Recomparison of Two Proactive Self-adaptation Methods: 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

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

ABSTRACT

Software-intensive systems face the challenge of uncertainties at runtime, and need to adapt themselves to keep satisfying their requirements. PLA and CobRA are two representative approaches to the construction of proactive self-adaption software system. The former is based on software architecture models and stochastic analysis, while the latter directly applies model predictive control (MPC). Unmodeled uncertainties, such as modeling error, has severe impact on decision making of PLA. Gabriel et. al proposed an implicit feedback to reduce the influence of model error on decision making. However, this feedback destroys the physical meaning of the model. In this paper, we combine PLA with an explicit prediction estimation part to reduce the impact of modeling error. Gabriel et al. have compared the two approaches in aspects of development cost, run-time performance and the efforts to achieve desired adaptation effects. They claimed that both of them can achieve the similar satisfying adaptation results. Based on their work, considering the effort of feedback on PLA, we discuss the merits of PLA and CobRA and where improvements can be made from three aspects, including environment handling, model building and asymmetrical latency handling. It turned out that feedback played a crucial role in both approaches and the two methods are complementary under different situations. We also observed that compared to PLA, MPC could not handle the asymmetrical latency directly.

CCS CONCEPTS

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

KEYWORDS

self-adaptation, PLA, CobRA, model predictive control

ACM Reference Format:
Jingxin Fan, Yanxiang Tong, and Xiaoxing Ma. 2019. Recomparison of Two Proactive Self-adaptation Methods: PLA and CobRA. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 9 pages.

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

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

1 INTRODUCTION

Software-intensive systems suffer uncertainties at runtime, such as fluctuant running environment, changing requirements and unexpected operating conditions which often reduces the quality of service delivery [5]. Self-adaptation is required to eliminate or mitigate the adverse effects of uncertainties. Software self-adaptation is the ability to adapt to uncertainties for continously delivering high-quality services through the dynamic reconfiguration of its parameters or architecture [2]. A self-adaptation system is composef of managed system, realizing the funtion of the system, and managering system, which aim to make adaptation decisions. Manageing system is often implemented as MAPE-K loop [8] which consists monitor, analyze, plan, and execute modules sharing a whole knowledge, shown in Figure 1. Monitor is used to collect information about the uncertainty need to be solved, such as the changes of the environment and requirements. Based on the acquired changes, adaptation objectives and adaptation tactics, the Plan module draws conclusion on how to adjust the software system to adapt to such changes (for example, replace component A with component B). Finally, the Execute module applies the evaluation results to the target software system to complete the self-adaptation process.

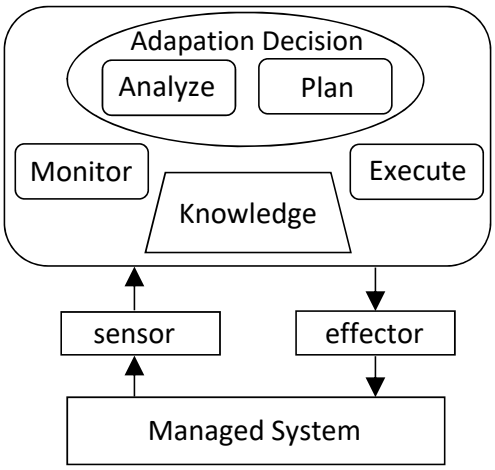

Figure 1: MAPE-K architecture

The early approaches of software self-adaptation are reactive. They calculate adaptation tactics only based on the captured changes

but do not consider changes in the future and the effects of adaptation tactics are regarded to be instant. The obvious disadvantage of those approaches is that they can not handle adaptation tactics with latency. Correspondingly, the later proposed proactive self-adaptation approaches predict future changes to make an adaptation strategy ahead of time and can handle tactic latency, the period of time between when the tactic is deployed and when the effect is shown.

To implement proactive software self-adaptation, two main approaches, PLA [13] and CobRA [1] are proposed. Both of them adopt the ideas from model predictive control (MPC) [16]. MPC is based on the prediction in adaptation horizon and on the optimization of a cost function to make adaptation decisons and is well suited for multi-objective problems. PLA is an architecture-based feed-forward method. PLA models the managed system and environment precisely and use models to describe uncertainty while CobRA is a feedback method based on control theory. CobRA regards the system as a linear model and treats the changing environment as disturbance, so that it does not model and predict the environment.

PLA and CobRA are quite different both in approach and implementation. Gabriel et al. have compared the two approaches in the light of development cost and run-time performance and shown their similar self-adaptation results on a benchmark [14]. They show that both can achieve the same good adaptation control effect. But they did not consider the impact of the unmodeled uncertainties on PLA and did not analyze why PLA and CobRA can achieve the similar self-adaptation results and investigate the capabilities of their different implementations for handling uncertainties. Therefore, based on their work, we first try to give an explanation for the similar self-adapatation results and enumerate the advantages and disadvantages of the two in different scenarios are put forward according to their differences.

Compared to CobRA, PLA is an open-loop approach that the adaptation tacitc does not affect the decision-making at the next period. One of the drawbacks of the feed-forward approach is that it can only address uncertainties that have been modeled. As an architecture-based modeling method, PLA relies on the accuracy of the system model and the environment model for adaptive decision making. Without feedback, modeling errors will inevitably have an impact on adaptive decisions. When there is a large error between the model and the actual system, the correct decision cannot be made. In Moreno2017, it is proposed that Kalman Filter is used to dynamically adjust the servicetime parameter of RUBIS system according to the differences between the predicted and measured values of the response time. This adjustment destroys the physical meaning of the model because the serviceTime should not be adjusted with the change of workload.

Therefore, we add a parameter estimation part to PLA, which can not only solve the influence of PLA model error on decision making, but also ensure that the physical meaning of PLA model is not destroyed. We define the modeling error as the uncertainty caused by modeling. Then we explicitly define a parameter to describe this uncertainty. Based on the difference between the output value of the model and the measured value of the actual system, we dynamically estimate the value of this parameter using Kalman Filter, so as to constantly match the model with the actual system reducing the influence of modeling error on decision making. Using Kalman Filter, the output value can be used to estimate the prediction.

Considering the effect of prediction estimating on PLA, we depict the differences between the two proactive approaches from three aspects, namely 1) environment prediction, 2) system modeling, and 3) asymmetric latency. We follow the same adaptation senario, a simulated system of RUBIS, with [14] to verify our conjecture and theoretically analyze the effects of these differences on self-adaptation decision. We think that the advantage of the PLA approach over CobRA is that it has a better understanding of the system and can solve problems such as asymmetric latency. Moreover, with the feedback of PLA, the need of the precision modeling is reduced, and the modeling error can be tolerated to some extent.

In the remainder of the paper, Section II presents the adaptation scenario we used and provides an overview of CobRA and PLA both on design and implementation. Next, Section III describes the need for feedback of PLA and gives an explanations for their similarities. Section IV describes our new model for PLA to handle the uncertainty in model building. Section V discusses three different aspects of CobRA and PLA considering feedback in PLA. Finally, Section VI presents our conclusions and the future work.

2 ADAPTATION SENARIO

The same as [14], we carry out our comparison on the RUBiS system [3], an open-source auction site prototype modeled after eBay.com which is widely used for evaluating software self-adaptation and cloud computing [6, 16]. As shown in Figure 2, RUBiS consists of a load-balancer, a web server tier and a database tier. Clients use browsers to send requests to the system. Load balancer distributes requests among servers following a round-robin policy. Then, servers access the database to obtain the data to render the page which includes mandatory content and optional content. Providing optional content such as recommendations of similar auction items for users can achieve more revenue but need more response time and an intolerable response time may lead to high penalty. Therefore, a brownout RUBiS [9] is proposed which gradually downgrade the possibility of optional content for user requests, called dimmer, for handling high workload. In order to keep and reproduce the same experimental conditions for the two approaches, we run our experiments on a simulated system of brownout RUBiS, called SWIM.

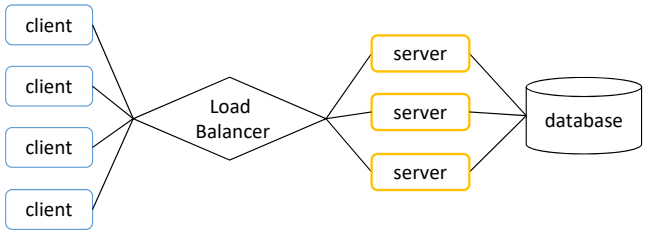

Figure 2: RUBiS architecture

The self-adapation goal of RUBiS is to capture the maximum revenue of user requests with the minimum server cost in spite of the uncertainties. There are three non-functional requirements concerned with our self-adaptation goal: the first one is keeping

the response time, denoted by r, below the threshold T; the second one is the target system shall provide high quality of content, namely maximize dimmer value d; the third one is that the target system shall operate under low cost, namely minimize the number of servers s. Then, the adaptation goal can be formally expressed as a utility function of these three requirements as follows:

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

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

$$U_C = \tau \cdot c \cdot (s^* - s) \quad (2)$$

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

$$U_R = \tau \cdot \alpha \cdot (d \cdot R_O + (1 - d) \cdot R_M) \quad (3)$$

where τ is the length of the interval, α is the average request rate in this interval, also called workload, and d is dimmer value. R_M and R_O are the rewards for serving a request with mandatory and optional content respectively. We have $R_O > R_M > 0$.

For RUBiS, there are two known kinds of uncertianties, namely 1) workload and 2) latency of booting a server which can be measured aftertime and it provides two types of adaptation tactics to adjust itself to deal with these uncertainties:

- `Add/remove a server`: The two tactics are used to change the number of servers. There is a delay in adding a server whereas the delay of removing a server can be ignored.
- `Increase/decrease dimmer value`: The two tactics are used to change the value of dimmer which indicates the proportion of a request including optional content ($d \in [0, 1]$) and the effect of them are regarded to be instant.

3 TWO PROACTIVE SELF-ADAPTAION APPROACHES: PLA AND COBRA

Both PLA and CobRA adopt the ideas of model predcitive control (MPC) to implement software self-adaptation but use different ways, shown in Figure 3. Next, we will describe PLA and CobRA in terms of design details and implementations.

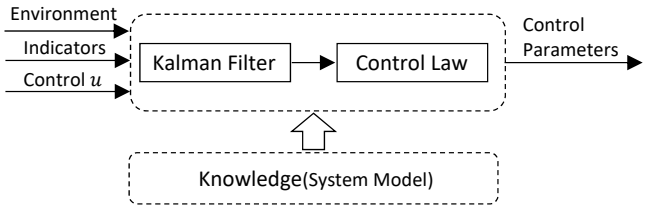

Figure 3: Proactive self-adaptation framework from the perspective of control theory

3.1 PLA

The framework of PLA is shown in Figure **??**, it uses DTMCs to model the probabilistic behavior of the environment and the managed system including tactic delay, leave adaptation decision underspecified through nondeterminism and then use the probabilistic model checker to find the nondeterministic choices that maximize the accumulated utility over the horizon [13].

For each adaptation period τ, PLA uses an autoregresive (AR) time series predictor, such as RPS toolkit [4], to predict the distribution of each environment variable and applies the Extended Pearson-Tukey (EP-T) [7] three point approximation to construct probability trees for next environment prediction. As shown in Figure 4, the root e_0 is the current value of a environment variable e and e_i is the next ith predicted distribution of this environment variable. The three point values are the 5th, 50th and 95th percentiles of the predicted distribution with transition probabilities 0.185, 0.630 and 0.185 respectively. It should be noted that there are often large deviations between the predicted values of the environment and the real ones because it is hard to satisfy the assumption that the changes of environment should be predictable.

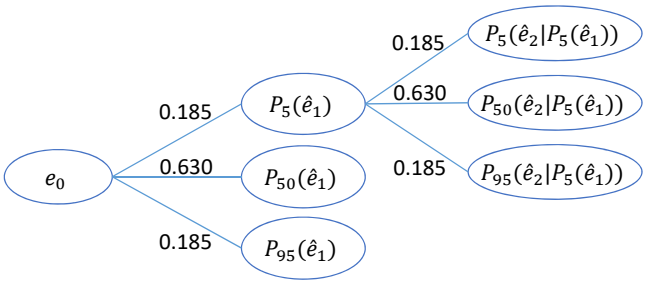

Figure 4: Probability tree of future environment

The managed system accompanied with adaptation tactics is modeled as a Markov decision process (MDP) [15] which uses state transitions to express all possible execution of adaptation tactics for handling the predicted environment. We can use a probabilistic model checker [13], such as PRISM [10] to exhaust state transitions of adaptation tactics over the horizon and find a path of state transitions (a strategy) to maximize the accumulated value of the utility function defined above. PLA accurately models the relationship between measurable requirement-related variables and adaptation parameters. For example, It uses LPS queuing theory model [18] to calculate the response time under specified configurations of server and dimmer. Obviously, the adaptation results of PLA is highly relied on the prediction values, thus it needs plenty of effort to build an accurate model of the environment and the target system in design time.

3.2 CobRA

The framework of PLA is shown in Figure **??**, it implements the self-adaptive system directly from control theory and the adaptation decision is aimed to find the right balance (equilibrium) between the conflicting requirements by maximizing the utility-based cost funtion defined as:

$$J_k = \sum_{i=1}^{H} [y^o_{k+i} - \overline{y}_{k+i}]^T Q_i [y^o_{k+i} - \overline{y}_{k+i}] + [\Delta u_{k+i-1}]^T P_i [\Delta u_{k+i-1}] \quad (4)$$

where y represents the set of measurable requirement-related variables called *indicators* and u represents the set of adaptation parameters called *control parameters*. y^o_{k+i} and $\overline{y}_{k+i}$ are setpoints (goals) and predicted values of indicators at next i-th adaptation period respectively. Q_i and P_i are symmetric positive semi-definite wetghting matrics. Q_i means that when not all the goals can be achieved, the controller will prefer the satisfaction of the goals with high weights. P_i means that the controller want to change the control parameters frequently with smaller weight. For RUBiS, the indictors is the average response time r and the *control parameters* are the number of servers s and the dimmer value d. As same to PLA, The control interval of CobRA is τ and at the start of each interval, a sequence values of control parameters over the horizon are calculated by controller.

CobRA abstracts the relationship between indictors and control parameters as a simple discrete-time linear model as follows:

$$\begin{cases} x_{k+1} = A \cdot x_k + B \cdot u_k \\ y_k = C \cdot x_k + D \cdot a_k \end{cases} \tag{5}$$

where x is the set of system states which is often a immeasurable variables but links control parameters to indicators [11] and a represents measurable environment variables, such as workload, which cannot be tuned can be used as a feedforward signal and k represents the k-th adaptation loop. Unlike PLA, CobRA does not predict the changes of future environment but treats the changes of environment as disturbance to the system instead. In our scenario, the known uncertianty of environment is arrival rate of user requests.

Considering the optimization variables of J_k, namely Δu_k, the nominal model (Eqn.5) is usually transformed in the augmented velocity form (Eqn.6), where the predicted indicators is unchanged but now expressed with respect to the state variations, denoted by Δx_k.

$$\begin{aligned} \overbrace{\begin{bmatrix} \Delta x_{k+1} \\ y_k \end{bmatrix}}^{\widetilde{x}_{k+1}} &= \overbrace{\begin{bmatrix} A & 0_{n\times p} \\ C & I_{p\times p} \end{bmatrix}}^{\widetilde{A}} \overbrace{\begin{bmatrix} \Delta x_k \\ y_{k-1} \end{bmatrix}}^{\widetilde{x}_k} + \overbrace{\begin{bmatrix} B \\ 0_{p\times m} \end{bmatrix}}^{\widetilde{B}} \Delta u_k \\ y_k &= \overbrace{\begin{bmatrix} C & I_{p\times p} \end{bmatrix}}^{\widetilde{C}} \overbrace{\begin{bmatrix} \Delta x_k \\ y_{k-1} \end{bmatrix}}^{\widetilde{x}_k} \end{aligned} \tag{6}$$

Above all, the self-adaptation decision of CobRA is to solve the optimization problem at start of each period definited as follows:

$$\begin{aligned} minimize_{\Delta u_{k+i-1}} \quad & J_k \\ subject\ to \quad & u_{min} \le u_{k+i-1} \le u_{max}, \\ & \Delta u_{min} \le \Delta u_{k+i-1} \le \Delta u_{max}, \\ & \tilde{x}_{k+i} = \tilde{A} \cdot \tilde{x}_{k+i-1} + \tilde{B} \cdot \Delta u_{k+i-1}, \\ & \overline{y}_{k+i-1} = \tilde{C} \cdot \tilde{x}_{k+i-1}, \\ & i = 1, ..., H. \end{aligned} \tag{7}$$

where Δu_{min} and Δu_{max} are the constraints for changing control parameters while u_{min} and u_{max} are the saturation values of control parameters. CobRA makes adaptation decisions to keep indicators as close to their setpoints as possible while satisfying those constraints. The solution of this convex quadratic programming problem is a sequence of future control variables' variations, denoted by $\Delta u^*_k, \Delta u^*_{k+1}, ..., \Delta u^*_{k+L-1}$ and only the first one is adopted to calculate the new control variables as $u_k = u_{k-1} + \Delta u^*_k$.

4 HANDELING UNMODELED UNCERTAINTIES IN PLA

In this section, we analyze an unmodeled uncertainty in PLA, namely the impact of modeling errors on decision making, and propose way to solve this impact with parameter estimation using Kalman Filter.

4.1 Modeling Error in PLA

As described above, PLA is an open-loop method which the current adaptation result does not affect the decision-making at the next period. Open-loop methods can only handle the modeled uncertainties. PLA hopes to carry out detailed modeling of the system and use the model to describe the uncertainties that needed to be addressed. However, there are still unmodeled uncertainties in PLA. First, it is inevitable that there will be deviations in environment prediction. Second, unlike physical systems, software systems are usually variable and nonlinear and it is difficult to model the software system in detail. We define system modeling errors and environment modeling errors as the uncertainty caused by modeling.

PLA uses LPS queuing model [18] to simulate system's service process and calculates the average response time of requests. Using LPS queuing theory, we can obtain the estimation of responsetime of the system under the current state of the environment, for example, the workload, and the state of the system, such as the number of servers, the average servicetime for jobs and the dimmer value. For this kind of queuing system, when the arrival rate of requests is higher than the service rate that the system can handle, the length of the queue will be infinitely long, while conversely, the queued system will gradually reach a stable state from the initial transition state.

However, while the workload constantly fluctuating, there is a heap for queuing time when the request arrives in the service queue, because the request is received within the server in the form of time slices. During this stacking, the actual response time value will be higher than the value estimated using the LPS queueing model that lead to the error of the model prediction. This error can lead to a mistake in decision making. Although LPS model is a pretty accurate description of limited process sharing system with round-roubin strategy, the accuracy of the model is based on a steady state. In reality, when the system is running, it is difficult to keep stable state resulted from workload fluctuation. There must be deviations between the predicted value based on queuing model and the actual measured response time. Because RUBiS is an exponential system, when measured response time is around the threshold T, model deviations will be highly harmful to PLA's self-adaptation results. When the system is faced with such a fluctuating load, it is possible that the actual response time has exceeded the threshold and the calculated value of the model is lower than the threshold. At this point, the Plan part does not make any decisions based on the Utility function. However, the real responsetime of the actual system has exceeded the threshold, so the server needs to be increased or dimmer value needs to be reduced to maximize

the revenue. Therefore, the modeling error will affect the decision, and then lead to the decline of system revenue. If PLA is just a feed-forward method, that is, the output of the model will not affect the model, the error between the model and the actual system will never be eliminated.

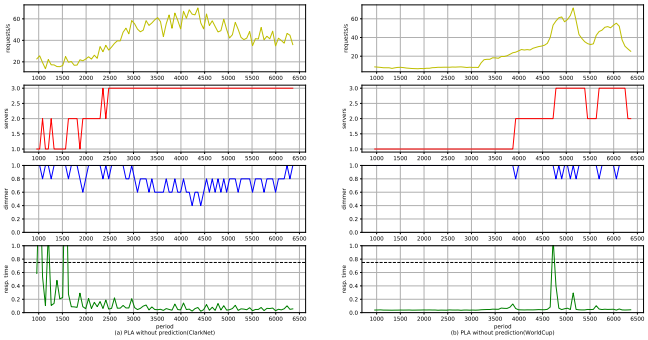

Figure 5: PLA without prediction for environment

We have carried out experiments to illustrate this problem. As shown in the Figure reffig:nopre, there is an obvious difference between the calculated value of the queuing theory model and the actual measured value. For n percent of the time, the value of response time calculated by the model does not exceed the threshold, but the value of the actual system exceeded, which requires decision making accordingly.

One way to solve the error of the model is to build a more accurate model as far as possible, but this is often difficult to do, and requires a high cost. Another approach is to use feedback to reduce the modeling error. If we add the prediction estimating part to PLA, such as using Kalman Filtering, the model can gradually be more consistent with the actual system according to the difference between the output value of the model and the actual value of the system.

4.2 Prediction Estimating for PLA

To handel the error of modeling, [12] proposed that Kalman Filter can be used to realize implicit feedback by dynamically estimating the value of *servicerate* at run time, according to the difference between the response time calculated by the queueing model and the value measured by the actual system. In this way, the impact of modeling errors can be reduced. The framework of PLA with implicit feedback is shown in Figure **??**. However, the *servicetime* value represents the average time for a request to be serviced and is determined by the request itself. In the queueing model, this value is usually random distributed or poisson distributed and can not be dynamically adjusted based on the difference between the model and the actual system. If we use Kalman Filtering to adjust this value in order to achieve prediction estimating, the physical meaning of the LPS model will be destroyed.

Therefore, based on their work, we design a framework of two layers to describe the uncertainty caused by modeling and reduce the impact of it on decision making. The first layer is the original PLA model, while the second layer is a prediction estimation using Kalman Filter. We dynamiclly estimate the prediction model to

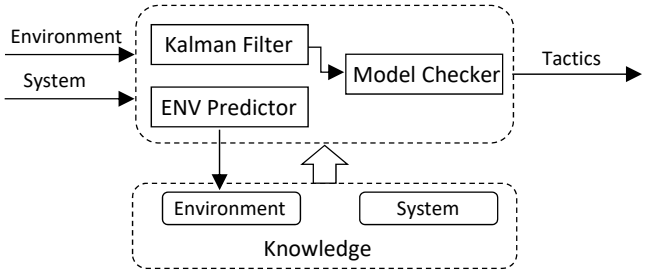

Figure 6: PLA framework with implicit feedback

reduce the influence of modeling error. The framework is shown in Figure **??**.

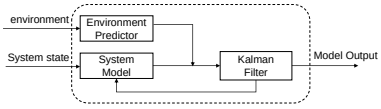

Figure 7: PLA framework with explicit feedback

We use X to represent the state of uncertainty caused by the modeling process of PLA, namely the modeling error. As shown in the figure, we add a part of explicit feedback on the basis of PLA model. According to the predicted and measured value of response time, this state quantity is dynamically adjusted to reducing the error of the model. Since this uncertainty arises with the modeling, the relationship between the state quantity X and the system model should be multiplicative. We design a simple model, as shown in the formula,

$$R = LPS(c, e) * X, \tag{8}$$

where R represents the predicted value of response time, and $LPS(c, e)$ represents the value of response time calculated by current system state c and environment state e according to LPS model. Then we dynamically use Kalman Filter to estimate the state X to reduce the impact of the uncertainty on decision making. We can set up the equation of state for this system

$$Xk = Ax_k - 1 + Bu_k - 1 + w_k - 1, \tag{9}$$

Where, $w_k - 1$ is the noise subject to mean value of 0, covariance of Q is gaussian distribution, that is, $w_k - 1$ N(0,Q), the observation equation is

$$z_k = Hx_k + v_k, \tag{10}$$

v_k is the observation noise subject to mean value of 0 and covariance of R. v_k $N(0.r)$, where, the observed quantity is the value of

response time, and the state quantity is the parameter X defined to describe the uncertainty. Then, at each decision, Kalman Filtering is used to adjust X according to the difference between the calculated amount and the measured amount based on the response time model. We implemented the new model on the SWIM simulation system and experiment on workload, shown in Figure reffig:new. The experimental results show that our model can also achieve better control effect without destroying the physical meaning of LPS model.

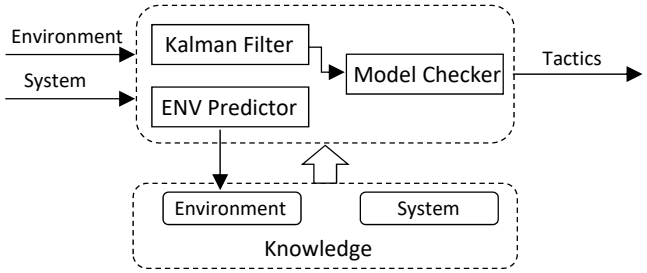

Figure 8: PLA framework with explicit feedback

5 DISCUSSION

While PLA and CobRA can achieve similar adaptation effect, there are some differences between them, which are mainly described in table 1.

One of the significant differences between CobRA and PLA is that the models are different. PLA is very sophisticated in modeling the system and the environment, so high cost is incurred in modeling. CobRA use Kalman Filter to realize feedback to tolerate changes in the environment and errors in the model. However, when we added feedback regulation to the PLA, we wanted to see if the PLA could do the same thing with CobRA, without the need for detailed modeling of the system and environment to achieve better control effects. We mainly investigate four significant differences, namely 1) environment handling; 2) system modeling; 3) asymmetric tactic delay handling and 4) constraints of regular terms and explore how these differences affect the adaptation decisions of PLA and CobRA.

5.1 Environment Handling

While implementing the self-adaptation of managed software system, PLA and CobRA handle the changes of environment differently. PLA assumes the changing envrionment is predictable and uses current and historical measurements to construct the probabilistc tree of 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. Whereas CobRA regards the assumed slowly changing environment as disturbances which is handled by closed-loop feedback control. Thus, it does not predict future environment but uses its measurements as a feedforward signal to improve the prediction of current indicators which is used in Kalman Filter to obtain the true states of current system.

When the environment changes regularly and easy to predict accurately, decision-making with future environment prediction is theoretically better than no prediction like CobRA beacause the effect of future envrionment changes on managed system can be considered ahead of time. However, the accurate environment prediction is hard to obtained and there is often deviation of system model. Practically, PLA uses feedback mechanism to handle model deviation. In this section, we want to explore the effect of PLA's environment prediction on adaptation decision-making.

We design a variant of PLA, denoted by *npPLA*, which does not predict the future workload but just use feedback mechanism to cope with changing environment like CobRA. We compared the adaptation results of PLA and npPLA under the two workloads. Figure 11 shows the adaptation process of PLA without environment prediction and Figure 12 shows the utility of npPLA compared with PLA. As we can see, the utility of npPLA is 11463 for WorldCup and 8005 for ClarkNet, which is similar to the utility of PLA under such workloads. The comparison result shows that the environment prediction contributes little to adaptation decision because the feedback mechanism of PLA can almost handle the slowly changing envrionment similar to CobRA and the deviation of environment prediction also needs to be compensated by feedback mechanism.

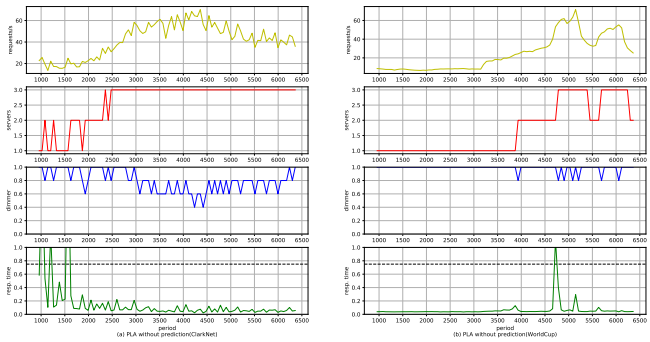

Figure 9: PLA without prediction for environment

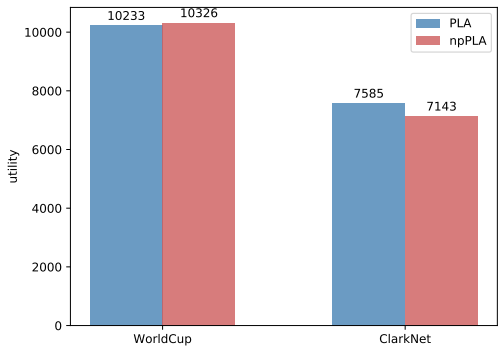

Figure 10: Utility of PLA and npPLA

Table 1: Differences between PLA and CobRA

Features	PLA	CobRA
Method	Architecture-based	Requirement-based
Model for Environment	DTMC model	Feedforward signal
Feedback	Open-loop	Closed-loop
Control	tactics	Control parameter values
Discrete or Continuous	Discrete	Continuous
Unsymmetric Latency	Model the unsymmetric latency	Can not model the unsymmetric latency
Total Model	Non-linear MDP Model	Linear Dynamic Model
Optimization	Utility function	System goals following boolean AND/OR semantics

5.2 Linear Model & Exponential Model

One of the most significant differences between the PLA and CobRA is system modeling which quantize the relationship of indictors and adaptation parameters. PLA regards the managed system as white-box and pays much effots on precise modeling. For RUBiS, it use LPS queuing model [18] to predict the average response time under specified servers and dimmer. CobRA assumes that the managed system is in vicinity of a equilibrum point despite of uncertainties [17]. Therefore, it regards the managed system as black-box and identify system model as a linear model using sampling data in vicinity of its equilibrum point. For systems like RUBiS that are exponential and sensitive to fluctuation of workload, linear model may makes poor self-adaptation. However, the precise system model of RUBiS is a steady state approximation of LPS queue in heavy traffic [18] and there are significant deviations between predicted indictors and their transient measurements which is relied on feedback mechanism to offset. In this section, we want to discuss the contributions of PLA's precise model to a good self-adaptation.

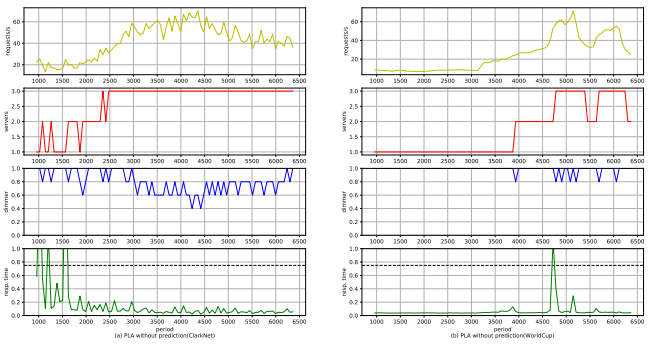

Figure 11: PLA without prediction for environment

First, in order to discuss the effectiveness of precise model, we assume that we can construct a highly accurate model of RUBiS system. Although LPS model is pretty accurate than simple linear model, We compare its control effects with LPS model and As shown in Figure [**?**], if the prediction value of precise model are consistent with the measurements, precise modeling can significant improve the self-adaptation performance.

Then we design a variant PLA named *linearPLA* to discuss the necessity of precise model. The same as CobRA, linearPLA uses

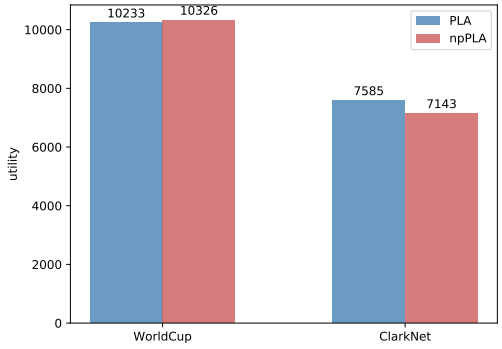

Figure 12: Utility of PLA and npPLA

a time-invariant liear model to describe the system instead of exponential LPS model. Figure **??** shows its adaptation process with the same two workloads and Figure **??** shows the utilitiey compared with PLA. Although linear model is more inaccurate than LPS model, they can get about the same adaptation effects because of the large deviation of precise model prediction and the effect of feedback. In conclusion, if complicate and precise system models also have signficant model uncertainties, simple linear models with good designed feedback mechanisms is a good choice to implementing self-adaptation as they need lower efforts on modeling.

5.3 Asymmetrical Latency

In RUBiS system, there is a certain latency when adding a server, but removing a server can be regarded as no delay. That means there is an asymmetrical latency with the control parameter s.

PLA models asymmetrical latency of each tactics into the state transistion while CobRA does not take asymmetrical latency into account. In control engineering, it always models the control actions corresponding to a control parameter is instant or have the symmetrical latency. Therefore, CobRA forces to model the tactic delay as symmetric. For example, removing server is also considered to have the same latency distribution of booting a server. Because of CobRA's receding horizon, when the workload is decreasing and there is no need to use more servers in the horizon, considering the latency of removing server, CobRA will make the decision to reduce the server in advance and increase dimmer value to compensate the latency which may reduce the adapation performance.

In order to discuss the influence of asymmetrical latency to CobRA's adaptation results, we change the latency of adding a server ranging from 30s to 180s. Figure **??** shows the control result and Figure **??** shows their utilities. When latency increases to 180s, the utility of CobRA has a significant decline. We transform RUBiS system to have symmetrical latency of executing server numbers.

We assume that the linear model of the system is obtained through system identification:

$$r = k_1 * s + k_2 * d + k_3 * a \tag{11}$$

To translate this into a state space model, we use the state matrix A to reflect the latency of adding a server. The matrixs in the state space model of (4) are:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & k_1 & k_2 \end{bmatrix}, D = \begin{bmatrix} 0 \\ 0 \\ k_3 \end{bmatrix} \tag{12}$$

That is:

$$r(k) = k_1 * s(k-2) + k_2 * d(k-1) + k_3 * a(k-1) \tag{13}$$

Under this model, it can be found that at time k, the number of servers used is the measured value at time $k-2$, namely removing servers is also modeled with a delay. When the adaptation decision is to reduce the number of servers, in order to maximize the revenue, it will remove a server ahead of time and increase dimmer to compensate for the impact of forced removing delay. This may cause the actual response time to exceed the threshold and, if not, reduce the load fluctuations the system can withstand.

One way to modify CobRA to allow for asymmetrical latency is to use a dual model. We can design another system model to be used when removing the server, and the state matrix A is the matrix that does not reflect the delay While optimizing the solution, the corresponding model is used when the decision of reducing the server occurs. However, there are many factors need to consider when using multi-model switching of CobRA, such as when and how to switch the models to ensure system states are stable.

5.4 Regularization

In CobRA's cost-funtion, it has constraints on the actuation of control parameters, prefer as little change as possible according to the weight matrixs. However, PLA does not constraint on the actuations. In addition to the expected value of indicators, it is also clear that users are concerned about actuation of control parameters. Frequently actuating the control parameters will cause expensive costs or specific efforts. Consequently, CobRA takes the input signal into account. However, for our experiment, RUBiS is just able to adjust discretely and servers are only allowed to increase one at a time, the difference between PLA and CobRA is not so much serious. We can regard it as when there are more than one ways to achieve maximun utility, CobRA prefers to choose the one with less actuation of control parameters while PLA will choose at random. When there can be huge adjustments, it will make a big difference.

6 CONCLUSION

In this paper, we explain the importance of feedback to the architecture-based proactive self-adaptation method: PLA and propose an explicit feedback model to reduce the impact of modeling error. Based on the work of [14], we analyze and compare the differences between PLA and CobRA considering the effort of feedback, for guiding the approch selection when implementing software self-adaptation and suggest ways to improve.

Gabriel et al. validated that the two approaches can achieve similar adaptation effects. We reproduce their experiments and explore the principle of this similarity. Both of them adopt ideas from MPC, using a receding horizon and prediction model to realize the feature proactive. And both of them heavily depend on feedback to cope with variance in modeling and prediction. Then, we enumerate and analyze the major four differences between the two approaches. PLA models and predicts future environment wheares CobRA regards it as disturbance. However, when the environment changes slowly, PLA can handle the uncertain future environment only using feedback, the same as CobRA. PLA regards the target system as white-box and tries to construct as accurate a model as possible, which needs high costs in engineering. On the contrary, CobRA simply abstract the target system as a dynamic linear model. Although the more accurate the model, the better the prediction, we can obtain satisfying results when replace PLA's exponential model with a linear one, which is much easy to establish. In addition, PLA can model asymmetrical latency of adding servers, while CobRA can not model it but can tolerate a certain degree of model inaccuracy. Moreover, CobRA explicitly constraits on how much the control parameters can be adjusted comparing to PLA.

In my future work, .

REFERENCES

[1] Konstantinos Angelopoulos, Alessandro V Papadopoulos, Vítor E Silva Souza, and John 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] Radu Calinescu, Danny Weyns, Simos Gerasimou, Muhammad Usman Iftikhar, Ibrahim Habli, and Tim Kelly. 2017. Engineering trustworthy self-adaptive software with dynamic assurance cases. *IEEE Transactions on Software Engineering* 44, 11 (2017), 1039–1069.

[3] OW2 Consortium et al. 2013. Rubis: Rice university bidding system. *URL http://rubis. ow2. org* (2013).

[4] Peter 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.

[5] Antonio Filieri, Martina Maggio, Konstantinos Angelopoulos, Nicolas d'Ippolito, Ilias Gerostathopoulos, Andreas Berndt Hempel, Henry Hoffmann, Pooyan Jamshidi, Evangelia Kalyvianaki, Cristian Klein, et al. 2015. Software engineering meets control theory. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press, 71–82.

[6] Mohammad A Islam, Shaolei Ren, A Hasan Mahmud, and Gang Quan. 2015. Online energy budgeting for cost minimization in virtualized data center. *IEEE Transactions on Services Computing* 9, 3 (2015), 421–432.

[7] Donald L Keefer. 1994. Certainty equivalents for three-point discrete-distribution approximations. *Management science* 40, 6 (1994), 760–773.

[8] Jeffrey O Kephart and David M Chess. 2003. The vision of autonomic computing. *Computer* 1 (2003), 41–50.

[9] Cristian Klein, Martina Maggio, Karl-Erik Årzén, and Francisco Hernández-Rodriguez. 2014. Brownout: Building more robust cloud applications. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 700–711.

[10] Marta Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of probabilistic real-time systems. In *International conference on computer aided verification*. Springer, 585–591.

[11] Martina Maggio, Alessandro Vittorio Papadopoulos, Antonio Filieri, and Henry Hoffmann. 2017. Automated control of multiple software goals using multiple

actuators. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering*. ACM, 373–384.
[12] Gabriel A. Moreno. 2017. Adaptation Timing in Self-Adaptive Systems.
[13] Gabriel A Moreno, Javier Cámara, David Garlan, and Bradley 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.
[14] Gabriel A Moreno, Alessandro Vittorio Papadopoulos, Konstantinos Angelopoulos, Javier Cámara, and Bradley 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.
[15] Martin L Puterman. 2014. *Markov Decision Processes.: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.
[16] S Joe Qin and Thomas A Badgwell. 2003. A survey of industrial model predictive control technology. *Control engineering practice* 11, 7 (2003), 733–764.
[17] E. D. Sontag. 2013. *Mathematical control theory: deterministic finite dimensional systems*. Vol. 6. Springer Science & Business Media.
[18] Jiheng Zhang and Bert Zwart. 2008. Steady state approximations of limited processor sharing queues in heavy traffic. *Queueing Systems* 60, 3-4 (2008), 227–246.