Submitted for publication

Comparing Model-Based Predictive Approaches to Self-Adaptation: CobRA and PLA

Gabriel A. Moreno*, Alessandro V. Papadopoulos†, Konstantinos Angelopoulos‡, Javier Cámara§ and Bradley Schmerl§
*Software Engineering Institute, Carnegie Mellon University, USA; Email: gmoreno@sei.cmu.edu
†Mälardalen University, Sweden; Email: alessandro.papadopoulos@mdh.se
‡University of Brighton, United Kingdom; Email: k.angelopoulos@brighton.ac.uk
§School of Computer Science, Carnegie Mellon University, USA; Email: {jcmoreno, schmerl}@cs.cmu.edu

***Abstract*—Modern software-intensive systems must often guarantee certain quality requirements under changing run-time conditions and high levels of uncertainty. Self-adaptation has proven to be an effective way to engineer systems that can address such challenges, but many of these approaches are purely reactive and adapt only after a failure has taken place. To overcome some of the limitations of reactive approaches (e.g., lagging behind environment changes and favoring short-term improvements), recent proactive self-adaptation mechanisms apply ideas from control theory, such as model predictive control (MPC), to improve adaptation. When selecting which MPC approach to apply, the improvement that can be obtained with each approach is scenario-dependent, and so guidance is needed to better understand how to choose an approach for a given situation. In this paper, we compare CobRA and PLA, two approaches that are inspired by MPC. CobRA is a requirements-based approach that applies control theory, whereas PLA is architecture-based and applies stochastic analysis. We compare the two approaches applied to RUBiS, a benchmark system for web and cloud application performance, discussing the required expertise needed to use both approaches and comparing their run-time performance with respect to different metrics.**

***Keywords*-Self-adaptation; adaptive system; model predictive control; latency; CobRA; PLA;**

I. Introduction

Modern software-intensive systems are required to provide service that can justifiably be trusted in spite of having to operate under uncertain run-time conditions that might include changes in the environment (e.g., resource variability, workload fluctuations) or the system itself (e.g., faults). Over the last decade, self-adaptation and autonomic computing approaches [1], [2] have explored different solutions to endow systems with the ability to adapt their structure and behavior at run time in response to changes, as a way of improving different system aspects such as performance or reliability.

While early approaches to self-adaptation tended to be *reactive* [3], adapting the system in response to changes and without reasoning about the long-term outcome of adaptation, some recent proposals [4]–[7] have shifted towards a *proactive* paradigm in which the ability to learn, predict, and act *ahead of time* (i.e., before the conditions that demand adaptation are actually given) are leveraged to improve the run-time behavior of the system. In particular, while reactive approaches tend to work well in systems in which the *adaptation latency* (i.e., the time it takes for an adaptation to become effective in the system) is low, this type of adaptation is sub-optimal and easily outperformed by proactive approaches in common application domains like cloud computing, where the latency is high. In such domains, adaptations like provisioning a new virtual machine for scaling out the system can take up to several minutes (leaving room for situations in which a reactive approach might spin up virtual machines to deal with a transient spike in load that would be gone by the time the virtual machines become active).

CobRA [6] and PLA [7], [8] are among the approaches on proactive self-adaptation that address situations like the one above. They have shown promising results in improving system performance and resilience by employing some of the main ideas behind model predictive control (MPC) [9], such as: (i) the use of models to predict future system behavior, (ii) the computation of a sequence of control actions, committing only to the first one, and (iii) the use of a *receding horizon* to make control more robust against disturbances and possible unpredictable system behavior not captured by models.

Despite the fact that CobRA and PLA both take inspiration from MPC, the two approaches present important differences in the models, reasoning mechanisms, and type of actuation employed to adapt the system. One of the major differences concerns the type of *prediction* employed to compute the optimal control strategy: while PLA employs an explicit representation of the predicted environment behavior to compute future combined system-environment behavior, CobRA does not include any prediction of the environment behavior but employs a dynamic system model combined with a Kalman Filter to predict future states. The second major difference affects *actuation*, which is carried out in CobRA by setting different control parameter values for actuation, whereas PLA executes adaptations on the system via *tactics*, which can range from changing parameter values, to carrying out system-wide changes. These divergences between approaches entail different levels of required engineering effort, limitations, and performance variations under different circumstances, which should be well-understood by researchers and practitioners.

In this paper we describe a comparison between CobRA and PLA, which contributes to the existing body of knowledge on self-adaptive systems by: (i) Enabling a better understanding of the trade-offs of each of the approaches, as well as their

suitability to different types of adaptation scenarios, and (ii) providing a template that can be adapted to carry out comparisons of other predictive approaches.

We illustrate our comparison on the Rice University Bidding System (RUBiS) [10], an open-source application widely employed as a benchmark in cloud computing [11]–[13] and for evaluation of self-adaptive solutions [7], [14]. To enable a fair comparison of the two approaches, we ran our experiments in a simulation of RUBiS. In that way, we were able to replicate experiments with exactly the same conditions for the two approaches, avoiding uncontrolled effects that could alter the results of experiments with the real system (e.g., background processes, network delays).

The results of our study show that CobRA and PLA present comparable levels of run-time performance in the general case, although their performance levels are achieved in different ways, and specific situations are better handled by one of the approaches. Moreover, both approaches exhibit robustness against the uncertainty associated with adaptation latency. Our study also shows that the prerequisites for applying these approaches are of a different nature: PLA requires the availability of theories and models for prediction (e.g., queuing models to predict performance), whereas CobRA does not require such models but demands some level of expertise of system dynamics and identification theory.

In the remainder of the paper, Section II presents the details of the adaptation scenario employed for our comparative study. Section III provides a summary of the main ideas behind model predictive control, as well as an overview of CobRA and PLA. Next, Section IV details our comparison and discusses results. Section V describes related work. Finally, Section VI presents conclusions and future research directions.

II. Adaptation Scenario

We illustrate our comparison of predictive approaches to self-adaptation on the Rice University Bidding System (RUBiS) [10], an open-source application that implements the functionality of an auctions website. Figure 1 depicts the architecture of RUBiS, which consists of a web server tier that receives requests from clients using browsers, and a database tier that acts as a data provider for the web tier. Our setup of RUBiS also includes a load balancer to support multiple servers in the web tier, which distributes requests among them following a round-robin policy. When a web server receives a page request from the load balancer, it accesses the database to obtain the data required to render the dynamic content of the page. The only relevant property of the operating environment that we consider in our adaptation scenario is the request arrival rate prescribed by the workload induced on the system.

The system includes two actuation points that can be operationalized by an adaptation layer to make the system self-adaptive and deal with the changing loads induced by variations in the request arrival rate:

- *Server Addition/Removal*. Server addition has an associated latency, whereas the latency for server removal is assumed to be negligible.

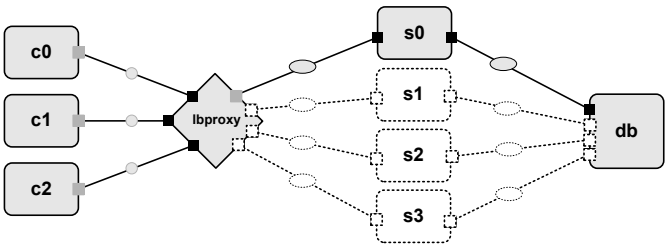


Fig. 1. RUBiS architecture.

- *Dimmer*. The version of RUBiS used for our comparison follows the *brownout* paradigm [14], in which the response to a request includes mandatory content (e.g., the details of a product), and optional content such as recommendations of related products. A *dimmer* parameter (taking values in the interval $[0, 1]$) can be set to control the proportion of responses including optional content.

TABLE I
Requirements for RUBiS.

Functional Requirements	
R1	The target system shall respond to every request for serving its content.
R2	The target system shall serve optional content to the connected clients.
Non-Functional Requirements	
NFR1	The target system shall demonstrate high performance. The average response time should not exceed T.
NFR2	The target system shall provide high availability of the optional content. Subject to NFR1, the percentage of requests with optional content (i.e., the dimmer value d) should be maximized.
NFR3	The target operating system shall operate under low cost. Subject to NFR1 and NFR2, the cost (i.e., the number of servers s) should be minimized.

The goals of the target system are summarized in two functional and three non-functional requirements (Table I). Note that there is a strict preference order among the non-functional requirements that deal with optimization, so trade-offs among different dimensions to be optimized are not possible (i.e., no solution should compromise on the maximization of the percentage of requests with optional content to reduce cost). The imposition of a preference order is aimed at better capturing real scenarios and is not related to limitations to any of the compared approaches, which are also able to capture non-strict preference orders among requirements.

We capture non-functional requirements of our adaptation scenario formally in a utility function that enables us to quantify the quality of their satisfaction. This utility function is defined based on two other functions, namely:

- Utility associated with *revenue* per time interval:

$$U_{R_\tau} \triangleq \tau \cdot a \cdot (d \cdot R_O + (1 - d) \cdot R_M)$$

where τ is the length of the interval, a is the average request rate, and d is the dimmer value. R_M and R_O are the rewards for serving a request with mandatory and optional content, respectively, with $R_O > R_M$.

- Utility associated with *cost* per time interval:

$$U_{C_\tau} \triangleq \tau \cdot c \cdot (s^* - s)$$

where s^* and s are the maximum available servers and the number of servers employed during the interval respectively. c is a constant capturing the cost of a server per time unit.

Based on the definitions above, the utility obtained in an interval is given by

$$U_\tau \triangleq \begin{cases} U_{R_\tau} + U_{C_\tau} & \text{if } r \leq T \wedge U_{R_\tau} = U^*_{R_\tau} \\ U_{R_\tau} & \text{if } r \leq T \wedge U_{R_\tau} < U^*_{R_\tau} \\ \tau \ \min(0, a - \kappa) R_O & \text{if } r > T \end{cases} \quad (1)$$

where $U^*_{R_\tau} = \tau a R_O$ denotes the optimal revenue utility achievable in the interval, and κ is the maximum request rate that the site is expected to handle. Hence, for requests served within the acceptable time threshold T, U_τ returns the addition of the utilities corresponding to revenue and cost, provided that the revenue utility is the optimal achievable, and only the revenue utility if it is sub-optimal (hence imposing the strict preference order between utility and cost). In contrast, if the request is served above T, U_τ returns a penalty directly proportional to the maximum amount of requests with optional content potentially lost during the interval.

III. Model-Based Predictive Self-Adaptation

We chose to compare recent approaches to self-adaptation that are formulated as an optimization problem over a prediction horizon and employ some of the main ideas behind model predictive control (MPC) [9], namely: (i) the use of models to predict future system behavior, (ii) the computation of a sequence of adaptation actions, and (iii) the use of a receding horizon (i.e., repeating the computation of the sequence of control actions from the *actual system state* after the execution of the first control action to account for potential disturbances). These characteristics provide a common ground for comparing CobRA and PLA, which are overviewed in this section.

A. Proactive Latency-aware Adaptation (PLA)

Proactive Latency-Aware Adaptation (PLA) is an approach that fits the general closed-loop control MAPE-K model [15]. The different MAPE stages share a knowledge base that integrates them, and cover the activities that are carried out by the control loop, namely: (i) monitoring the system and the environment; (ii) analyzing the information monitored and deciding whether the system has to adapt; (iii) planning the best course of action for adaptation; and (iv) executing adaptation. The aforementioned notional elements are realized in PLA as follows (Figure 2):

Knowledge Model. In line with architecture-based self-adaptation approaches [16], PLA employs an abstract representation of the system and its environment that captures important characteristics and properties employed to reason about adaptations. More specifically, the knowledge base captures information about:

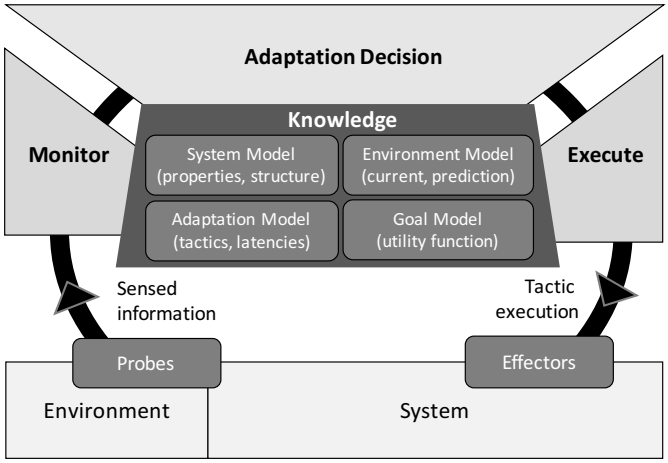


Fig. 2. The PLA adaptation loop.

- *Managed subsystem*. Properties related to the system to be controlled. In RUBiS, this includes the number of servers connected to the load balancer and able to serve requests, the maximum number of servers supported, the dimmer setting, and the observed average response time.
- *Managing subsystem*. Properties related to the possible adaptations, such as the latency of adaptation tactics, and the progress of their execution at run time.
- *Environment*. Properties of the environment relevant to the adaptation decisions. In our scenario, we only consider the observed request arrival rate, and estimations of arrival rates in the near future.
- *Goals*. System goals are encoded as a decision utility function $\hat{U}_\tau$, which assigns utility per measurement interval and has to be maximized. In RUBiS, the decision utility function is a simple additive utility function implementing U_τ (defined in Eq. (1)), with an additional consideration employed to avoid unstable solutions that would make it very difficult to regain control of the system. In particular, in a case in which all the configurations would exceed the target response time, the utility function could choose the one with the smallest number of servers to minimize cost. This is not the right decision because removing resources from an overloaded system would cause the backlog of requests to increase at a higher rate, making the recovery of the system in subsequent decisions even more unlikely. Therefore, an exception to this rule is included in $\hat{U}_\tau$ to favor the configuration with the most servers and lower dimmer setting in such a case.

Monitoring. Observations about the current state of the system and the environment are collected and aggregated as needed to update the model. In RUBiS, the request arrival rate at the load balancer is monitored, and its average and standard deviation are reflected in the model. Concerning the system, architectural changes are also reflected in the model (e.g., a server is marked as "active" in the model when it finishes booting and is effectively connected to the load balancer).

Adaptation Decision. The analysis and planning stages of MAPE-K are combined into a single activity in PLA. The decision-making activity can be regarded as *analysis* from the perspective that it is trying to resolve the need to adapt by determining if there is a configuration that will yield a higher utility than the current one. However, it can also be regarded as *planning*, since the same process entails actually finding that higher-utility configuration. Concretely, the high-level decision that the adaptation is making is choosing which adaptation tactic(s) should be started at the current time instant (if any) to maximize the aggregate utility the system will provide over the rest of its execution (bounded by the horizon).

PLA discretizes the execution timeline in decision periods of duration τ, and solves the adaptation decision problem at the start of each period. The process starts with encoding the behavior of the system and the environment from the current execution instant until the time horizon into processes that are composed in parallel to build a Markov decision process (MDP) model. The key idea behind the MDP model is leaving the decision to execute adaptation tactics under-specified in the process that encodes the behavior of the system via nondeterminism. In such a way, an MDP solver (e.g., a probabilistic model checker [7] or stochastic dynamic programming-based [8]) can be employed to resolve that nondeterminism in a way that maximizes the utility over the horizon, resulting in the synthesis of a strategy that specifies the tactics (or the lack thereof) that should be executed at every time step to achieve that outcome. However, PLA employs only the information for the first time step and discards the rest, recomputing the entire strategy at the start of the next τ-period to correct potential deviations in the realization of the environment from the prediction employed to solve the adaptation decision, consistent with MPCs.

Execution. The set of tactics computed by the adaptation decision are executed by an execution manager. The execution of tactics is asynchronous with respect to the adaptation decision, so that if a tactic with a latency longer than the adaptation period has to be executed (e.g., addition of a server), the adaptation decision can still be run according to its period. This enables PLA to complement slow tactics with fast ones (as long as they do not interfere with each other), executing them in parallel (e.g., it can change the dimmer value right away, without waiting for an ongoing server addition to complete).

B. Control-based Requirements-oriented Adaptation (CobRA)

The Control-based Requirements-oriented Adaptation (CobRA) approach also follows the MAPE-K model, combining principles from Requirements Engineering and Control Theory. The architecture of CobRA is depicted in Figure 3, where each component corresponds to one of the MAPE stages.

Knowledge Model. CobRA [6] combines requirements and analytical models, along with information from the environment to construct adaptations. More specifically, CobRA's knowledge model is composed of the following elements:

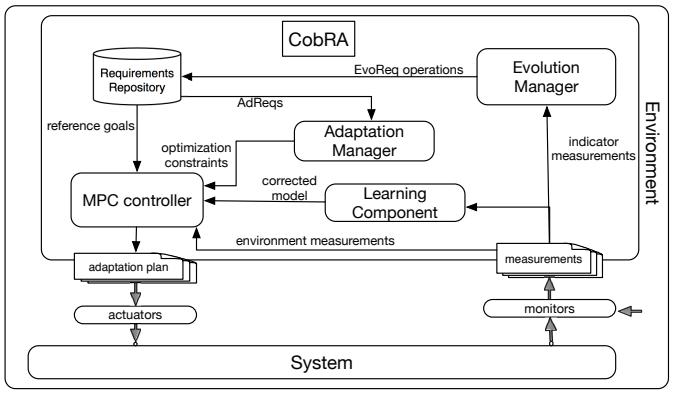


Fig. 3. The CobRA framework.

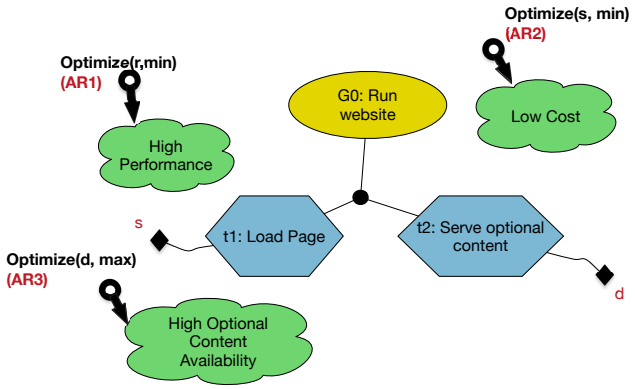


Fig. 4. The target system's goal model.

- *Goal Model*. System goals are captured by models as the one in Figure 4 and refined by following boolean AND/OR semantics. The root goal G0 is refined to two tasks that must both be successfully executed for G0 to be satisfied. The tasks t1 and t2 realize R1 and R2 described in the Table I. The non-functional requirements are captured by the softgoals High Performance, Low Cost, and High Optional Content Availability.
- *Awareness Requirements (AwReqs)*. This type of requirement imposes constraints over other requirements [17]. AR1 and AR2 dictate that the response time r and the number of servers s should always be minimized, whereas AR3 prescribes the continuous maximization of the dimmer's value d. The variables r, s and d, named indicators, are used to measure the satisfaction of the system's AwReqs and constitute the system's output. The difference between the desired value for an indicator and the measured value at run-time is known as the *control error*.
- *Control Parameters*. The variables of the system that can be tuned by CobRA in order to fulfil its goals. These variables constitute the system's input and for our scenario are the number of servers s and the dimmer value d. CobRA discretizes the execution timeline in decision periods of duration τ, and solves the adaptation decision problem at the start of each period, as well as PLA.
- *Adaptation Requirements (AdReqs)*. This type of require-

ment constrains the adaptation process [18] (e.g., by imposing that only one server can be added at a time).

- *Evolution Requirements (EvoReqs)*. This type of requirement accommodates changes over time of the goals [19], e.g. changing the threshold of the acceptable response time.
- *Environment*. Variables that cannot be tuned by CobRA but affect the satisfaction of the system's goals. In our scenario, such a variable is the arrival rate of requests a, that is measured and used as a feedforward signal.
- *Analytical Model*. This model is a mathematical representation of how a change in the control parameters affects the value of the indicators over time [6]. More specifically, the system behavior at the k-th decision period is described by a discrete-time linear dynamic system

$$\begin{cases} x(k+1) = A \cdot x(k) + B \cdot \Delta CP(k) \\ I(k) = C \cdot x(k) + D \cdot a(k) \end{cases} \tag{2}$$

  where $x \in \mathbb{R}^n$ is the *state* of the system, $\Delta CP \in \mathbb{R}^m$ is the set of control parameters variations, and $I \in \mathbb{R}^p$ is the set of indicators. The state x does not always correspond to physical aspects of the system, but it is useful in defining succinctly the relation between CP and I, even in presence of adaptation latency.
- *Cost-Function*. The optimality of CobRA's adaptation is relative to a cost function J which instructs CobRA to minimize the control errors for all indicators by tuning the parameters that require the minimum adaptation effort. More specifically, given the expected values I°, the measured values I and the systems control parameters ΔCP, the cost function minimized by CobRA is:

$$J_k = \sum_{i=1}^{H} \sum_{j=1}^{p} q_j \left(I^\circ_{k+i,j} - I_{k+i,j} \right)^2 + \sum_{j=1}^{m} r_j \Delta CP^2_{k+i-1,j} \tag{3}$$

  where H is the prediction horizon, $q_j > 0$ and $r_j > 0$ are weights associated with the priority of the j_{th} indicator or control parameter, and can be calculated using the Analytic Hierarchy Process (AHP) [20].

Monitoring. CobRA collects information about its environment and the fulfilment of the system's requirements. More specifically, it monitors the arrival rate of requests and the system's indicators.

Adaptation Decision. The first step in the adaptation process is executed by the Evolution Manager component which is responsible for updating the system's goals as prescribed by the EvoReqs of the stakeholders. When the system's requirements are updated, the MPC controller receives as input the expected values for each indicator. In our scenario, the expected values would be $s^\circ = 1$, $r^\circ = 0$ and $d^\circ = 1$. Note that the number of servers and the dimmer value are both control parameters and indicators. This is common for systems where minimization of used resources is required. Then, the MPC controller solves the following optimization problem based on the current request arrival rate and the analytical model:

$$\begin{aligned} \underset{\Delta CP_{k+h}}{\text{minimize}} \quad & J_k \\ \text{subject to} \quad & I_{\min} \le I_{k+h} \le I_{\max} \\ & CP_{\min} \le CP_{k+h} \le CP_{\max} \\ & \Delta CP_{\min} \le \Delta CP_{k+h} \le \Delta CP_{\max} \\ & x_{k+h+1} = A \cdot x_{k+h} + B \cdot \Delta CP_{k+h} \\ & I_{k+h} = C \cdot x_{k+h} + D \cdot a_k \\ & x_k = x(k), \quad h = 0, \ldots, H-1. \end{aligned} \tag{4}$$

The cost function J_k is optimized over a prediction horizon H, based on the analytical model (2), that is an approximation of the actual system. To compensate for the analytical model's imprecision, the Learning Component, which is an implementation of a Kalman filter, corrects the model based on the system's actual behavior.

The solution of (4) is a plan of actions for the considered prediction horizon, and, similarly to PLA, only the first element of the plan is applied; the entire strategy is recomputed at the beginning of the next τ-period.

Execution. The application of the new values of the control parameters is operated by actuators that are domain specific and their implementation is the responsibility of system engineers. The latencies of the actuation process are part of the system's dynamics captured in (2).

C. Main differences between CobRA and PLA

Despite the fact that both CobRA and PLA use some of the basic elements of MPC, they exhibit some relevant differences in the way in which they instantiate these elements:

Prediction. PLA employs an explicit representation of the predicted environment behavior to compute the predicted system-environment combined behavior and the corresponding optimal control strategy. CobRA does not predict the future behavior of the environment and employs a dynamic model combined with a Kalman Filter to predict future states and compute the optimal control strategy assuming that the average workload will be constant over the next τ-period.

Actuation. CobRA is formalized based on the notion of "control increment" (control parameter value ranges are ordered sets) and relies on setting different control parameter values for actuation. In contrast, PLA executes adaptations on the system via tactics, which are captured as scripts whose execution can effect changes that can range from modifying parameter values, to carrying out system-wide changes.

Goal model. While CobRA tries to keep the values of indicators around reference values obtained from the goal model, PLA problem optimization is driven by a utility function that captures goals.

IV. Comparison

The comparison of CobRA and PLA presented in this section aims at facilitating an understanding of their trade-offs and degree of suitability for different adaptation scenarios.

TABLE II
ARTIFACT SPECIFICATION AND EXPERTISE REQUIRED FOR APPLYING COBRA AND PLA.

	CobRA			PLA		
Category	**Artifact**	**Prerequisite**	**Specification**	**Artifact**	**Prerequisite**	**Specification**
Prediction	Dynamic model	Domain knowledge or system identification	Manual specification or automated identification through experiments	Environment model	Domain knowledge and prediction approach	Implementation of automated generation of DTMC encoding of the environment at run time
Adaptation decision	Cost function	–	Manual	Decision utility function	–	Manual
	System model	Convex optimization	Automated	System model	Domain (e.g., Queuing Theory)	Manual
				Tactic latency model	Profiling	Annotation

We compare each approach according to two types of criterion: (i) development time, which is related to the artifacts and the required expertise needed for instantiating the approaches for an adaptation scenario, and (ii) run time, which concerns both adaptation performance and SASO properties (i.e., Stability, Accuracy, Settling time, and Overshoot) [21], [22] under a set of representative adaptation scenarios.

A. Development-time

1) PLA: PLA requires the following artifacts to be produced at development time.[1]

a) Configuration space: The properties of the system that are needed to compute the utility function, or to evaluate applicability conditions of adaptation tactics. In the case of RUBiS, these are the number of active servers, and the dimmer value. In addition, code for generating all the valid configurations at system startup is needed. Doing this requires discretizing continuous properties, such as the dimmer, in a small number of levels. The definition of a configuration and its properties can be implemented either by extending an existing class or by using a generic one that supports dynamic definition at startup.

b) Environment properties: These involves defining the properties of the environment that are needed for computing the utility function. In RUBiS, this is the request arrival rate. This can be done through the same mechanisms of extension or dynamic definition.

c) Environment model: An environment model, encoded as a DTMC, is updated before every adaptation decision. The class that encodes the DTMC is part of the framework, however what is needed at development time is the implementation that generates the DTMC using this class, inserting nodes and probabilistic transitions as needed. In the case of RUBiS, a time series predictor is used to forecast future arrival rates, and the output of this predictor is used to generate an environment probability tree as described in [7].

d) Utility function: The function that computes the decision utility value. This function is invoked during the decision, receiving an instance of a system configuration, and an instance of an environment state, and it must return the computed utility. Since the utility function usually depends on emergent properties of the system, such as response time, the most demanding aspect of implementing this function is computing the estimation of these properties based on system and environment state. In RUBiS, for example, queuing theory equations are used to estimate response time [23], but it is also possible to use performance evaluation tools like OPERA [24].

e) System and tactic models: Models of the system and the tactics specified in Alloy. Although this sounds like particularly burdensome, it is actually straightforward following the patterns shown in [8]. For the system model, the properties of a configuration have to be defined (i.e., the number of servers and dimmer setting for RUBiS). For tactics, there are two templates, depending on whether the tactic has latency or is immediate. The template has to be customized to (i) reflect how the completion of the tactic affects system configuration properties, and (ii) determine the tactic applicability as a predicate over system properties. In addition, a tactic compatibility predicate that indicates whether there is a conflict for concurrent execution with other tactics must be provided, as described in [8].

f) Alloy-to-YAML conversion: PLA-SDP uses Alloy offline to generate reachability predicates that are encoded as YAML files then used at run time. The customization effort for this step requires creating a configuration class in Java with the same system properties defined before, and a method that parses the Alloy output to get the values of these properties. The rest of the interaction with Alloy is already implemented.[2]

2) CobRA: CobRA requires the following artifacts to be produced at development time.

a) Performance indicators: Indicators and performance goals that must be measured to achieve the desired behavior of the system.[3] In the case of RUBiS, these are the response time, the number of active servers, and the dimmer value.

b) Configuration space: The control parameters on which CobRA can act for affecting the identified performance indicators. In the case of RUBiS, these are the number of active servers and the dimmer value. In addition, the definition of the control parameters' saturations is needed in order to set up the optimization problem. Differently from PLA, the control parameters are considered to be continuous values.

[1]We focus on the details of PLA-SDP [8], which requires the most development-time effort of the two PLA solutions.

[2]This step can be completely automated, but since that has not been done yet, it is included in this list.

[3]Performance indicators in PLA are already identified as part of the properties of the system in the configuration space.

For example, in the case of RUBiS, both the dimmer value and the number of servers are considered to be real numbers. The number of servers is therefore actuated with the rounding operator to make it discrete.

c) System model: Requires the definition of a set of discrete-time difference equations, that can be obtained either by exploiting some insight on the system to be controlled, or by using system identification techniques [25]. This step requires expertise of system dynamics and identification theory.

d) Environment properties: Same as in PLA.

e) Cost function: The function (3) that is minimized to make the adaptation decision. In particular, this cost function needs information on the performance goals, tuning with respect to the weights, and the prediction horizon. These design parameters have to be set only once, and affect the obtainable performance in terms of the SASO of the adaptation strategies [22], [26].

f) Python implementation: The implementation of CobRA requires the definition of the system matrices of Equation (2), and the selection of the design parameters for the cost function (3). There is a publicly available implementation of the MPC controller[4], and a developer needs only to write the interface between the system and the MPC controller object.

B. Run-time

In this section, we compare PLA and CobRA according to run-time criteria that include adaptation performance, as well as SASO properties of the adaptation.

1) Experimental Setup: To better compare the two approaches, we ran experiments in a simulation of RUBiS. In that way, we could replicate experiments with exactly the same conditions for both approaches, avoiding uncontrolled effects that could alter the results of experiments with the real system (e.g., background processes, network delays).

We used SWIM, a simulation of web applications with the architecture of RUBiS as shown in Figure 1. To simulate the requests generated by users, SWIM reads the time stamp for each request from trace files, and replays the traces with the requests happening with their recorded interarrival time. The requests arrive at the load balancer, and are forwarded to one of the servers following a round robin algorithm. Each server simulates the processing of requests in the web server, with a maximum of 100 concurrent requests, while the rest waits in a queue. When more than one request is being processed by a server, their execution time is affected by the need to share the processor in the server.

The processing of a request is simulated only in the time that it consumes, not the results it produces. The service time (i.e., the amount of time processing the request would take if there was no contention) is drawn from a normal distribution truncated so that service times are always positive. For our experiments, the mean and variance of the distribution were obtained profiling the service time of RUBiS running in a privately hosted virtual machine. Since SWIM supports brownout, when a request starts processing, it randomly decides whether its response will include the optional content or not according to the current dimmer setting. Depending on the type of response, the service time is drawn from a random distribution that represents the processing of that type of response. All the random number generators used in the simulation are seeded so that it is possible to replicate experiments with the same conditions. In addition, SWIM simulates the effect of caching, with responses taking longer for newly instantiated servers.

[4]https://github.com/apapadopoulos/MPyC

TABLE III
SUMMARY OF ADAPTATION SCENARIOS.

Parameter	**Scenario**					
	1	2	3	4	5	6
Trace	WC	WC	WC	CN	CN	CN
Avg. server boot latency [s]	60	60	180	60	60	180
Std. Dev. Server boot latency [s]	6	20	18	6	20	18

SWIM provides a TCP interface that allows the adaptation manager to access probes and effectors it can use to monitor and execute adaptation actions on the system. The probes provide the following information: dimmer setting, number of servers and active servers, utilization of each server, average request arrival rate, and average throughput and response time for the two kinds of responses. The effectors allow changing the dimmer setting, removing and adding servers. All operations have negligible execution time, except for adding a server, which takes an amount of time configurable in the simulation. This time simulates the time it takes to boot a server, or instantiate a new VM in the cloud.

To simulate the traffic generated by the users of the website, we used traces from the WorldCup '98 trace archive [27], and from the ClarkNet traces [28]. Even though these traces do not correspond to an auctions website, and are several years old, they still represent a significant and realistic traffic for our simulated platform [29]. In fact, both traces were scaled down to last for 105 minutes, and to reach the maximum capacity of the validation setup at their peak. For each trace, we experimented with various server boot latencies governed by different (truncated) normal probability distributions, yielding a total of six scenarios for our study (Table III). The baseline for the standard deviation was selected according to some experimental results on server boot presented in [30].

2) Performance: In this first set of experiments, a threshold of $T = 0.75$ seconds was used to leave a safety margin to the user's tolerable waiting time, estimated to be around four seconds [31], similarly to [14], [32], [33]. Both CobRA and PLA were tuned on the WorldCup trace scenario, in the case of deterministic boot latency, equal to one minute. Then, the scenarios consider a random server boot time (c.f. Table III).

For performance evaluation, we considered: (i) the utility function (1), which captures system requirements and their priorities, (ii) the percentage of served optional content, (iii) the percentage of requests that were served late with respect to the threshold, and (iv) the average servers used along the scenario.

TABLE IV
ADAPTATION PERFORMANCE RESULTS.

Metric	Approach	Scenario (WorldCup'98)		
		1	2	3
Utility	CobRA	4297	4307	4213
	PLA	4156	4133	3863
% Optional	CobRA	97.6	98.5	90.9
	PLA	79.8	79.2	71.4
% Late	CobRA	0.8	0.6	1.3
	PLA	1.2	1.2	1.8
% Avg. Servers	CobRA	1.9	1.9	1.9
	PLA	1.8	1.8	1.8

Metric	Approach	Scenario (ClarkNet)		
		4	5	6
Utility	CobRA	5378	5315	5266
	PLA	5306	5321	5244
% Optional	CobRA	89.0	85.8	82.9
	PLA	69.1	69.0	65.4
% Late	CobRA	3.3	3.4	3.8
	PLA	0.6	0.5	0.4
% Avg. Servers	CobRA	3.0	3.0	3.0
	PLA	2.8	2.7	2.8

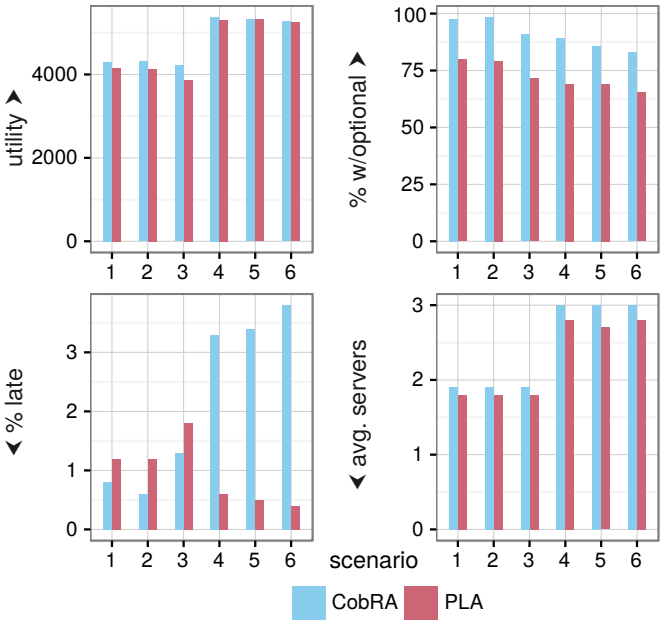


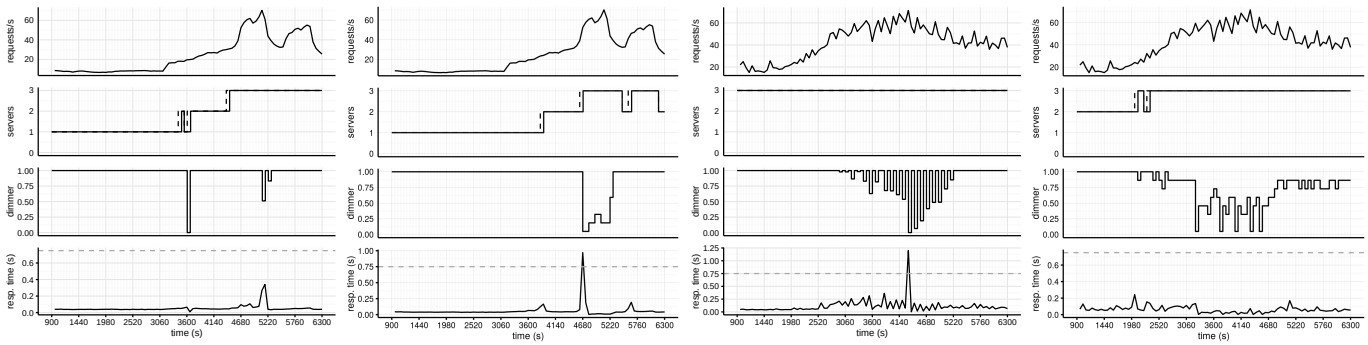
Fig. 5. Adaptation performance results.

Results are shown in Figure IV-B2 and Table IV.

Focusing only on the utility metric (the higher the better), CobRA performs slightly better than PLA across all scenarios, except for Scenario 5, in which PLA does slightly better. Both approaches are robust against the uncertainty associated with server boot time. If the percentage of optional content is considered (the higher the better), CobRA manages to serve between 17% and 19% more requests with optional content. This aspect becomes really relevant when dealing with e-commerce websites, where it has been shown that serving the optional content can increase the revenue of the provider by up to 50% [34]. However, this aspect is only one of the factors that contribute to utility according to (1), which forces a trade-off with the timeliness of the responses, given that when the average response time in a period exceeds the threshold T, a penalty is imposed. Considering the percentage of late requests, CobRA performs better (lower is better) in the WorldCup scenarios than under the ClarkNet workload. This is due to the fact that CobRA was tuned considering a workload with less continual oscillations, and the selected design parameters are pushing for optimizing performance, rather than robustness against abrupt changes in the workload. In contrast, PLA manages to keep this metric almost constant across all scenarios, showing much better performance with respect to CobRA in the ClarkNet scenarios. Finally, for the average number of active servers (the lower the better), PLA manages to always provide reasonably good performance, while using less servers. This aspect is consistent with the results observed for the amount of optional content served. Indeed, the higher the served optional content, the higher the number of servers needed to keep low the response time. Since CobRA serves more optional content, it also needs more servers to keep the response time under the threshold. However, the utility function (1) gives a reward for saving resources if the dimmer is at the maximum setting. Unlike CobRA, PLA explicitly considers this reward in its decision utility function and is more conservative with server usage.

For space limitation, we include only the graphs for the execution of Scenarios 1 and 4 (Figure 6) with both approaches. The solid line in the servers plot represents active servers, whereas the dashed portion indicates a server that is booting but not active yet. In both scenarios, the two approaches achieve similar values for the utility function, but their runtime decisions look quite different. In general, CobRA pushes a bit more for the maximization of the optional content, while PLA pushes more on the minimization of the number of servers. Similar behavior can be observed also for the other scenarios. In these figures it is also possible to observe the difference in dimmer control for the two approaches, with PLA using only the eight levels in which the dimmer setting was discretized for this system. In Scenario 1 (Figures 6a and 6b), PLA has a response time violation at the 4680 second mark. Since PLA uses a model of the predicted environment behavior that depends on a time series predictor, this is probably a case in which the time series predictor failed to estimate the rapid increase in traffic, thus postponing the addition of a server. In addition, it is possible to see that in Scenario 4 (Figures 6c and 6d), the workload is more crooked than in the first scenario, and this makes CobRA act more on the dimmer value, especially during the high peaks of incoming requests.

3) SASO Properties: For the second set of experiments, we considered a trace with a constant average arrival rate of 30 requests per second, and we evaluated the transient with respect to a cold start. In particular, we considered two types of scenarios: (i) when the minimum amount of servers is initially allocated (Scenario Low), and (ii) when all the servers are initially allocated (Scenario High). These two types of scenarios are needed since the system behaves differently



(a) CobRA (Scenario 1) (b) PLA (Scenario 1) (c) CobRA (Scenario 4) (d) PLA (Scenario 4)

Fig. 6. Results obtained with the two frameworks in Scenario 1 and 4.

when scaling up or down, due to its nonlinear nature. We considered the same probabilistic characterization for server boot time presented in Table III for the two scenarios.

In this context, the regular definitions of the SASO properties do not apply, since there is no setpoint for the considered problem. Therefore, we adapted the definitions as follows:

- *Stability* tells if the behavior of the system converges to a constant value. It is a binary property.
- *Accuracy* is related only to the response time, and it is defined as the average response time exceeding the threshold T over a run of N τ-periods:

$$A = \frac{1}{N} \sum_{t=1}^{N} \max\left(r_t - T, 0\right).$$

- *Settling time* is the minimum time for converging to a stable solution.
- *Overshoot* is the maximum value of the residual response time exceeding the threshold during the transient:

$$O = \max_{t=1\ldots N} \left(r_t - T, 0\right).$$

TABLE V
SASO PROPERTIES RESULTS.

Metric	Approach	Scenario (Low)		
		1	2	3
Stability	CobRA	✓	✓	✓
	PLA	✓	✓	✓
Accuracy	CobRA	2.1×10^{-4}	3.1×10^{-4}	2.6×10^{-3}
	PLA	2.1×10^{-4}	1.4×10^{-3}	1.9×10^{-4}
Settling time	CobRA	300	300	480
	PLA	180	120	300
Overshoot	CobRA	0.77	1.12	5.18
	PLA	0.77	3.69	0.67
Metric	**Approach**	**Scenario (High)**		
		4	5	6
Stability	CobRA	✓	✓	✓
	PLA	✓	✓	✓
Accuracy	CobRA	0	0	0
	PLA	0	0	0
Settling time	CobRA	300	300	300
	PLA	60	60	60
Overshoot	CobRA	0	0	0
	PLA	0	0	0

Table V summarizes the obtained results. When considering the SASO properties, PLA performs better than or comparably to CobRA in all the metrics. In particular, PLA converges to a solution faster than CobRA, showing a lower settling time in all the experiments. This is related to the fact that CobRA is an iterative process that requires some time to converge to a stable solution. In fact, in the experiments, CobRA tries to reduce the number of servers before converging to the stable solution, and this requires some more iterations.

4) Discussion: The results of our comparison show comparable levels of run-time performance for CobRA and PLA. However, we have observed that different situations are handled more gracefully by one or the other approach, depending on the specific run-time conditions. These observations lead us to identifying some considerations that have to be made when choosing an approach:

- *Tradeoffs in tuning CobRA*. The reduced performance of CobRA under workloads that experience continual brusque oscillations (e.g., Scenario 4) highlights the fact that the approach relies on targeted tuning of the design parameters to optimize different concerns that may sometimes be conflicting (in this case, robustness against abrupt changes vs. performance). In contrast, PLA does not require parameter tuning,[5] and decisions are always optimal with respect to what the decision utility function dictates, provided that the prediction of the environment is accurate, as noted next.
- *Reliance on accurate environment prediction of PLA*. The response violation experienced with PLA in Scenario 1 highlights the reliance of the approach on the performance of the time series predictor. If the predictor fails to foresee events in the environment like abrupt increases in workload, the performance of the approach can suffer. It is worth noting that the time series predictor has parameters that could be tuned. However, the tuning in the case of PLA is done only for the environment prediction, independent from the system behavior, for which it does not require tuning. With CobRA, on the other hand, both environment and system behaviors are conflated in a single model.

[5]Although the length of the horizon and the decision interval are parameters that could be tuned to particular domains, no tuning was done, and the heuristics described in [8] were used.

V. Related Work

Comparative evaluation of approaches to self-adaptation has already been carried out in different contexts within the self-adaptive systems community. Angelopoulos et al. [35] report on a comparison between the use of architectural and requirement-centric models for adaptation, identifying the advantages and shortcomings of each approach, and pointing at potential combinations of features that might improve adaptation. Shevtsov et al. [36] compare a control-based and an architecture-based approach to self-adaptation, identifying differences between them in performance and formal guarantees. Cámara et al. compare code-based and architecture-based self-adaptation mechanisms, focusing on performance [37] and probabilistic guarantees on system resilience [38].

Aside from comparative evaluations of adaptive systems, other related work deals with establishing the criteria for evaluation of adaptive properties that can be used as a basis for comparison. Kaddoum et al. [39] and Villegas et al. [22] aim at assessing the impact of self-* properties on different aspects of the system, such as performance, in addition to comparing the adaptive features of different systems. Cámara et al. propose a framework for comparative evaluation of adaptive systems that focuses on resilience [40].

All the aforementioned works target approaches that tend to be reactive in nature, do not require prediction of future system-environment behavior, and work well in settings in which adaptation latency is low and can be overlooked as a first-order comparison element. In contrast, the study presented in this paper compares proactive approaches that incorporate major differences with respect to reactive approaches in the way in which they carry out adaptation. These major differences are mainly derived from the prediction and optimization mechanisms employed to overcome some of the limitations in reactive approaches, and hence demand specific comparison methods that factor in these elements.

Some approaches in cloud computing employ look-ahead control to improve energy consumption and performance [41], [42], as well as for dealing with multiple service level objectives [43]. These approaches offer remarkable performance and efficiency improvements, although they tend to be fairly application-specific. Approaches in the area of service-based systems [5], [44], [45] employ proactive adaptation to improve the quality of service of a system over time. Although these approaches employ prediction to improve the operation of the system, they are latency-agnostic, and their look-ahead is limited. These approaches could be compared following an analogous approach to the one proposed in this paper.

VI. Conclusions

In this paper we compared two predictive approaches for self-adaptive software both in terms of development complexity, and of run-time performance. While the two approaches perform comparably in general, their run-time behavior can be significantly different, both in terms of resource utilization and the ways in which they attempt to maximize performance. Moreover, our observations have highlighted the fact that the two approaches have different dependencies that clearly condition their run-time performance and concern design parameter tuning in the case of CobRA, and times series prediction in PLA. Our study has also shown that the prerequisites for applying these approaches are of a different nature: PLA requires the availability of theories and models for predicting specific types of property (e.g., queuing models to predict performance), whereas CobRA does not require such models but demands some level of expertise of system dynamics and identification theory. In addition to the complementary strengths of both approaches, each is better suited for different kinds of control problems. CobRA is better suited to dealing with continuous control inputs, whereas PLA, being tactic-based, is better for discrete control. This can be observed in Figure 6, with CobRA managing better the dimmer, and PLA controlling better the number of servers. In addition, PLA can naturally deal with the (in)applicability of adaptation tactics depending not only on the system state but also on what other tactics are being executed. These complementing features indicate that a combined approach might lead to improved results over using either of the approaches in isolation.

In future work, we plan to expand our comparison with additional case studies, as well as other model predictive approaches. We also plan to compare approaches based on additional properties, such as resilience. Finally, we intend to generalize the methodology followed for our comparison to build a framework for evaluating proactive self-adaptive approaches that can be reused for further comparisons.

Acknowledgment

This work was partially supported by the Swedish Foundation for Strategic Research under the project "Future factories in the cloud (FiC)" with grant number GMT14-0032.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. [Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution. (DM-0004369).

References

[1] B. H. C. Cheng, H. Giese, P. Inverardi, J. Magee, and R. de Lemos, "08031 – Software Engineering for Self-Adaptive Systems: A Research Road Map," in *Software Engineering for Self-Adaptive Systems, 13.1. - 18.1.2008*, 2008.

[2] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing - degrees, models, and applications," *ACM Comput. Surv.*, vol. 40, no. 3, 2008.

[3] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker, "A survey on engineering approaches for self-adaptive systems," *Pervasive and Mobile Computing*, vol. 17, pp. 184–206, 2015.

[4] R. Calinescu, L. Grunske, M. Z. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic qos management and optimization in service-based systems," *IEEE Trans. Software Eng.*, vol. 37, no. 3, pp. 387–409, 2011.

[5] J. Hielscher, R. Kazhamiakin, A. Metzger, and M. Pistore, "A framework for proactive self-adaptation of service-based applications based on online testing," in *1st European Conference on Towards a Service-Based Internet*, ser. LNCS, P. Mahonen, K. Pohl, and T. Priol, Eds. Springer Berlin Heidelberg, 2008, vol. 5377, pp. 122–133.

[6] K. Angelopoulos, A. V. Papadopoulos, V. E. Silva Souza, and J. Mylopoulos, "Model predictive control for software systems with cobra," in *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '16. New York, NY, USA: ACM, 2016, pp. 35–46.

[7] G. A. Moreno, J. Cámara, D. Garlan, and B. R. Schmerl, "Proactive self-adaptation under uncertainty: a probabilistic model checking approach," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, 2015, pp. 1–12.

[8] ——, "Efficient decision-making under uncertainty for proactive self-adaptation," in *2016 IEEE International Conference on Autonomic Computing, ICAC 2016, Wuerzburg, Germany, July 17-22, 2016*, 2016, pp. 147–156.

[9] E. Camacho and C. Bordons, *Model Predictive Control*, ser. Advanced Textbooks in Control and Signal Processing. Springer London, 2004.

[10] "Rice University Bidding System," http://rubis.ow2.org.

[11] K. Qazi, Y. Li, and A. Sohn, "Workload prediction of virtual machines for harnessing data center resources," in *Proceedings of the 2014 IEEE International Conference on Cloud Computing*, ser. CLOUD '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 522–529.

[12] M. A. Islam, S. Ren, A. H. Mahmud, and G. Quan, "Online energy budgeting for cost minimization in virtualized data center," *IEEE Transactions on Services Computing*, vol. 9, no. 3, pp. 421–432, May 2016.

[13] S. Duttagupta, R. Virk, and M. Nambiar, "Predicting performance in the presence of software and hardware resource bottlenecks," in *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2014)*, July 2014, pp. 542–549.

[14] C. Klein, M. Maggio, K. Årzén, and F. Hernández-Rodriguez, "Brownout: building more robust cloud applications," in *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, P. Jalote, L. C. Briand, and A. van der Hoek, Eds. ACM, 2014, pp. 700–711.

[15] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[16] D. Garlan, B. R. Schmerl, and S. Cheng, "Software architecture-based self-adaptation," in *Autonomic Computing and Networking*, 2009, pp. 31–55.

[17] V. E. S. Souza, A. Lapouchnian, W. N. Robinson, and J. Mylopoulos, "Awareness requirements for adaptive systems," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2011, pp. 60–69.

[18] K. Angelopoulos, V. E. S. Souza, and J. Mylopoulos, "Dealing with multiple failures in zanshin: a control-theoretic approach," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2014, pp. 165–174.

[19] V. E. S. Souza, A. Lapouchnian, K. Angelopoulos, and J. Mylopoulos, "Requirements-driven software evolution," *Computer Science-Research and Development*, vol. 28, no. 4, pp. 311–329, 2013.

[20] T. L. Saaty, "What is the analytic hierarchy process?" in *Mathematical models for decision support*. Springer, 1988, pp. 109–121.

[21] J. Hellerstein, S. Parekh, Y. Diao, and D. M. Tilbury, *Feedback control of computing systems*. IEEE Press, John Wiley & Sons, 2004.

[22] N. M. Villegas, H. A. Müller, G. Tamura, L. Duchien, and R. Casallas, "A framework for evaluating quality-driven self-adaptive software systems," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '11. New York, NY, USA: ACM, 2011, pp. 80–89.

[23] J. Zhang and B. Zwart, "Steady state approximations of limited processor sharing queues in heavy traffic," *Queueing Systems*, vol. 60, no. 3-4, pp. 227–246, nov 2008.

[24] M. Litoiu and C. Barna, "A performance evaluation framework for web applications," *Journal of Software: Evolution and Process*, vol. 25, no. 8, pp. 871–890, 2013.

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

[26] Y. Diao, J. Hellerstein, S. Parekh, R. Griffith, G. Kaiser, and D. Phung, "Self-managing systems: a control theory foundation," in *Engineering of Computer-Based Systems, 2005. ECBS '05. 12th IEEE International Conference and Workshops on the*, April 2005, pp. 441–448.

[27] M. Arlitt and T. Jin, "A workload characterization study of the 1998 World Cup Web site," *IEEE Network*, vol. 14, no. 3, pp. 30–37, 2000.

[28] M. F. Arlitt and C. L. Williamson, "Web server workload characterization," in *Proceedings of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems - SIGMETRICS '96*, vol. 24, no. 1. New York, New York, USA: ACM Press, may 1996, pp. 126–137.

[29] A. A. Eldin, A. Rezaie, A. Mehta, S. Razroev, S. S. d. Luna, O. Seleznjev, J. Tordsson, and E. Elmroth, "How will your workload look like in 6 years? analyzing wikimedia's workload," in *Proceedings of the 2014 IEEE International Conference on Cloud Engineering*, ser. IC2E '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 349–354.

[30] A. V. Papadopoulos, A. Ali-Eldin, K.-E. Årzén, J. Tordsson, and E. Elmroth, "PEAS: A performance evaluation framework for auto-scaling strategies in cloud applications," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, vol. 1, no. 4, pp. 15:1–15:31, 2016.

[31] F. F.-H. Nah, "A study on tolerable waiting time: how long are web users willing to wait?" *Behaviour & Information Technology*, vol. 23, no. 3, pp. 153–163, 2004.

[32] A. V. Papadopoulos, C. Klein, M. Maggio, J. Dürango, M. Dellkrantz, F. Hernández-Rodriguez, E. Elmroth, and K.-E. Årzén, "Control-based load-balancing techniques: Analysis and performance evaluation via a randomized optimization approach," *Control Engineering Practice*, vol. 52, pp. 24–34, 2016.

[33] D. Desmeurs, C. Klein, A. V. Papadopoulos, and J. Tordsson, "Event-driven application brownout: Reconciling high utilization and low tail response times," in *2015 International Conference on Cloud and Autonomic Computing*, Sept 2015, pp. 1–12.

[34] D. Fleder, K. Hosanagar, and A. Buja, "Recommender systems and their effects on consumers: The fragmentation debate," in *Proceedings of the 11th ACM Conference on Electronic Commerce*, ser. EC '10. New York, NY, USA: ACM, 2010, pp. 229–230.

[35] K. Angelopoulos, V. E. S. Souza, and J. Pimentel, "Requirements and architectural approaches to adaptive software systems: a comparative study," in *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2013, San Francisco, CA, USA, May 20-21, 2013*, M. Litoiu and J. Mylopoulos, Eds. IEEE Computer Society, 2013, pp. 23–32.

[36] S. Shevtsov, M. U. Iftikhar, and D. Weyns, "Simca vs activforms: comparing control- and architecture-based adaptation on the TAS exemplar," in *Proceedings of the 1st International Workshop on Control Theory for Software Engineering, CTSE@SIGSOFT FSE 2015, Bergamo, Italy, August 31 - September 04, 2015*, A. Filieri and M. Maggio, Eds. ACM, 2015, pp. 1–8.

[37] J. Cámara, P. Correia, R. de Lemos, D. Garlan, P. Gomes, B. R. Schmerl, and R. Ventura, "Incorporating architecture-based self-adaptation into an adaptive industrial software system," *Journal of Systems and Software*, vol. 122, pp. 507–523, 2016.

[38] J. Cámara, P. Correia, R. de Lemos, and M. Vieira, "Empirical resilience evaluation of an architecture-based self-adaptive software system," in *QoSA'14, Proceedings of the 10th International ACM SIGSOFT Conference on Quality of Software Architectures (part of CompArch 2014), Marcq-en-Baroeul, Lille, France, June 30 - July 04, 2014*, 2014, pp. 63–72.

[39] E. Kaddoum, C. Raibulet, J.-P. Georgé, G. Picard, and M.-P. Gleizes, "Criteria for the evaluation of self-* systems," in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '10. New York, NY, USA: ACM, 2010, pp. 29–38.

[40] J. Cámara and R. de Lemos, "Evaluation of resilience in self-adaptive systems using probabilistic model-checking," in *7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2012, Zurich, Switzerland, June 4-5, 2012*, 2012, pp. 53–62.

[41] M. Gaggero and L. Caviglione, "Predictive control for energy-aware consolidation in cloud datacenters," *IEEE Transactions on Control Systems Technology*, vol. 24, no. 2, pp. 461–474, March 2016.

[42] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," in *Proceedings of the 2008 International*

Conference on Autonomic Computing, ser. ICAC '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 3–12.

[43] H. Ghanbari, M. Litoiu, P. Pawluk, and C. Barna, "Replica placement in cloud through simple stochastic model predictive control," in *Proceedings of the 2014 IEEE International Conference on Cloud Computing*, ser. CLOUD '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 80–87.

[44] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic QoS Management and Optimization in Service-Based Systems," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 387–409, May 2011.

[45] C. Wang and J.-L. Pazat, "A Two-Phase Online Prediction Approach for Accurate and Timely Adaptation Decision," *2012 IEEE Ninth International Conference on Services Computing*, pp. 218–225, Jun. 2012.