected cost function ℓ_k. By construction, the cost function penalizes all the time instants when g_m is not equal to the goal g, therefore the MPC leads to a minimum settling time solution.

Real-Time Computation. Since the proposed solution solves an optimization problem at each control instant, it is critical to discuss timing issues that could prolong the controller's execution. In some cases, the time required for computing the next control action might be longer than the time between two subsequent control actions. To address this issue, there is a vast literature in the control community on how to implement fast solvers (*e.g.*, [29, 48]). The area has been explored especially when these solvers are part of embedded systems [28, 31]. For an overview on the matter see [60].

Often, such advanced algorithms are not required when dealing with software components. For example, one can set $\Delta a_{k+1} = \Delta a_{k+2} = \ldots = \Delta a_{k+L-1} = 0$ and solve the optimization problem for Δa_k, which is the only one that will actually be applied at time k. This modification reduces the complexity to be just $O(v^3)$. Another approach exploits interior point algorithms, which iteratively update a feasible, but sub-optimal, solution to the constraints. If the iteration did not converge before a new control action is required, it can be forced to stop and return the current sub-optimal solution. Finally, another possibility to deal with real-time deadlines is exploiting the MPC's proactive nature. At each time step, the MPC computes a plan of future actions Δa_{k+i-1}, $i = 1, \ldots, L$. According to the receding horizon principle, only the first one is applied; *i.e.*, $\Delta a(k) = \Delta a_k^\star$. Assuming that at the next control instant the solver takes more time to converge and that a new control action is required before the optimal solution is found, one can store the previously computed plan and apply the second control action; *i.e.*, $\Delta a(k+1) = \Delta a_{k+1}^\star$. Doing so is obviously sub-optimal, but fulfills real-time deadlines and execution constraints.

5 EXPERIMENTAL EVALUATION

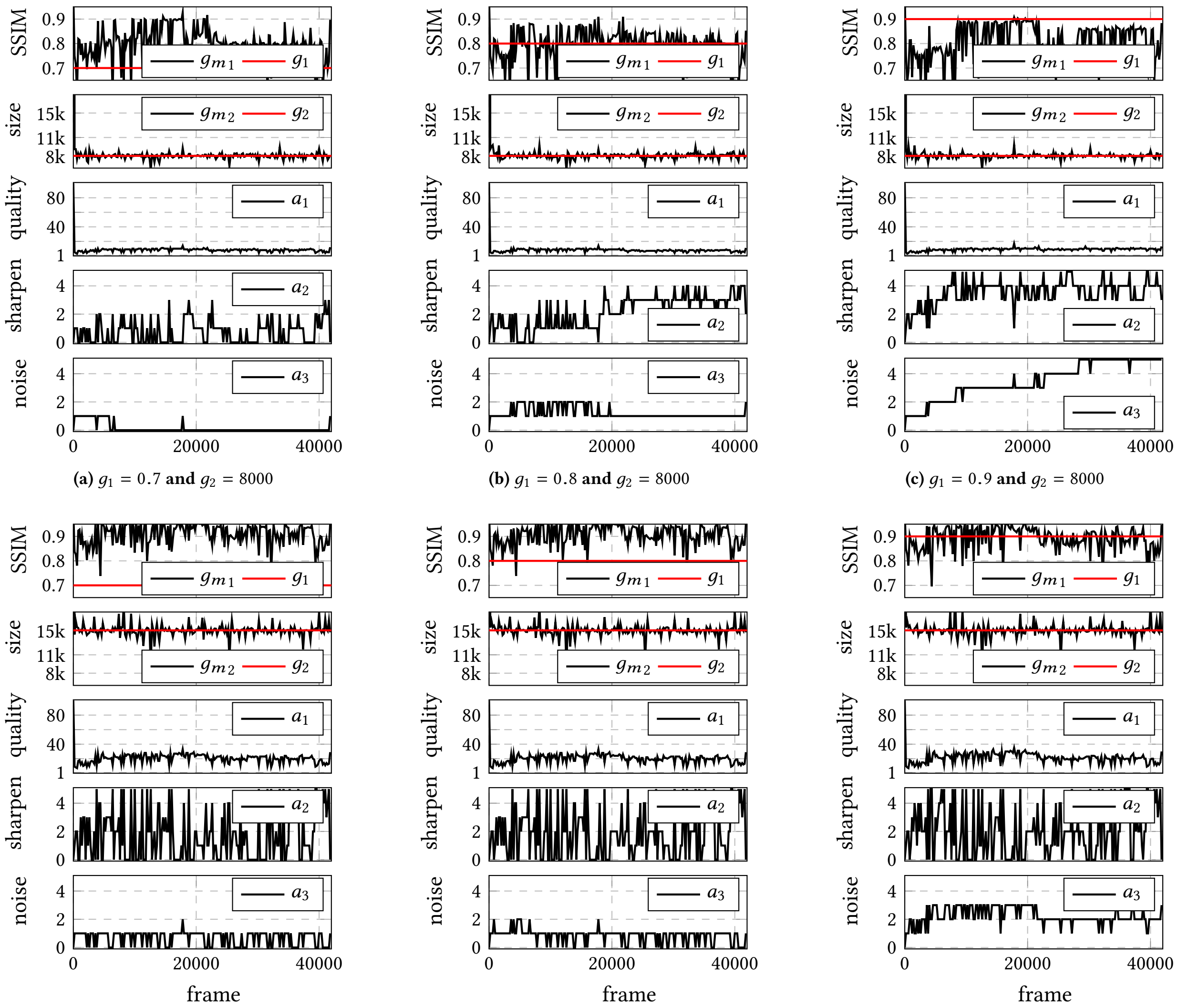
In this section we present the application of the proposed methodology to three different case studies: the first case study is based on a video encoder, the second on a radar positioning system and the third one on a dynamic binder. Finally, we also show some results about the real-time computation and the overhead of the control signal generation.

5.1 Video Compression

Prior work demonstrated automatic synthesis of a single-input, single-output controller for lossy video compression [15]. We extended this case study to achieve multiple goals using multiple actuators and made it available for comparison with other techniques [42, 43].

The actuators $\mathcal{A}$ are:

- a_1, the same quality parameter used in [15] to specify the compression density. It ranges between $a_{1,\min} = 1$ and $a_{1,\max} = 100$, where 100 preserves all frame details and 1 produces the highest compression. We specify a weight $d_1 = 1000$.



(a) $g_1 = 0.7$ **and** $g_2 = 8000$

(b) $g_1 = 0.8$ **and** $g_2 = 8000$

(c) $g_1 = 0.9$ **and** $g_2 = 8000$

(d) $g_1 = 0.7$ **and** $g_2 = 15000$

(e) $g_1 = 0.8$ **and** $g_2 = 15000$

(f) $g_1 = 0.9$ **and** $g_2 = 15000$

Figure 3: Results for the video experiment.

- a_2, the sharpen parameter, which specifies the size of a sharpening filter to be applied to the image. The size ranges between $a_{2,\min} = 0$ and $a_{5,\max} = 5$ where 0 indicates no sharpening. We select a weight $d_2 = 100000$ for this actuator. Given its reduced range compared to a_1, we would like to use it less.
- a_3, noise, which specifies the size of a noise reduction filter, which also varies between $a_{3,\min} = 0$ and $a_{3,\max} = 5$. We specify a weight $d_3 = 100000$, equivalent to sharpen.

The goals $\mathcal{G}$ include:

- g_1, the SSIM [56] that quantifies the similarity between the original and compressed frames. SSIM is a unitless metric that ranges from 0 to 1, with near 1 indicating similar images. As SSIM is between 0 and 1, we use weight $w_1 = 1000$ so that the corresponding component of the cost function J is in the hundreds;
- g_2, the frame size (in kilobytes). We use a weight $w_2 = 0.0001$ so this second goal is considered slightly more important than the first. When the controller can reach only one goal, we prefer to hit the size target, making communication predictable.

Clearly, these two goals conflict with one another. When a specific frame size is set, this will correspond to a specific value for the SSIM on the frame. Similarly, if a specific SSIM is reached, the corresponding frame will have a prescribed size. We conduct this test to show how the controller trades off a goal for the other to achieve the optimal value for the cost function.

We run the video compression example using the Obama Victory Speech video[3] with a resolution of 854×480 pixels and with different combinations of goals g_1 and g_2, using a prediction horizon of $L = 4$. Specifically, we run all possible combinations where $g_1 \in \{0.7, 0.8, 0.9\}$ and $g_2 \in \{8000, 15000\}$. Notice that this is a stress test. In fact, even setting the values of quality, sharpen and noise that would achieve the lowest possible SSIM, this value hardly ever

[3] https://www.youtube.com/watch?v=nv9NwKAjmt0

becomes lower than 0.75, therefore the 0.7 setpoint is not feasible. Also, the goals' conflicting nature makes it impossible to reach most goal combinations simultaneously. For example, when $g_1 = 0.9$, the frame size often exceeds 15000.

Figure 3 shows the six different experiments with the different values of g_1 and g_2. In these experiments, there are only two feasible values for the setpoints: (1) $g_1 = 0.8$ and $g_2 = 8000$ (Figure 3b) and (2) $g_1 = 0.9$ and $g_2 = 15000$ (Figure 3f). In all others it is impossible to achieve both the SSIM and frame size setpoints. Therefore, as shown in the figures, the controller opts to reach g_2, which has an higher relevance: $w_2 \times g_2$. In the first row, Figure 3a shows that g_2 and g_{m_2} are basically equal, while the achieved SSIM g_{m_1} is higher than desired. The encoding quality a_1 is kept low and there is no active noise compensation, while the sharpen value a_2 varies during the execution. Figure 3b shows that both the SSIM and the size setpoint are achieved using some sharpening, a small amount of noise reduction, and a quality similar to that used for the previous combination of setpoints. When the SSIM goal is increased – so, information loss should be diminished – even more noise correction and sharpening is added, as shown in Figure 3c. The setpoint g_1 is reached for some frames, but overall the size limit (and its heavier weight in the cost function) leads to SSIM below the setpoint.

Figures 3d, 3e and 3f show the goal $g_1 = 0.9$ and $g_2 = 15000$ can be achieved by selecting the values of a_1, a_2 and a_3 and the controller therefore selects appropriate values to achieve both the setpoints. In the opposite case (Figures 3d and 3e) the size setpoint is achieved, while the similarity index is kept as close as possible.

5.2 Secure Radar System

This second case study features a cyber-physical system: a secure radar. The radar moves on a mobile platform, possibly a drone, and detects small boats that may be pirates [11]. Once the radar has compiled a list of possible pirates, it encrypts it using the Advanced Encryption Standard (AES) [6], and sends it to a centralized location where multiple lists are merged. Encryption is necessary to avoid providing information on whether pirates have been detected.

The secure radar must meet performance goals for both the radar and the encryption. The first goal is to ensure that the software processes frames at the same rate as the sensor produces them, the second is to ensure timely delivery to the central entity that merges the lists. We meet these goals with two actuators: the number of cores allocated to the radar system (all additional cores can be used by encryption) and the processor's clockspeed.

Specifically, the set of actuators $\mathcal{A}$ consists of:

- a_1 is the number of cores assigned to the radar signal processing application. We measure this as a percentage of the available cores, and our test platform has 12. We assume a minimum of one core must be assigned to the radar. We therefore set $a_{1,\min} = 1/12$ and $a_{1,\max} = 1$, any cores not used by the radar processing can be used by encryption. We assign $d_1 = 1$.
- a_2 is a single clock speed to be used for the processor. Our platform, in fact, does not allow us to set a clock speed per core. We measure this value as a percentage, where the hardware supports a minimum speed of 1.6 GHz and a maximum speed of 3.201 GHz[4], $a_{2,\min} = 0.499$ and $a_{2,\max} = 1$. We assign $d_2 = 1$, since we do not want to enforce any actuators precedence.

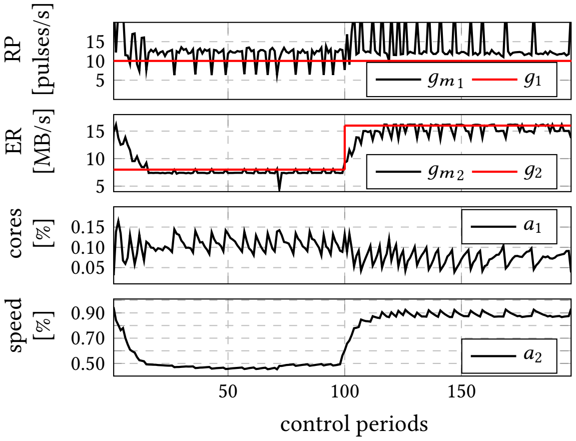


Figure 4: Results for the radar experiment.

The set of goals $\mathcal{G}$ includes:

- g_1, the Radar Performance (RP) measured in radar pulses processed per second. We use an existing radar benchmark [11], which can operate in many different processing modes. For this case study we configure the radar so that it needs to process 10 pulses per second to keep pace with the sensor. Also, $w_1 = 1$.
- g_2, the Encryption Rate (ER) of the AES software. We use OpenAES[5] for encryption and we measure its rate in MB/s. For each possible pirate in each pulse we need to maintain an encryption rate of approximately 0.8 MB/s. To keep up with the radar, we need 8 MB/s per target; *i.e.* the required encryption rate will vary as the number of targets changes. Our control system treats this as a change in setpoint and handles it automatically. Finally, we set $w_2 = 1$ to not privilege any goal.

The combination of actuators and goals used in this case study demonstrates one of the key benefits of the proposed approach. Prior work presented a similar case study of a secure radar system [16]. That case study used a hierarchical control scheme to implement MIMO control and required that both cores and clockspeed be used to manage the radar's performance. This policy leaves no system-level actuators for encryption, which instead is required to switch to a shorter, and less secure, key length to meet encryption performance requirements. In contrast, the technique presented in this paper allows actuators to be used to meet multiple goals. Specifically, clockspeed will be used to meet both goals, meaning that we do not need to reduce security to maintain encryption performance using the proposed technique.

We run the secure radar with the specified goals and actuators and a prediction horizon of $L = 5$. It must maintain a radar performance of 10 pulses/s. Initially, the radar detects a single possible pirate, which requires an encryption performance of 8 MB/s. After 100 control periods, a second possible pirate is detected. This additional target does not affect the radar performance (the same signal processing algorithms are used), but it requires the encryption performance to rise to 16 MB/s.

Figure 4 shows the results of this case study. There are four charts, the top shows the radar performance, the second shows AES performance, the third shows the percentage of cores assigned

[4] Technically, setting the maximum speed turns frequency control over to hardware which can use Intel's TurboBoost to occasionally increase speed beyond the listed maximum for brief periods of time.

[5] http://nalramli.com/OpenAES/

to the radar, and the final chart shows the percentage of the maximum clockspeed used. Solid red horizontal lines show the g_1 and g_2 for each of the radar and AES; g_1 is a constant and g_2 changes when the second target is detected. As shown in the figure, the controller meets both goals, quickly pulling performance down to the desired level for each goal. When the second possible pirate appears, the controller reacts by both increasing clockspeed (which increases performance for both applications) and removing one core from the radar, making it available to AES. As noted above, this use of actuators that affect both goals simultaneously is not possible with prior approaches like [16, 50]. This demonstrated ability to meet multiple goals simultaneously with multiple actuators is a unique contribution of our approach. We note that there is some small amount of oscillatory behavior here due to our use of discrete actuators in this example. There are 12 cores and 12 clockspeeds in the system and when the controller produces a continuous actuation value, we select the highest discrete setting above that value. Therefore, this example also demonstrates the methodology works even with discrete actuators.

5.3 Multi-Objective Dynamic Binding

Dynamic binding is a critical means of adapting Service Oriented Applications (SOAs) [7]. The binding mechanism selects a service to process an incoming request from a set of functionally equivalent alternatives, based on quality criteria. In this experiment, we adopt the same settings of [16]; *i.e.*, the controller has three goals in different, conflicting, dimensions: reliability, performance, and cost.

The controller binds each request to one of three services, and for each service it decides among five different service levels. A higher service level reduces the response time (performance) at a higher cost. Also, we use this case study to show how the solution scales with the size of the problem. The controller uses a prediction horizon of $L = 100$. Given the prediction horizon and the size of the involved matrices, we expect the overhead of the controller execution to be quite high. Because of that, we also executed the controller with the real-time optimization mentioned in Section 4 (setting $\Delta a_{k+1} = \Delta a_{k+2} = \ldots = \Delta a_{k+L-1} = 0$), to show the difference in the resulting trace. In the following we distinguish between the MPC solution and the *MPC fast* solution, which uses the real-time optimization.

More precisely, the set of actuators $\mathcal{A}$ is specified as follows.

- a_1 is the fraction of requests to be served by Service 1. $a_{1,\min} = 0.0$, $a_{1,\max} = 1.0$, $d_1 = 100$ (MPC) and $d_1 = 2000$ (MPC fast).
- a_2 is the fraction of requests to be served by service 2, among those not served by implementation 1; *i.e.*, Service 2 will serve a fraction $a_2 \cdot (1 - a_1)$ of the incoming requests. $a_{2,\min} = 0.0$, $a_{2,\max} = 1.0$, $d_2 = 100$ and $d_1 = 2000$ for MPC fast. Service 3 will serve $(1 - a_2) \cdot (1 - a_1)$ requests. While a linear selection model can be defined, we deliberately used this more complex selection model (from [16]) to demonstrate how the controller handles nonlinear transfer functions by automatic, higher-order linear approximations.
- a_3 is the service level requested to Service 1. $a_{3,\min} = 1$, $a_{3,\max} = 5$, $d_3 = 1$ for the MPC solution and $d_3 = 20$ for MPC fast. The service levels are integer numbers. The controller computes a real value; this value is approximated using a pulse width modulation [36] over an actuation window of 4 time steps. For example, if the reference is 3.73, the actuator will hold level 4 for three steps and level 3 for one step, obtaining an average of 3.75 over the actuation window, which is the closest achievable approximation. The feedback mechanism will take care of the approximation error automatically. The same approximation is used also for a_4 and a_5.
- a_4 is the service level requested to Service 2 $a_{4,\min} = 1$, $a_{4,\max} = 5$, $d_4 = 1$ for the MPC solution and $d_4 = 20$ for MPC fast.
- a_5 is the service level requested to Service 3 $a_{5,\min} = 1$, $a_{5,\max} = 5$, $d_5 = 1$ for the MPC solution and $d_5 = 20$ for MPC fast.

The goals in [16] are prioritized: the controller achieves the reliability goal first, then performance, and finally minimizes the cost in best effort. The proposed MPC controller has no notion of priority, thus we will set the weight of each goal to practically approximate the prioritization scheme of [16]. More precisely, the set of goals $\mathcal{G}$ is the following:

- g_1 is the user-perceived reliability, defined as the fraction of requests served successfully over those received since the last control decision. The weight associated to this goal is $w_1 = 10$
- g_2 is the performance, quantified by the end-to-end response time (in milliseconds). To quantify the error, we measure the average response time since the last control decision. The weight of g_2 is $w_2 = 10^{-1}$
- g_3 is the cost (in 10^{-2}\$). In [16], the cost is a free dimension to be minimized in best effort. To emulate this minimization goal, we will set g_3 close to 0 (non zero to avoid numerical issues); to approximate the best effort priority, we give the goal a low weight: $w_3 = 10^{-10}$.

Each service is configured by three parameters: nominal reliability r_i, performance coefficient t_i, and cost coefficient c_i. For each incoming request, the service implementation flips a fair coin to decide whether to raise an exception or not, according to the nominal reliability r_i. The processing time for each request is sampled from an exponential distribution with mean $t_i/(l_i^2)$ where l_i is the service level at the time of request processing. Notably, the time required to process the request is an inverse quadratic function of the service level, introducing another nonlinearity in the transfer function. The cost of processing an incoming request is $c_i \cdot l_i$. The nominal values r_i, t_i, and c_i are not known by the controller, which can only measure: the time to process a request, how many requests are successful or raise an exception, and the cost of each request.

The experimental results are shown in Figure 5. The configuration parameters of the three services are the following: $\{r_1 = .9, t_1 = 2, c_1 = 15\}$, $\{r_2 = .65, t_2 = 10, c_2 = 10\}$, $\{r_2 = .45, t_2 = 20, c_2 = 5\}$. The experiments last 800 control periods. The setpoints for reliability and performance are changed during the experiment. In the period 0-300, g_1 and g_2 are feasible and achieved with minimal cost; similarly in 300-420, where the reliability setpoint changed. In the period 420-650, an infeasible goal is requested for reliability and the controller goes as close as possible to its satisfaction, while achieving the performance goal. At time 520, the required response time is reduced; the controller achieves this harder goal, though at a higher cost. Finally, in the period 650-800 the reliability goal is also raised; the high reliability and low response time are achieved, though with an higher cost. Finally, as a comparison between the MPC and the MPC fast solution, one can see the difference in the cost (the last plot of Figure 5), around the control period 650, when the cost for the solution presented in [16] is the highest of the three, followed bu the non-optimal solution computed by the the MPC

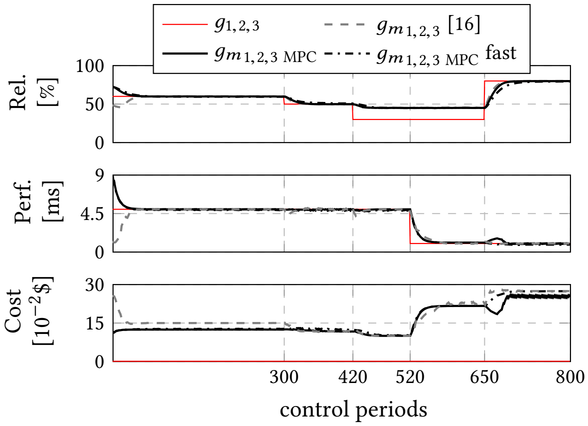


Figure 5: Results for the multi-objective dynamic binder experiment.

fast solver. The best of the three costs is the one achieved by the MPC solution presented in this paper.

As a comparison, the performance of the controller from [16] is also reported in Figure 5. Due to its global optimization capabilities, the MPC controller achieves the reliability and performance goals with a minor cost. The total cost for the solution presented in [16] is 138.49$, for the MPC solution is 124.82$ and for the MPC fast solution is 130.76$, which corresponds to a save of $1.71 \cdot 10^{-2}$$ per time istant for the MPC solution – 9.87% cheaper – and of $0.97 \cdot 10^{-2}$$ for MPC fast – 5.59% cheaper. Indeed, trying to achieve the three goals with a cascade schema, according to the prioritization, [16] cannot guarantee global optimality on all the three dimensions at the same time. Finally, the MPC controller synthesis requires a much smaller learning time, exploring only a small number of system configurations. Notably, [16] uses an online learning mechanism, which can detect changes in the services' performance and adapt the controller online. While in this paper we focus on a static model construction, recursive state space identification [37] can be used to refine the model online, as well as using the measurements from the running system to train a new state space model in parallel and switching when a change is detected [14].

5.4 Control Computation Overhead

Finally, we analyze the cost to compute the control signal for the three given case studies. Figure 6 shows the empirical distribution of the computation times for 10000 executions of the controller code and Table 1 reports some statistics. As can be seen, the video and radar case studies—presented respectively in Section 5.1 and 5.1—are quite fast, with computation times less than 10 *ms*. On the contrary, the optimal solution for the dynamic binding case study—presented in Section 5.3—takes a quarter of a second. Indeed, the dynamic binding problem is costlier because of the longer prediction horizon. Due to the longer execution times, the dynamic binder is a good case study for the optimizations discussed in Section 4—in particular, constraining the solver to find a solution for the current time only while setting the actuator changes for future time instants in the prediction horizon to zero. The faster solution is not optimal, as shown in Figure 5—but it trades optimality for computation time. In fact, the computation times for the MPC fast algorithm is comparable with the times obtained for the other case studies, where the problem size is much smaller.

Table 1: Statistics on Ovehread Data.

Case Study	Average [s]	Standard Deviation [s]
Video	0.00305	0.00074
Radar	0.00471	0.00091
Dynamic Binder	0.20030	0.02332
Dynamic Binder (fast)	0.00184	0.00036

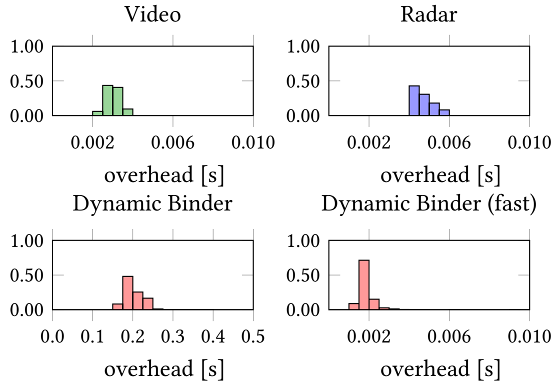


Figure 6: Empirical distribution of the duration of the control signal computation for given case studies.

Whenever a lower computation time is required, the optimization can be turned on with a boolean flag in the code for the controller initialization, making our proposal flexible and capable of accommodating different requirements and execution scenarios.

6 CONCLUSION

We propose a formal method to design self-adaptive software capable of targeting multiple objectives simultaneously. Unlike prior work, our approach exploits all the available tuning parameters that affect the software behavior. Our method is based on system identification and control theory. Through experimentation, it builds an equation-based model of the software system and uses that model to automatically synthesize a model predictive controller. The use of control theory allows us to distinguish between feasible and infeasible objectives, and formally guarantee that the goals are reached whenever feasible. Compared with the state of the art, this is the first contribution that simultaneously uses all available actuators to tackle all objectives.

We combined the theoretical guarantees with tests on different domains, from dynamic binding to radar positioning and video compression. In all our case studies, our proposal has shown that the method is functional and versatile. From the technical standpoint, this advancement opens new perspective because it formally exploits the actuators' inter-dependencies on multiple goals.

ACKNOWLEDGEMENTS

This work was partially supported by the Swedish Research Council (VR) for the projects "Cloud Control" and "Power and temperature control for large-scale computing infrastructures", and by the Swedish Foundation for Strategic Research under the project "Future factories in the cloud (FiC)" with grant number GMT14-0032. Henry Hoffmann's work on this project was partially funded by the U.S. Government under the DARPA BRASS program, by the Dept. of Energy under DOE DE-AC02-06CH11357, by the NSF under CCF 1439156, and by a DOE Early Career Award.

REFERENCES

[1] Konstantinos Angelopoulos, Alessandro Vittorio Papadopoulos, Vítor E. Silva Souza, and John Mylopoulos. Model predictive control for software systems with cobra. SEAMS '16, pages 35–46. ACM, 2016.

[2] Luciano Baresi, Sam Guinea, Alberto Leva, and Giovanni Quattrocchi. A discrete-time feedback controller for containerized cloud applications. In *Proceedings of the 2016 11th Joint Meeting on Foundations of Software Engineering*, FSE 2016, 2016.

[3] Yuriy Brun, Giovanna Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzé, and Mary Shaw. Engineering self-adaptive systems through feedback loops. In *Software Engineering for Self-Adaptive Systems*, pages 48–70. 2009.

[4] E.F. Camacho and C. Bordons. *Model Predictive Control.* Springer London, 2004.

[5] Betty Cheng, Rogerio de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. Software engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems.* Springer, 2009.

[6] Joan Daemen and Vincent Rijmen. *The Design of Rijndael.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[7] Elisabetta Di Nitto, Carlo Ghezzi, Andreas Metzger, Mike Papazoglou, and Klaus Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engineering*, 15(3-4):313–341, 2008.

[8] Yixin Diao, Joseph L. Hellerstein, Sujay Parekh, Rean Griffith, Gail Kaiser, and Dan Phung. Self-managing systems: A control theory foundation. ECBS. IEEE CS, 2005.

[9] Nicolas D'Ippolito, Víctor Braberman, Jeff Kramer, Jeff Magee, Daniel Sykes, and Sebastian Uchitel. Hope for the best, prepare for the worst: Multi-tier control for adaptive systems. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 688–699, New York, NY, USA, 2014. ACM.

[10] Xavier Dutreilh, Aurélien Moreau, Jacques Malenfant, Nicolas Rivierre, and Isis Truck. From data center resource allocation to control theory and back. CLOUD, pages 410–417. IEEE CS, 2010.

[11] Anne Farrell and Henry Hoffmann. Meantime: Achieving both minimal energy and timeliness with approximate computing. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 421–435, Denver, CO, June 2016. USENIX Association.

[12] Antonio Filieri, Carlo Ghezzi, Alberto Leva, and Martina Maggio. Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements. ASE, pages 283–292. IEEE CS, 2011.

[13] Antonio Filieri, Carlo Ghezzi, and Giordano Tamburrelli. Run-time efficient probabilistic model checking. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 341–350, New York, NY, USA, 2011. ACM.

[14] Antonio Filieri, Lars Grunske, and Alberto Leva. Lightweight adaptive filtering for efficient learning and updating of probabilistic models. In *Proceedings of the 37th International Conference on Software Engineering*, ICSE 2015, pages 200–211. IEEE, May 2015.

[15] Antonio Filieri, Henry Hoffmann, and Martina Maggio. Automated design of self-adaptive software with control-theoretical formal guarantees. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE, pages 299–310, New York, NY, USA, 2014. ACM.

[16] Antonio Filieri, Henry Hoffmann, and Martina Maggio. Automated multi-objective control for self-adaptive software design. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 13–24, New York, NY, USA, 2015. ACM.

[17] Antonio Filieri, Martina Maggio, Konstantinos Angelopoulos, Nicolas D'Ippolito, Ilias Gerostathopoulos, Andreas Hempel, Henry Hoffmann, Pooyan Jamshidi, Evangelia Kalyvianaki, Cristian Klein, Filip Krikava, Sasa Misailovic, Alessandro Vittorio Papadopoulos, Suprio Ray, Molzam Sharifloo, Amir, Stepan Shevtsov, Mateusz Ujma, and Thomas Vogel. Software Engineering Meets Control Theory. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, Firenze, Italy, May 2015.

[18] Antonio Filieri, Martina Maggio, Konstantinos Angelopoulos, Nicolás D'ippolito, Ilias Gerostathopoulos, Andreas Berndt Hempel, Henry Hoffmann, Pooyan Jamshidi, Evangelia Kalyvianaki, Cristian Klein, Filip Krikava, Sasa Misailovic, Alessandro V. Papadopoulos, Suprio Ray, Amir M. Sharifloo, Stepan Shevtsov, Mateusz Ujma, and Thomas Vogel. Control strategies for self-adaptive software systems. *ACM Trans. Auton. Adapt. Syst.*, 11(4):1–31, February 2017.

[19] Antonio Filieri, Giordano Tamburrelli, and Carlo Ghezzi. Supporting self-adaptation via quantitative verification and sensitivity analysis at run time. *IEEE Transactions on Software Engineering*, 42(1):75–99, January 2016.

[20] H. Ghanbari, M. Litoiu, P. Pawluk, and C. Barna. Replica placement in cloud through simple stochastic model predictive control. In *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pages 80–87, June 2014.

[21] Graham C. Goodwin, Stefan F. Graebe, and Mario E. Salgado. *Control System Design.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[22] Mark Harman, Yue Jia, William B. Langdon, Justyna Petke, Iman Hemati Moghadam, Shin Yoo, and Fan Wu. Genetic improvement for adaptive software engineering (keynote). SEAMS, pages 1–4. ACM, 2014.

[23] Joseph L. Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M. Tilbury. *Feedback Control of Computing Systems.* John Wiley & Sons, 2004.

[24] Henry Hoffmann. Coadapt: Predictable behavior for accuracy-aware applications running on power-aware systems. In *26th Euromicro Conference on Real-Time Systems, ECRTS 2014, Madrid, Spain, July 8-11, 2014*, ECRTS 2014, pages 223–232, Washington, DC, USA. IEEE Computer Society.

[25] Henry Hoffmann. Jouleguard: energy guarantees for approximate applications. In *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP 2015, Monterey, CA, USA, October 4-7, 2015*, pages 198–214, 2015.

[26] Henry Hoffmann, Martina Maggio, Marco D. Santambrogio, Alberto Leva, and Anant Agarwal. A generalized software framework for accurate and efficient management of performance goals. EMSOFT, pages 1–10. IEEE Press, 2013.

[27] Connor Imes, David H. K. Kim, Martina Maggio, and Henry Hoffmann. POET: a portable approach to minimizing energy under soft real-time constraints. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium, Seattle, WA, USA, April 13-16, 2015*, pages 75–86, 2015.

[28] J.L. Jerez, P.J. Goulart, S. Richter, G.A. Constantinides, E.C. Kerrigan, and M. Morari. Embedded online optimization for model predictive control at megahertz rates. *Automatic Control, IEEE Transactions on*, 59(12):3238–3251, Dec 2014.

[29] Juan L. Jerez, Eric C. Kerrigan, and George A. Constantinides. A sparse and condensed QP formulation for predictive control of LTI systems. *Automatica*, 48(5):999–1002, 2012.

[30] Christos Karamanolis, Magnus Karlsson, and Xiaoyun Zhu. Designing controllable computer systems. HOTOS, pages 9–15. USENIX Association, 2005.

[31] David HK Kim, Connor Imes, and Henry Hoffmann. Racing and pacing to idle: Theoretical and empirical analysis of energy optimization heuristics. In *Cyber-Physical Systems, Networks, and Applications (CPSNA), 2015 IEEE 3rd International Conference on*, pages 78–85. IEEE, 2015.

[32] Basil Kouvaritakis and Mark Cannon. *Model Predictive Control – Classical, Robust and Stochastic.* Springer International Publishing, 2016.

[33] Jeff Kramer and Jeff Magee. Self-managed systems: An architectural challenge. In *2007 Future of Software Engineering*, FOSE, pages 259–268, Washington, DC, USA, 2007. IEEE CS.

[34] D. Kusic, J.O. Kephart, J.E. Hanson, Nagarajan Kandasamy, and Guofei Jiang. Power and performance management of virtualized computing environments via lookahead control. In *Autonomic Computing, 2008. ICAC '08. International Conference on*, June 2008.

[35] Dara Kusic and Nagarajan Kandasamy. Risk-aware limited lookahead control for dynamic resource provisioning in enterprise computing systems. *Cluster Computing*, 10(4):395–408, 2007.

[36] William S Levine. *The control handbook.* CRC, 1996.

[37] Lennart Ljung. *System Identification: Theory for the User.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.

[38] Chenyang Lu, Ying Lu, Tarek F. Abdelzaher, John A. Stankovic, and Sang Hyuk Son. Feedback control architecture and design methodology for service delay guarantees in web servers. *IEEE Trans. Parallel Distrib. Syst.*, 17(9):1014–1027, 2006.

[39] J.M. Maciejowski. *Predictive Control: With Constraints.* Prentice Hall, 2002.

[40] M. Maggio, H. Hoffmann, M.D. Santambrogio, A Agarwal, and A Leva. Controlling software applications via resource allocation within the heartbeats framework. CDC, pages 3736–3741. IEEE, 2010.

[41] M. Maggio, H. Hoffmann, M.D. Santambrogio, A. Agarwal, and A. Leva. Power optimization in embedded systems via feedback control of resource allocation. *IEEE Trans. Control Syst. Technol.*, 21(1):239–246, 2013.

[42] Martina Maggio, Alessandro Vittorio Papadopoulos, Antonio Filieri, and Henry Hoffmann. Self-adaptive video encoder: Comparison of multiple adaptation strategies made simple. In *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '17, pages 123–128, Piscataway, NJ, USA, 2017. IEEE Press.

[43] Martina Maggio, Alessandro Vittorio Papadopoulos, Antonio Filieri, and Henry Hoffmann. Self-Adaptive Video Encoder: Comparison of Multiple Adaptation Strategies Made Simple (Artifact). *Dagstuhl Artifacts Series*, 3(1):2:1–2:3, 2017.

[44] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. Proactive self-adaptation under uncertainty: A probabilistic model checking approach. In *Proceedings of Foundations of Software Engineering*, ESEC/FSE 2015, pages 1–12, New York, NY, USA, 2015. ACM.

[45] Simon Oberthür, Carsten Böke, and Björn Griese. Dynamic online reconfiguration for customizable and self-optimizing operating systems. EMSOFT. ACM, 2005.

[46] T. Patikirikorala, A Colman, J. Han, and Liuping Wang. A systematic survey on the design of self-adaptive software systems using control engineering approaches. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on*, SEAMS, pages 33–42, June 2012.

[47] Raghavendra Pothukuchi, Amin Ansari, Petros Voulgaris, and Josep Torrellas.

Using multiple input, multiple output formal control to maximize resource efficiency in architectures. *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 00:658–670, 2016.

[48] S. Richter, C. N. Jones, and M. Morari. Computational complexity certification for real-time mpc with input constraints based on the fast gradient method. *IEEE Transactions on Automatic Control*, 57(6):1391–1403, June 2012.

[49] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2), May 2009.

[50] Stepan Shevtsov and Danny Weyns. Keep it simplex: Satisfying multiple goals with guarantees in control-based self-adaptive systems. In *Proceedings of the 2016 11th Joint Meeting on Foundations of Software Engineering*, FSE 2016, 2016.

[51] Sebastian Uchitel, Victor A. Braberman, and Nicolas D'Ippolito. Runtime controller synthesis for self-adaptation: Be discrete! In *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '16, pages 1–3, New York, NY, USA, 2016. ACM.

[52] G. van der Veen, J.-W. van Wingerden, M. Bergamasco, M. Lovera, and M. Verhaegen. Closed-loop subspace identification methods: an overview. *Control Theory Applications, IET*, 7(10), July 2013.

[53] Michel Verhaegen and Vincent Verdult. *Filtering and System Identification: A Least Squares Approach.* Cambridge University Press, New York, NY, USA, 2012.

[54] Lixi Wang, Jing Xu, H.A. Duran-Limon, and Ming Zhao. Qos-driven cloud resource management through fuzzy model predictive control. In *Autonomic Computing (ICAC), 2015 IEEE International Conference on*, pages 81–90, July 2015.

[55] Yang Wang and S. Boyd. Fast model predictive control using online optimization. *Control Systems Technology, IEEE Transactions on*, 18(2), March 2010.

[56] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[57] Danny Weyns, M. Usman Iftikhar, Didac Gil de la Iglesia, and Tanvir Ahmad. A survey of formal methods in self-adaptive systems. C3S2E, pages 67–79, 2012.

[58] Chu-Pan Wong, Christian Kästner, Thomas Thüm, and Gunter Saake. On essential configuration complexity: Measuring interactions in highly-configurable systems jens meinicke. ASE. IEEE CS, 2016.

[59] Eric Yuan, Naeem Esfahani, and Sam Malek. A systematic survey of self-protecting software systems. *ACM Trans. Auton. Adapt. Syst.*, 8(4), 2014.

[60] Melanie N. Zeilinger, Davide M. Raimondo, Alexander Domahidi, Manfred Morari, and Colin N. Jones. On real-time robust model predictive control. *Automatica*, 50(3):683–694, 2014.

[61] Qi Zhang, Quanyan Zhu, M.F. Zhani, and R. Boutaba. Dynamic service placement in geographically distributed clouds. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 526–535, June 2012.