Overwhelming Uncertainty in Self-adaptation: An Empirical Study on PLA and CobRA

Jingxin Fan
State Key Laboratory for Novel Software Technology,
Nanjing University
Nanjing, China
jingxinfan.nju@gmail.com

Yanxiang Tong
State Key Laboratory for Novel Software Technology,
Nanjing University
Nanjing, China
tongyanxiang@gmail.com

Yi Qin
State Key Laboratory for Novel Software Technology,
Nanjing University
Nanjing, China
yiqincs@nju.edu.com

Xiaoxing Ma
State Key Laboratory for Novel Software Technology,
Nanjing University
Nanjing, China
xiaoxing.ma@gmail.com

ABSTRACT

Self-adaptation is a promising approach to enable software systems to address the challenge of uncertainty. Different from traditional reactive adaptation mechanisms that focus on the system's current environment state only, proactive adaptation mechanisms predict the potential environmental changes and make better adaptation plan accordingly. PLA and CobRA are two representative approaches to build proactive self-adaptation mechanisms. Despite their different design and implementation details, Proactive Latency-aware Adaptation (*PLA* for shot) and Control-based Requirements-oriented Adaptation (*CobRA* for short) are reported to have a very similar performance in supporting self-adaptation. In this paper, we conduct an in-depth comparison between these two approaches, trying to explain their effectiveness. We separate a proactive self-adaptation mechanism into three modules, namely *system modelling*, *environment predicting*, and *uncertainty filtering*. We identify the design choices of PLA and CobRA approaches, in terms of these three modules. We performed an ablation study on the three modules of PLA and compared their performance with CobRA. Our study reveals **the very important role of uncertainty filtering in supporting self-adaptation**, as well as **the huge impact of a fluctuant environment on a self-adaptation mechanism**. Based on this observation, we briefly discuss a conceptual self-adaptation mechanism, MAPE-U (monitoring, analyzing, planning, executing with uncertainty).

CCS CONCEPTS

• **Software and its engineering** → *Software system structures.*

KEYWORDS

self-adaptation, PLA, CobRA, model predictive control

ACM Reference Format:
Jingxin Fan, Yanxiang Tong, Yi Qin, and Xiaoxing Ma. 2019. Overwhelming Uncertainty in Self-adaptation: An Empirical Study on PLA and CobRA. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 10 pages.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00

1 INTRODUCTION

Modern software systems suffer from uncertainty, such as fluctuant running environment, context-aware requirements, and unexpected operating situations, which often reduces the quality of the service delivered by the system [8]. Self-adaptation has been proposed to eliminate the impact of uncertainty. Typical applications of self-adaption systems include Wireless sensor networks [14], Cyber-physical systems [6], and Cloud computing [19].

Self-adaptation enables a software system to continuously deliver high-quality services through re-configuring its components and functionalities [4]. Different from traditional approaches that adjust a system according to the pre-defined workflow, self-adaptation evolves a system based on the system's running environment. As such, a self-adaptive system can be conceptually described and implemented as a MAPE-K loop, which consists of four phases of monitoring, analyzing, planning, executing, and the domain knowledge that guides the adaptation [12].

Take an adaptive web server system as an example. Figure 1 illustrates a MAPE-K adaptation mechanism for such a system. The mechanism first monitors the managed system by collecting the information of the system and its running environment, such as the number of working servers, the number of arriving requests, and the number of delivered services. Then, after analyzing the collected information (e.g., calculating some high-level indicators such as the average response time of the requests, and the network latency time), the mechanism plans a conclusion on how to adjust the managed system to adapt to the sensed situation, for example, increasing the number of working servers. Finally, the mechanism executes the adaptation on the managed system to complete a MAPE-K loop.

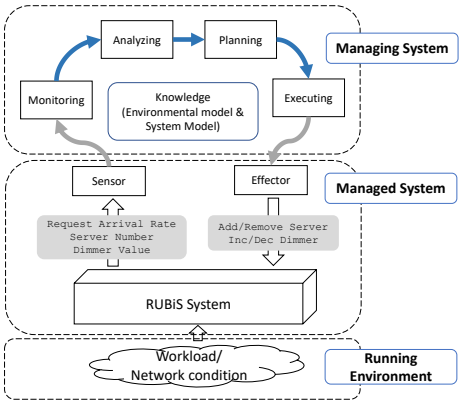

Figure 1: MAPE-K architecture

In the early years, the proposed self-adaptation approaches often use a reactive strategy [16]. More specifically, these approaches plan the adaptation action according to the current environmental information only, and never consider the potential environmental changes in the future. The disadvantage of such a reactive strategy is that it is not fit for the adaptation actions with latency. In the above example, both increasing and decreasing the number of servers requires a reactive time to be activated. As a result, by the time that the number of servers is actually changed, the performed adaptation action might no longer fit the new environmental situation.

To address the limitation of these reactive approaches, researchers have proposed proactive self-adaptation (e.g., PLA [16] and CobRA [1]), which not only considers the current environment situation but also dynamically predicts the future environmental changes. Both PLA and CobRA leverage the idea of model predictive control [20] that describes the problem of predicting future environmental changes by multi-objective optimal programming. However, these two approaches use different strategies and techniques in implementing their mechanism. PLA is an architecture-based approach, which uses a state-machine-based model to describe the managed system and its running environment, while CobRA is a control-based approach, which uses a reactive control-loop to connect the managed system's input and output values, as well as its adaptation actions.

An interesting point around these two approaches is that, despite their different design and implementation details, PLA and CobRA show quite similar performance in supporting self-adaption [18]. This result validates the effectiveness of predicting environmental changes in self-adaptation, but also leaves a new question: *how could a developer choose from PLA and CobRA in her/his self-adaption systems?* In contrast to their similar performance, the suitable applications of PLA and CobRA are different. If the developer has profound knowledge of the managed system and its running environment, PLA is a better choice since it depends on precise modelling of the managed system. If the developer is less familiar with the managed system, but can run the managed system in different situations, CobRA is a better choice since its relatively-simple linear model enables one to identify the system model based on observable execution traces.

In this paper, we conduct an empirical study on PLA and CobRA's effectiveness in supporting self-adaptation and try to give an in-depth explanation of these two approaches' performance. To achieve these, we further compare the workflow of these two approaches and separate each of them into three modules, namely *system modelling*, *environment predicting*, and *uncertainty filtering*. In system modelling, PLA uses a limited processor sharing (LPS) model to describe the managed system, while CobRA uses a linear numeric model. In environment predicting, both PLA and CobRA use model predictive control to consider the future changes; and in uncertainty filtering, PLA uses Kalman filter to posterior adjust its system model, while CobRA uses Kalman filter to posterior compensate its estimation of the system states.

Based on the comparison result, we use ablation study to investigate the effectiveness of these three modules of PLA and compares them with CobRA in different environmental situations. The investigation's result shows that:

- *Imprecise system modelling can be compensated by uncertainty filtering to some extents.*
- *Environment predicting is less effective than precise system modelling and well uncertainty filtering.*
- *Uncertainty filtering has a larger impact on the self-adaptation's performance than the other two modules in handling frequent environmental changes.*

In summary, our empirical study reveals **the very important role of uncertainty filtering** in supporting self-adaptation, as well as **the huge impact of a fluctuant environment on a self-adaptation mechanism**. Based on this observation, we simply discuss a conceptual improvement of PLA, an MAPE-U mechanism (monitoring, analyzing, planning, executing with uncertainty). The main idea of MAPE-U is that we could separate uncertainty filtering from the other parts of self-adaptation.

The main contributions of this paper are as follows.

(1) We conduct an empirical study on the three sub-module's contributions towards PLA and CobRA's effectiveness in supporting self-adaptation.
(2) We empirically find that uncertainty filtering is considerably more important than system modelling and environment erecting in supporting self-adaptation.

In the remainder of the paper, Section 2 introduces our preliminaries, including a web service-based motivating scenario of self-adaptation, and two proactive self-adaption approaches of PLA and CobRA. Section 3 performs an in-depth comparison of sub-modules of system modelling, environment predicting, and uncertainty filtering of PLA and CobRA. Section 4 conducts an ablation study on the three modules of PLA, and compare them with CobRA. Section 5 describes a conceptual self-adaptation mechanism based on our empirical findings. Section 6 introduces the related work and Section 7 concludes the paper.

2 PRELIMINARIES

2.1 RUBiS: a motivating scenario for self-adaptation

We use the RUBiS system as the subject system to be managed by self-adaptation. RUBiS [5] is an open-source web service prototype system. It has been widely used as an evaluation subject by the community of self-adaptive systems [11, 20]. The architecture of

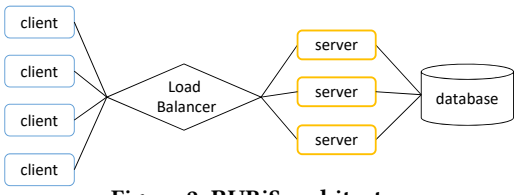

Figure 2: RUBiS architecture

RUBiS mainly consists of three parts, a load balancer, a web server tier, and a database tier, as shown in Figure 2. The load balancer distributes the client requests among the working servers following a round-robin policy. The web server tier includes the clients that accept request from the users and servers that render web pages with the content retrieved from a database. The database tier manages the servers' request to the database and provides the servers with both mandatory data and optimal data.

Considering the limited network and computation resources, RUBiS enables its servers to deliver services of different level of qualities. The web page consisting of mandatory content (produced from the mandatory data) meets a user's basic requirements, while the web page consisting of both mandatory content and optimal content (produced from the optimal data) meets a user's higher-level requirements. As such, service delivered by the servers can be configured by a parameter called "dimmer" [13], which determines the ratio of mandatory content and optimal content rendered by a server. Basically, a lower dimmer value means a low-level quality of the service delivered by the server, which could increase the throughput of the system; a higher dimmer value means the server is now delivering more optimal content, which could decrease the system's throughput.

2.2 RUBiS as an Adaptation System

The RUBiS system is a natural target of self-adaptation. The system has to balance its throughput and quality of service. Such balancing is further affected by the uncertainty such as the network condition and user workload. To precisely measure the gain and lose from the self-adaptation, we could use the following utility function to quantify the quality of the self-adaptation [18]. We use r to denote the response time, d to denote the dimmer value and s to denote the number of working servers. We depict the system's workload by the average request rate α within a time interval τ. The utility function uses two parameters R_M and R_O to represent the system gaining for serving a request with mandatory content and optimal content, respectively.

$$U_\tau = \begin{cases} U_R + U_C & r \le T \wedge U_R = U_R^* \\ U_R & r \le T \wedge U_R < U_R^* \\ \tau \cdot min(0, \alpha - \kappa) R_O & r > T \end{cases} \quad (1)$$

Here U_R represents the utility associated with *revenue* per time interval:

$$U_R = \tau \cdot \alpha \cdot (d \cdot R_O + (1 - d) \cdot R_M) \quad (2)$$

and U_R^* represents the utility when dimmer value is set to 1:

$$U_R^* = \tau \cdot \alpha \cdot \cdot R_O \quad (3)$$

and U_C represents the utility associated with *cost* per time interval:

$$U_C = \tau \cdot c \cdot (s^* - s) \quad (4)$$

Equation (1) describes three entangled requirements. First, the response time r of all the request should never exceed a threshold T. Second, the content quality delivered by a server should be as high as possible, namely the dimmer value d should be maximized. Third, the cost of the system should be as low as possible, namely the number of servers s should be minimized.

RUBiS provides two pairs of adaptation actions to configure the system's provided service. One is `Add/remove a server`, which can be used to change the number of working servers. And another one is `Increase/decrease dimmer value`, which can be used to change the value of dimmer and further indicates the quality of service delivered to the concerned request.

The uncertainty faced by RUBiS can be generally classified into two types, internal uncertainty and external uncertainty. The former concerns the system's running environment, e.g., the upcoming workload and network condition, and the latter concerns the system's control mechanism (e.g., the time latency of booting a server, and the time latency of removing a server's workload and shutdown the server).

2.3 Two Proactive Self-adaptation Methods: PLA and CobRA

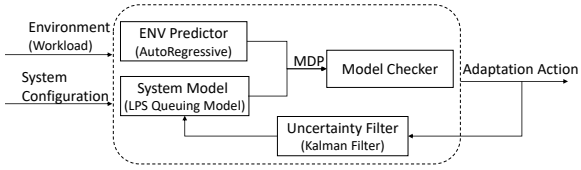

Figure 3: Structure of PLA

PLA is proposed by Gabriel et al. [16] in 2015. It is an architecture-based approach that needs to precisely model the managed system and its running environment. More specifically, PLA relies on a LPS queuing model to describe the managed system's working status and the corresponding environmental dynamics and adaptation actions. By using an auto-regressive model, PLA would dynamically predict the distribution of the system mole's future state. PLA describes the uncertainty based on the Markov decision process (MDP), and tries to eliminate the negative impact of uncertainty by using the Kalman filter. The adaptation action of PLA is determined by using a non-deterministic model-checker to maximize the adaptation goal. Figure 3 illustrates PLA's basic structure.

2.3.1 CobRA. **CobRA** is proposed by Konstantinos et al. [1] in 2016. It is a control-based approach that use control theory to describe the numeric relationship between the managed system's input, output and control actions. More specifically, CobRA uses a linear model to capture the dynamic features of the managed system. CobRA does not predict the future environmental changes explicitly, but regards the potential environmental changes as a kind of fluctuation and leave it to uncertainty filtering. CobRA uses Kalman filter to posteriorly adjust its predicted system states, and compensates it with the possible fluctuation that can be viewed

Table 1: Differences between PLA and CobRA

Features	PLA	CobRA
System Modelling	A LPS queuing model based on accurate modelling	A linear dynamic model based on system identification
Environment predicting	Explicit prediction based on auto-regression model	Implicit prediction based on posterior model adjustment
Uncertainty Filtering	Using MDP to dynamically adjust the model parameters	Using Kalman filter to dynamically compensate the model states

as a prediction of the future environmental states. The adaptation logic determined by CobRA is determined by the control theory. Fig

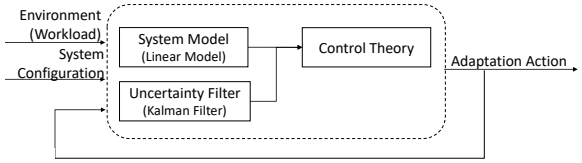

Figure 4: Structure of CobRA

3 COMPARISON BETWEEN PLA AND COBRA

In this section, we summarize the differences between PLA and CobRA, in terms of system modelling, environment predicting and uncertainty filtering, which are presented in Table 1. The detailed comparison is as follows.

3.1 System modeling

One of the most significant differences between the PLA and CobRA is system modeling which quantizes the relationship of the managed system's input, output, and the adaptation actions. PLA regards the managed system as a white-box and pays lots of efforts on precise modeling of the system's behaviour in the running environment. CobRA assumes that the managed system is in the vicinity of an equilibrium point despite uncertainties [22]. Therefore, it regards the managed system as a black-box and identifies system model as a linear model using sampling data in the vicinity of its equilibrium point.

For systems like RUBiS that are exponential and sensitive to the fluctuation of workload, CobRA's linear model may result in a less-effective self-adaptation due to its highly-dynamic running environment. However, the precise system model of RUBiS is a steady-state approximation of LPS queue in heavy traffic [24], and there are significant deviations between predicted indicators and their transient measurements which are relied on uncertainty filtering mechanism to offset. Therefore, the impact of different choices in system modelling should be investigated (the results are presented in Figure 6).

3.2 Environment predicting

PLA and CobRA handle the changes of environment differently. PLA assumes the change of environment is predictable and uses current and historical measurements to construct the probabilistic tree of the future environment. Then, the predicted environment will be used to calculate the accumulated utility for all possible adaptation strategies and the adaptation strategy corresponding to maximum accumulated utility in our adaptation decision. However, CobRA regards the assumed slowly changing environment as disturbances which are handled by closed-loop feedback control. Thus, it does not predict the future environment but uses its measurements as a feed-forward signal to improve the prediction of current indicators which is used in Kalman Filter to obtain the true states of the current system.

For the RUBiS self-adaptive system, the environment is describes as system's workload. PLA uses time series predictor (e.g., the RPS toolkit [7]) to predict future workloads. However, there are always errors in the predicting process, especially when there is no periodic change in the environment. We also study the impact of different prediction methods in our later empirical study (the results are presented in Figure 8).

3.3 Uncertainty filtering

Both of PLA and CobRA have uncertainty filtering modules to tolerate the impact of uncertainties, such as environment fluctuating and modeling error. PLA uses Kalman Filter to dynamically adjust the parameters of the system model. More specifically, since PLA's system model might be built according to the managed system's behaviour in a developing environment, the parameters of the model need to be shifted into the managed system's behaviour in the running environment. As such, PLA uses Kalman filter to dynamically adjust its model's parameters based on the observation and posterior validation of the managed system's current execution. CobRA also uses Kalman filter to handle uncertainty. However, it directly adopts the ideas from control theory and describes the behaviour of the system with the state space equation.

In supporting self-adaptation for RUBiS, PLA adjusts the parameter of service time in the system model by Kalman Filter, so as to realize the feedback, while CobRA directly regulates the state of the system. We finally study the impact of the different usage of Kalman filter of the two mechanisms (the results are presented in Figure 10).

4 AN ABLATION STUDY ON THE EFFECTIVENESS OF PLA AND COBRA

In this section, we conduct an ablation study on the effectiveness of PLA's three modules, namely system modelling, environment predicting, and uncertainty filtering. Our ablation study is trying to answer the following research question:

RQ1: How do the three modules of PLA contribute to its effectiveness in supporting self-adaptation?

We also compare the performance of PLA-based mechanisms with a complete CobRA mechanism, trying to better understand the contribution of PLA and CobRA's different design and implementation choices. More specifically, the major difference between PLA and CobRA is their system modelling and environmental predicting. We want to answer the following research question.

RQ2: How do the different design choices in system modelling and environmental predicting contribute to the different performance of PLA and CobRA?

4.1 Experiment Preparation

Subject. We conducted all the experiments based on the aforementioned RUBiS system. We used SWIM (A Simulator of Web Infrastructure and Management), a discrete event simulator, as the running environment of RUBiS. We injected two different real-world network workloads as the input of RUBiS, one is called "WorldCup [2]", and the other one is called "ClarkNet [3]". We set the values for the server to be in the range of $[1, 2, 3]$ and dimmer to be in [0 1], and the $responseTime$ threshold to be 0.75.

Metrics. For each pass of the experiments, we measured the $utility$ value as the performance of the self-adaptation mechanism achieved. Notice the value of $utility$ is calculated based on a complete trace according to Equation (1). To capture the temporal features of the compared self-adaptation mechanisms, we also reported the values of $responseTime$, $server$, and $dimmer$, respectively.

Settings of PLA and CobRA. Based on our comparison of PLA and CobRA, we derived five different settings of PLA and CobRA, as follows.

PLA_{S1} denotes the complete PLA mechanism, namely with its LPS queuing theory model (system modelling), model predictive controlling (environment predicting), and Kalman-filter-based feedback (uncertainty filtering).

PLA_{S2} denotes the PLA mechanism without system modelling. More specifically, we replace the LPS queuing theory model of the original PLA with a less accurate crude linear model. Considering that the LPS queuing theory model is an exponential model, the building and maintaining of it could bring large time and space overhead to the self-adaptation mechanism.

PLA_{S3} denotes the PLA mechanism without environmental predicting. More specifically, we simply disable the prediction function of PLA and make its adaptation focus on the current system state.

PLA_{S4} denotes the PLA mechanism without uncertainty filtering. More specifically, we remove the implicit feedback module in PLA and make PLA it an open-loop method that never uses observable information to posterior adjust its internal states.

$CobRA_{S1}$ denotes the complete CobRA mechanism, namely with its linear state-space model (system modelling), model predictive controlling (environment predicting), and Kalman-filter-based feedback (uncertainty filtering). We consider only one setting for CobRa since its sub-modules are entangled and cannot be separated as PLA's sub-modules.

Procedures. To answer *RQ1*, we compared the performance of PLA_{S1} and PLA_{S2}, PLA_{S1} and PLA_{S3}, as well as PLA_{S1} and PLA_{S4}, respectively.

To answer *RQ2*, we compared the performance of PLA_{S1} and $CobRA_{S1}$, PLA_{S2} and $CobRA_{S1}$, PLA_{S3} and $CobRA_{S1}$, as well as PLA_{S4} and $CobRA_{S1}$, respectively.

4.2 Experiment Results

RQ1 (Comparison of PLA's three modules). We first compare the achieved utility values of PLA_{S1} and PLA_{S2} in Figure 5, concerning the two different workloads. The result shows that the complete PLA mechanism constantly performs better than the PLA mechanism without accurate system modelling. This is reasonable since the accurate LPS model of PLA_{S1} costs much more time and space efforts, comparing with the linear model of PLA_{S2}. For the two different workloads, PLA_{S1} achieves 30% higher utility value compared with PLA_{S2} on ClarkNet, and 5% higher utility value on WorldCup. The reason for this difference among the two workloads is probably because that ClarkNet suffers more short-term fluctuation compared with WorldCup. Such fluctuant makes the accurate system modelling more effective in coupling with the environment of ClarkNet.

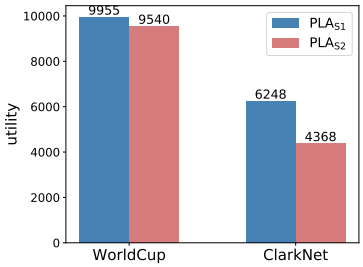

Figure 5: The utility comparison of PLA_{S1} and PLA_{S2}

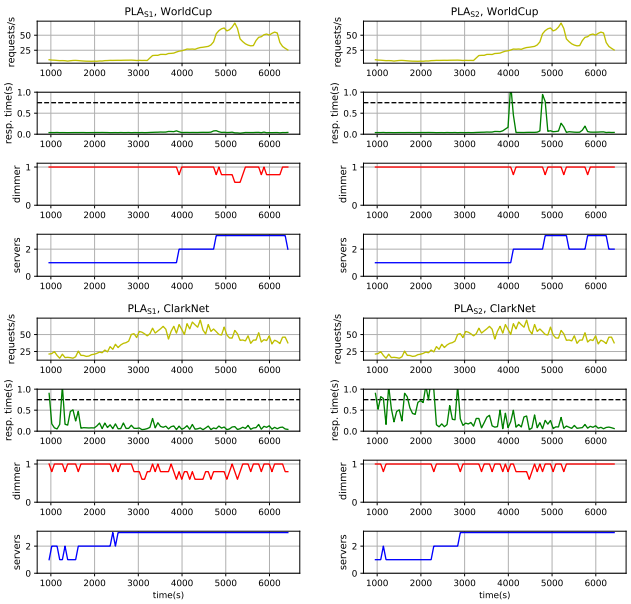

Figure 6: The detailed comparison of PLA_{S1} and PLA_{S2}

Figure 6 compares the detailed temporal features of the two mechanisms, i.e., the request per second, the average response time, the value of dimmer, and the number of servers. The results validate our previous finding that PLA_{S1} outperforms PLA_{S2}. Consider the response time, which reflects the direct adaptation effect, PLA_{S1}'s controlled response time never exceeds the pre-defined threshold (i.e., 0.75 seconds) on WorldCup, and only 2% of its total running time reports a threshold-overshoot response time on ClarkNet. This number is still constantly better than that of PLA_{S2}, whose controlled response time exceeds the threshold for 3% and 10% of its total running time on WorldCup and ClarkNet, respectively.

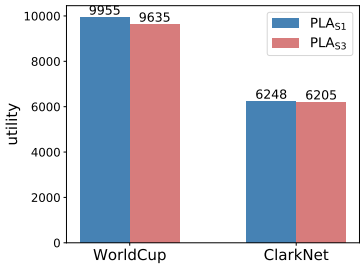

Figure 7: The utility comparison of PLA_{S1} and PLA_{S3}

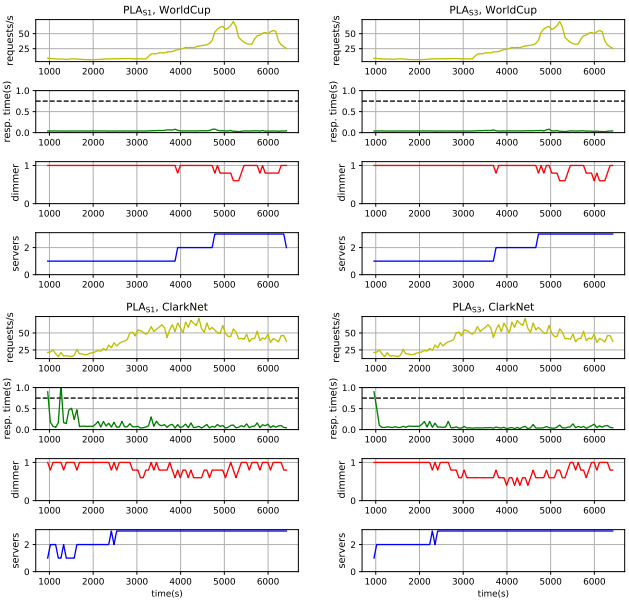

Figure 8: The detailed comparison of PLA_{S1} and PLA_{S3}

We then compare the achieved utility values of PLA_{S1} and PLA_{S3} in the Figure 7. Surprisingly, the reported utility values do not show a significant difference. The gap between PLA_{S1} and PLA_{S3}'s achieved utility values on WorldCup is 3%, and the gap on ClarkNet is mere 0.6%. The detailed temporal features of self-adaptation mechanisms of PLA_{S1} and PLA_{S3}, as shown in Figure 8, also suggest a less effective role of environment predicting in PLA. Since we set a quite loose threshold for the response time, PLA_{S3} even achieves a better performance than PLA_{S1} that it never reports an overshoot on WorldCup, and only 1% of its total running time suffers overshoot on ClarkNet.

Figures 7 and 8 suggest that whether to predict the future environment has no significant influence on the effectiveness of self-adaptation. This is a very interesting finding since both PLA and CobRA claim that their model predictive control, which leverages the prediction of future environmental changes, is the key to their improved performance comparing with those traditional reactive self-adaptation mechanisms. Our conjecture to this finding is that the future environmental changes are encoded in the current system state of both PLA and CobRA's system model. As such, environment predicting is less effective, and can be removed for the reducing of time and space overheads.

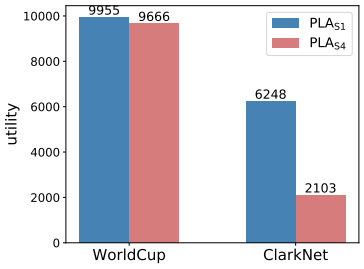

Figure 9: The utility comparison of PLA_{S1} and PLA_{S4}

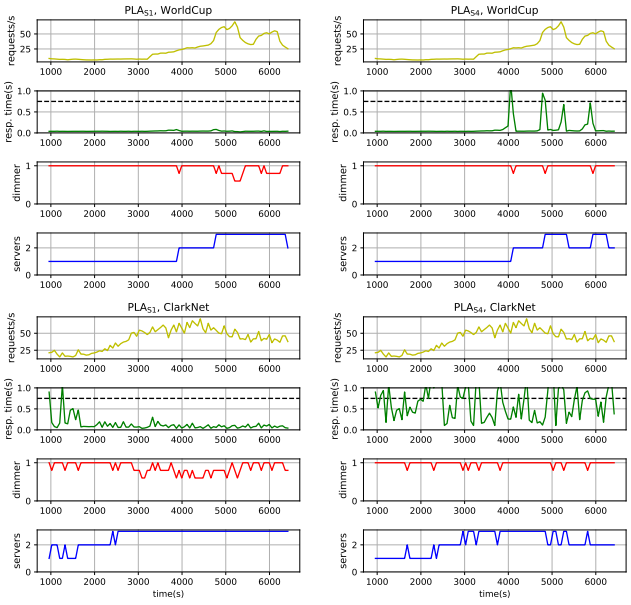

Figure 10: The detailed comparison of PLA_{S1} and PLA_{S4}

Finally, we compare the achieved utility values of PLA_{S1} and PLA_{S4} in the Figure 9. The result shows that when uncertainty filtering is disabled, PLA's achieved utility value suffers a significant drop. On workload WorldCup, the value of utility is reduced by 3%, and on workload ClarkNet, the value is reduced by 60%. When considering the detailed performance features, as shown in the Figure 10, the result is quite similar. PLA_{S4}'s controlled response time exceeds the threshold for 3% and 37% of its total running time on WorldCup and ClarkNet, respectively. Noticing that the number of PLA_{S1} is 0% and 2%, we again find the self-adaptation mechanisms could suffer from short-term fluctuation if the error between its estimated system state and the actual state is not well-addressed.

The reason for the significant effectiveness of uncertainty filtering is that it could affect the accuracy of both system modelling and environment predicting. On the one hand, there is a certain gap between the system model and the actual system, and the uncertainty filtering module can narrow this gap by adjusting the model parameters. On the other hand, environment predicting could has some errors, especially when there is no law to be followed for environmental change. The uncertainty filtering module can make the adaptation controller tolerate the effects of this error to some extent.

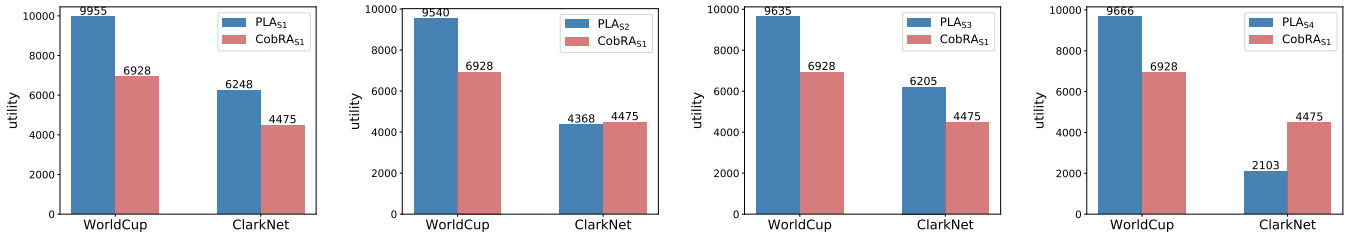

Figure 11: The utility comparison of $CobRA_{S1}$ with PLA_{S1}, PLA_{S2}, PLA_{S3} and PLA_{S4}

In summary, we conclude our answer to the research question *RQ1* as follows. *Environment predicting does not show significant contribution to PLA's effectiveness in supporting self-adaption. The contribution of the remaining two modules, system modelling and uncertainty filtering increase with the growth of the fluctuation that the managed system's running environment suffers. PLA has a greater performance drop when its uncertainty filtering module is disabled, comparing with the drop when the system modelling module is disabled.*

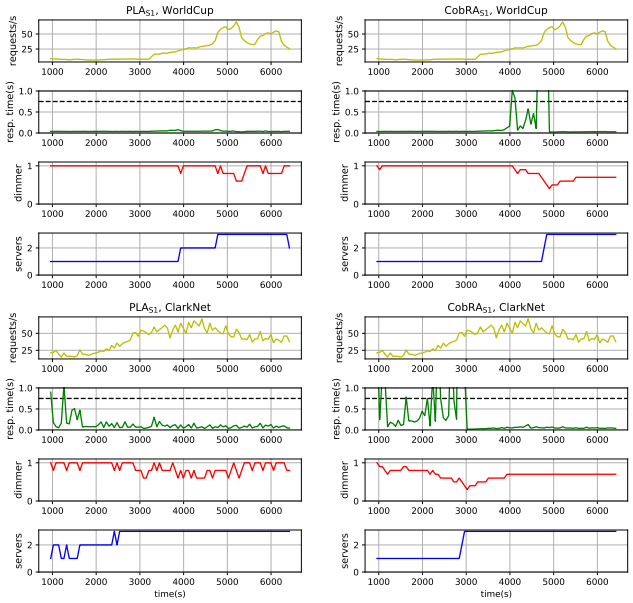

Figure 12: The detailed comparison of PLA_{S1} and $CobRA_{S1}$

RQ2 (Comparison of between PLA and CobRA). We now study the contribution of PLA's and CobRA's different design and implementation choices. The achieved utility values of PLA_{S1}, PLA_{S2}, $PLA_{S3}$and PLA_{S4} comparing with $CobRA_{S1}$ are shown in the Figure 11, concerning the two workloads.

We first compared the performance of a complete PLA mechanism with a complete CobRA mechanism. The result shows that the complete PLA mechanism constantly performs better the complete CobRA mechanism. For the two different workloads, PLA_{S1} achieves 30% higher utility value compared with $CobRA_{S1}$ on WorldCup, and 28% higher utility value on ClarkNet. With respect to the different settings, PLA and CobRA's performance varies on different workloads. For workload WorldCup, all three PLA-based settings perform better than $CobRA_{S1}$. Specifically, when comparing with $CobRA_{S1}$, PLA_{S2} achieves a 27% higher utility value, while both PLA_{S3} and PLA_{S4} achieves a 28% higher utility value. For workload ClarkNet, $CobRA_{S1}$ performs better than PLA_{S2} (2% higher utility value) and PLA_{S4} (53% higher utility value), and performs worse than PLA_{S3} (27% lowere utility value).

To investigate the different performance of the PLA-based approaches and CobRA, we study the detailed temporal features of the compared settings. Figure 12 compares the detailed temporal features of PLA_{S1} and $CobRA_{S1}$. For workload WorldCup, 6% of $CobRA_{S1}$'s running time reports an overshot response time (i.e., response time over the predefined thresholds of 0.75 seconds), and PLA_{S1} never reports such overshoot. For workload ClarkNet, both PLA_{S1} and $CobRA_{S1}$ have 2% of their total running time report an overshot response time. This is due to the different paradigm of these two mechanisms: PLA is an architecture-based approach, which requires the developers have a well-understanding of the managed system; CobRA is a control-based approach, which requires less-accurate understanding and introduces system identification to compensate the inaccuracy.

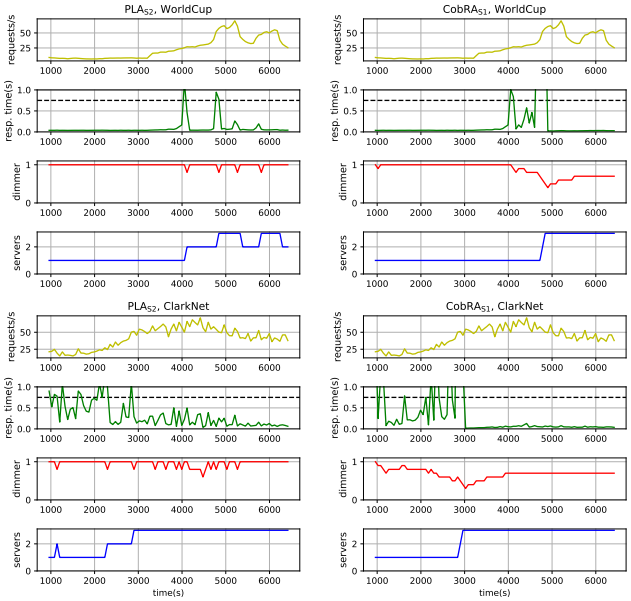

Figure 13: The detailed comparison of PLA_{S2} and $CobRA_{S1}$

Figure 13 compares the detailed temporal features of PLA_{S2} and $CobRA_{S1}$. For the workload WorldCup, only 3% of PLA_{S2}'s total

running time suffers an overshoot response time, while that number of $CobRA_{S1}$ is 6%. For workload ClarkNet, $CobRA_{S1}$ achieves 8% less time of the controlled overshot time (2% compared with PLA_{S2}'s 10%). This different performance is reasonable since in PLA_{S2} we replace its accurate LPS-based system model with an inaccurate linear model. For a stable workload like WorldCup, the limitation of such inaccurate system model could be covered up, while for a fluctuant workload like ClarkNet, the same limitation could be amplified and results in a great performance drop.

Figure 14 compares the detailed temporal features of PLA_{S3} and $CobRA_{S1}$. PLA_{S3} also avoids reporting any overshoot response time on workload WorldCup, and only 1% of its total running time suffers overshoot on workload ClarkNet. The corresponding numbers of $CobRA_{S1}$ is 6% for workload WorldCup, and 2% for workload ClarkNet. Notice that this pair of settings is the only one that a partial PLA mechanism outperforms the complete CobRA mechanism in both two workloads. This validates our previous findings in *RQ1* that environment predicting is not as important as the other two modules, in terms of supporting self-adaptation.

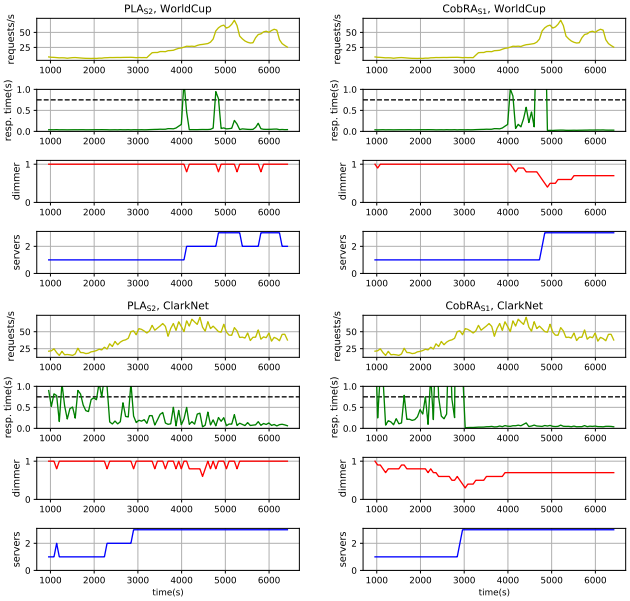

Figure 14: The detailed comparison of PLA_{S3} and $CobRA_{S1}$

Last but not least, Figure 15 compares the detailed temporal features of PLA_{S4} and $CobRA_{S1}$. For the third time, we observe the switch of leading positions on different workloads. However, the difference of the leading advantages here is much larger than that of the other experiments in *RQ2*. On workload WorldCup, PLA_{S4} achieves 3% shorter time that suffers overshoot response time (3% compared to $CobRA_{S1}$'s 6%). On workload ClarkNet, $CobRA_{S1}$ achieves 35% shorter time that suffers overshoot response time (2% compared to PLA_{S4}'s 27%). This dramatic switch of the leading advantage echoes our finding in *RQ1* that uncertainty filtering has a very significant role in the effectiveness of self-adaptation. To some extends, we believe that it is the contribution of the active uncertainty filtering module in PLA_{S3} compensate its performance in the previous comparison between PLA_{S4} and $CobRA_{S1}$.

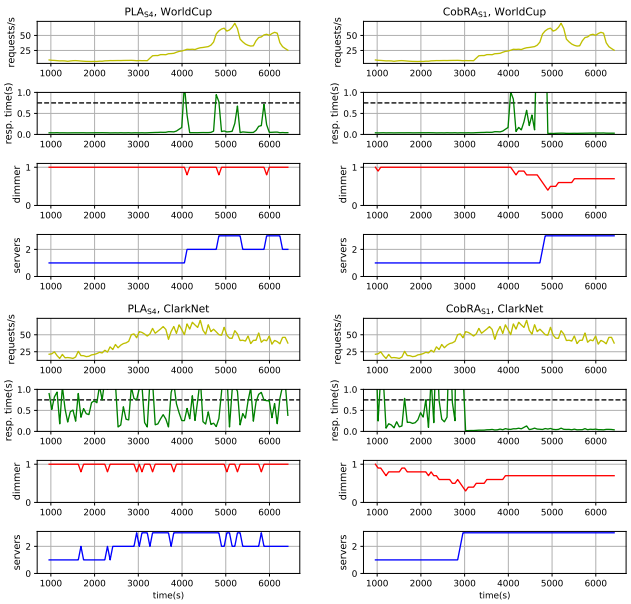

Figure 15: The detailed comparison of PLA_{S4} and $CobRA_{S1}$

In summary, we conclude our answers to the research question *RQ2* as follows. *PLA's accurate LPS-based system model is more effective in supporting self-adaptation, compared with CobRA's linear model. When running in a highly dynamic environment, uncertainty filtering is very important for self-adaptation, which can compensate for the performance of the other two modules to some extends.*

4.3 Take-away Points from the Study

Based on the answers to both *RQ1* and *RQ2*, we summarize three take-away points as follows.

- *Unprecise system modelling can be compensated by uncertainty filtering to some extents.*
- *Environment predicting is less effective than precise system modelling and well uncertainty filtering.*
- *Uncertainty filtering has a larger impact on the self-adaptation's performance than the other two modules in handling frequent environmental changes.*

4.4 Threats to Validation

One major concern on the validity of our empirical conclusions is the selection of evaluation subjects in our evaluation. We only use one subject as the managed system. This might harness the generalization of our conclusions. Nevertheless, a comprehensive evaluation requires a full understanding of suitable managed systems for experimentation. This requirement restricts our choice of possible experimental subjects. We select RUBiS as the managed system since we have spent nearly two years on this system. However, we believe that RUBiS is a quite representative target for self-adaptation, and it has been used in many other pieces of work in the community of self-adaptive systems [17] [16].

Another concern is about the managed system's running environment of . We conducted the experiments by using a network event simulator, SWIM, instead of deploying RUBiS in the real network environment. The simulated environment might not allow

our conclusions to be generalized to the real environment. Nevertheless, a comprehensive evaluation requires the support of suitable environments for experimentation, which should be both observable and controllable. This requirement restricts our choice of possible evaluation subjects. We try to make our experiments realistic by using real-world workloads in the simulator. Consider that we can only afford to run the experiments on two workloads, due to the limitation of time and efforts. We try to make the selected workloads representative enough, in which WorldCup is a gentle workload, and ClarkNet is a fluctuant one.

5 A CONCEPTUAL IMPROVEMENT OF PLA BY SEPARATING UNCERTAINTY

Our empirical study reveals **the very important role of uncertainty filtering in supporting self-adaptation**. In this section, we briefly discuss a conceptual improvement of PLA mechanism by promoting its uncertainty filtering to be a first-class element.

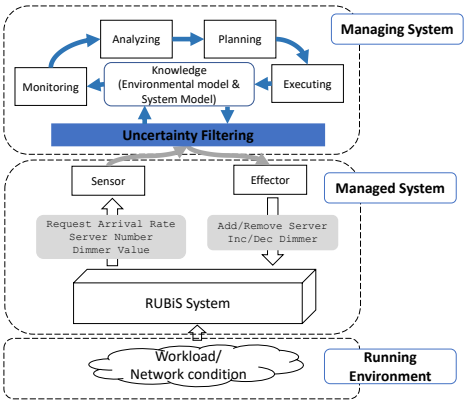

Figure 16: The MAPE-U mechanism

Figure 16 presents the structure of the proposed MAPE-U mechanism. In the traditional MAPE-K mechanism, adaptation decision is made based on the knowledge of the current system state. This mechanism is improved by proactive self-adaptation, such as PLA and CobRA, by combining the aforementioned knowledge with the prediction of the system's future states. Our MAPE-U mechanism further improves PLA and CobRA by using a separated uncertainty filtering module to posterior adjust the knowledge about the prediction.

MAPE-U defines a compensating coefficient model to describe the connection between uncertainty and the knowledge of the system and its surrounding environment. We identify a compensation factor according to the measured values of the system's output, and the model-based hesitated values of the output. The compensation factors are later fed into a Kalman filter procedure to dynamically adjust the estimated values of the system's states.

Take our aforementioned motivation system of RUBiS as an example. Let us denote the compensation coefficient is K. The value of K is determined by the uncertainty of system modelling and environmental prediction. Here we consider such uncertainty as to the error of the estimated response time and the predicted response time. We represent the predicted value of response time as y, and the measured responds time $LPS(x)$, in which we denote the system state as x. The relationship between K and the system model can be described in the following equation.

$$y = LPS(x) \cdot K, \tag{5}$$

According to this relationship, we use the following equation to describe the state-space model of the concerned self-adaptation mechanism:

$$x_k = A \cdot x_{k-1} + B \cdot u_{k-1} + w_{k-1} y_k = C \cdot x_k + v_k \tag{6}$$

where x_i describes the system state at the time of i, u_i describes the system's adaptation action (i.e., the executed value of "dimmer") at the time of i, and y_i describes the system's output value (i.e., the response time) at the time of i. A, B, and C describes the control-based model of the MAPE-K adaptation, in which A describes the latency property of the adaptation, B describes the control property of the adaptation, and C describes the correlation between the system's internal states and its output. We use w_i to describe the noise of the adaptation, which is subject to a gaussian distribution of $N(0, Q)$. We use v_i to describe the noise of the observation, which is also subject to a gaussian distribution of $N(0, R)$. We assume that the covariance of noise w_i and v_i can be identified base through monitoring the system's execution traces.

Based on this formal description, MAPE-U uses Kalman filter to dynamically estimate the compensation coefficient at runtime and adjust its knowledge of the system and surrounding environment. Basically, a MAPE-U mechanism considers two types of uncertainty, internal uncertainty that affects the internal states of the system, and external uncertainty that affects the external observation of the system's output. According to our state-space model, MAPE-U needs to identify the covariance of internal uncertainty (i.e., w) and external uncertainty (i.e., v) in advance.

Notice that currently, the MAPE-U mechanism is a purely-conceptual model based on our empirical findings. We plan to implement it on RUBiS, and investigate its effectiveness in the future.

6 RELATED WORK

Software adaptation has been attracted to lots of research efforts from the community of software engineering. Proactive self-adaptation further improves the traditional reactive self-adaptation, and has become one of the research hotspots in recent years. Basically, the existing self-adaptation mechanism can be classified into architecture-based ones and control-based ones. The former depends on an accurate discrete model of the managed system, while the latter simple use a numeric model of the managed system's input and output values.

PLA, which is proposed by Gabriel et al. [16], is a representative architecture-based mechanism. As discussed previously, PLA uses a formal model of the managed system. The adaptation action in PLA is decided based on a non-deterministic model checker which tries to maximize the adaptation utility. Gabriel et al. [17] further improved PLA by using Alloy models to building a more accurate model of the managed system. Beside these PLA-based approaches, Hielscher et al. [10] proposed an active adaptive framework that uses online testing to detect problems before they occur in actual transactions, and triggers adaptation when the current execution

fails in a test. Wang et al. [23] used QoS degradation online prediction to trigger preventive adaptation before SLA violations.

Control-based mechanism emerges due to their simpler assumptions on the managed system and its running environment. Garlan et al. [9] proposed Rainbow adaptive conceptual framework, which is a Software adaptation framework based on feedback structure. Rainbow focuses on the modelling of the dynamic relationship between requirements and possible adaptations. Konstantinos et al. [1] proposed a control theory based self-adaptation framework, which is referred to as CobRA. CobRA proposed using control theory to design a controller that optimizes the cost-function-specified adaptation goal. Maggio et al. [15] proposed an automated approach to merge multiple adaptation goals within a single controller.

There are also existing empirical studies on the different performance of different self-adaptation mechanism. Shevtsov et al. [21] compared a control-based mechanism, SimCA, with an architecture-based mechanism, ActivFORMS. They reported that the architecture-based mechanism is better for systems that require low settling time while the control-based method achieves better results in the presence of runtime fluctuation. Gabriel et al. [18] compared PLA and CobRA from the aspects of development cost and run-time performance. They reported that both can achieve the same good adaptation control effect. Different from these pieces of work, we perform an in-depth comparison between PLA and CobRA's different design and implementation choices in their three sub-modules and find the difference in their performance in different environmental workloads.

7 CONCLUSION AND FUTURE WORK

In this paper, we conduct an in-depth comparison between two proactive self-adaptation approaches, PLA and CobRa, in terms of their system modelling, environment predicting, and uncertainty filtering. We performed an ablation study on the three modules of PLA and compared their performance with CobRA. Our empirical study reveals **the very important role of uncertainty filtering** in supporting self-adaptation, as well as **the huge impact of a fluctuant environment on a self-adaptation mechanism**. Based on this observation, we simply discuss a conceptual self-adaptation mechanism, MAPE-U, which separates the logic of uncertainty filtering with the logic of adaptation-decision making. We plan to implement MAPE-U mechanism, and experimentally validate its effectiveness as our future work.

8 ACKNOWLEDGEMENTS

This work was supported in part by the National Natural Science Foundation of China (#61932021, #61902173, #62025202) and the Natural Science Foundation of Jiangsu Province (#BK20190299).

REFERENCES

[1] K. Angelopoulos, A. V. Papadopoulos, V. E. Silva Souza, and J. Mylopoulos. 2016. Model predictive control for software systems with CobRA. In *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing systems*. ACM, 35–46.

[2] M. Arlitt and T. Jin. 2000. A workload characterization study of the 1998 world cup web site. *IEEE Network* 14, 3 (2000), 30–37.

[3] M. F. Arlitt and C. L Williamson. 1996. Web server workload characterization: The search for invariants. *ACM SIGMETRICS Performance Evaluation Review* 24, 1 (1996), 126–137.

[4] R. Calinescu, D. Weyns, S. Gerasimou, M. U. Iftikhar, I. Habli, and T. Kelly. 2017. Engineering trustworthy self-adaptive software with dynamic assurance cases. *IEEE Transactions on Software Engineering* 44, 11 (2017), 1039–1069.

[5] E. Cecchet. 2009. Rubis: Rice university bidding system. http://rubis.ow2.org.

[6] I. Chun, J. Kim, W. T. Kim, and E. Lee. 2011. Self-Managed System Development Method for Cyber-Physical Systems. In *Control and Automation, and Energy System Engineering*. Springer, 191–194.

[7] P. A. Dinda. 2006. Design, implementation, and performance of an extensible toolkit for resource prediction in distributed systems. *IEEE Transactions on Parallel and Distributed Systems* 17, 2 (2006), 160–173.

[8] A. Filieri, M. Maggio, K. Angelopoulos, N. d'Ippolito, I. Gerostathopoulos, A. B. Hempel, H. Hoffmann, P. Jamshidi, E. Kalyvianaki, C. Klein, et al. 2015. Software engineering meets control theory. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 71–82.

[9] D. Garlan, S. W. Cheng, A. C. Huang, B. Schmerl, and P. Steenkiste. 2004. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer* 37, 10 (2004), 46–54.

[10] J. Hielscher, R. Kazhamiakin, A. Metzger, and M. Pistore. 2008. A framework for proactive self-adaptation of service-based applications based on online testing. In *European Conference on a Service-Based Internet*. Springer, 122–133.

[11] M. A. Islam, S. Ren, A. H. Mahmud, and G. Quan. 2015. Online energy budgeting for cost minimization in virtualized data center. *IEEE Transactions on Services Computing* 9, 3 (2015), 421–432.

[12] J. O. Kephart and D. M. Chess. 2003. The vision of autonomic computing. *Computer* 36, 1 (2003), 41–50.

[13] C. Klein, M. Maggio, K. E. Årzén, and F. Hernández-Rodriguez. 2014. Brownout: Building more robust cloud applications. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 700–711.

[14] L. Määttä, J. Suhonen, T. Laukkarinen, T. D. Hämäläinen, and Marko Hännikäinen. 2010. Program image dissemination protocol for low-energy multihop wireless sensor networks. In *2010 International Symposium on System on Chip*. IEEE, 133–138.

[15] M. Maggio, A. V. Papadopoulos, A. Filieri, and H. Hoffmann. 2017. Automated control of multiple software goals using multiple actuators. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 373–384.

[16] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl. 2015. Proactive self-adaptation under uncertainty: a probabilistic model checking approach. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 1–12.

[17] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl. 2016. Efficient decision-making under uncertainty for proactive self-adaptation. In *2016 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 147–156.

[18] G. A. Moreno, A. V. Papadopoulos, K. Angelopoulos, J. Cámara, and B. Schmerl. 2017. Comparing model-based predictive approaches to self-adaptation: CobRA and PLA. In *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 42–53.

[19] V. Nallur and R. Bahsoon. 2012. A decentralized self-adaptation mechanism for service-based applications in the cloud. *IEEE Transactions on Software Engineering* 39, 5 (2012), 591–612.

[20] S. J. Qin and T. A. Badgwell. 2003. A survey of industrial model predictive control technology. *Control Engineering Practice* 11, 7 (2003), 733–764.

[21] S. Shevtsov, M. U. Iftikhar, and D. Weyns. 2015. 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*. 1–8.

[22] E. D. Sontag. 2013. *Mathematical control theory: deterministic finite dimensional systems*. Vol. 6. Springer Science & Business Media.

[23] C. Wang and J. L. Pazat. 2012. A two-phase online prediction approach for accurate and timely adaptation decision. In *2012 IEEE Ninth International Conference on Services Computing*. IEEE, 218–225.

[24] J. Zhang and B. Zwart. 2008. Steady state approximations of limited processor sharing queues in heavy traffic. *Queueing Systems* 60, 3-4 (2008), 227–246.