Re-Comparison of Two Proactive Self-adaptation Methods: PLA and CobRA

Ben Trovato*
G.K.M. Tobin*
trovato@corporation.com
webmaster@marysville-ohio.com
Institute for Clarity in Documentation
Dublin, Ohio

The Thørväld Group
Hekla, Iceland
larst@affiliation.org

Inria Paris-Rocquencourt
Rocquencourt, France

ABSTRACT

Software-intensive systems face the challenge of uncertainties at runtime which need the ability to adapt itself to keep satisfying the requirements. Recently, there are two proactive approaches, namely PLA and Cobra, which implement the self-adaptation from the perspective of software engineering and control theory respectively and both of them adopt the ideas from model predictive control (MPC). In order to provide a guidance to select an approach, 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. In this paper, we revisit their comparison of the two approaches by exploring the principle for similar adapatations and further investigating their differences and superiorities in case of environment prediction, asymmetric latency and actuation regularizer. We apply the two appraoches to the benchmark system called RUBiS as Gabriel et al. do and reuse their self-adaptation scenario. It is shown that the environment prediction mechanism of PLA contributes less to a better self-adaptation comparing to the implicit feedback mechanism. Moreover, CobRA doesn't model asymmetrical latency which may get poor adaptation results.

CCS CONCEPTS

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

KEYWORDS

self-adaptation, PLA, CobRA, model predictive control

*Both authors contributed equally to this research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference'17, July 2017, Washington, DC, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM. . . $15.00

ACM Reference Format:
Ben Trovato and G.K.M. Tobin. 2019. Re-Comparison of Two Proactive Self-adaptation Methods: PLA and CobRA. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 10 pages.

1 INTRODUCTION

Software-intensive systems suffer uncertainties at runtime, such as fluctuant running environment, changing requirements and unexpected operating conditions which often reduce the quality of service delivery [7]. Therefore, 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 [4]. A self-adaptive system is often implemented as MAPE-K loop [10] which consists monitor, analyze, plan, and execute modules sharing a whole knowledge, shown in Figure 1. For each loop, the monitor module calculates statistical values of the measurements captured by sensors and then the analyze module and plan module, together referred to as descision-making module, generates an adaptation strategy, often corresponding to a set of adaptation tactics, such as adding a server, and finally the execute module will deploy these tactics by effectors.

The early approaches of software self-adaptation are reactive, namely 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. Correspondingly, the later proposed proactive self-adaptation approaches predict future changes to make an adaptation strategy ahead of time and can handle tactic latency between when the tactic is deployed and when the effect is shown.

In the control field, model predictive control (MPC) [17] regards the controlled system as a black-box, often modelled as a linear system, with multiple inputs and outputs. It uses the current measurments, the predicted future outputs for a receding horizon based on system model and the known constrainsts to generate the desired inputs for the receding horizon. To implement software self-adaptation, two main proactive approaches, PLA [14] and CobRA [1], are proposed. Both of them adopt the ideas from MPC, but have different

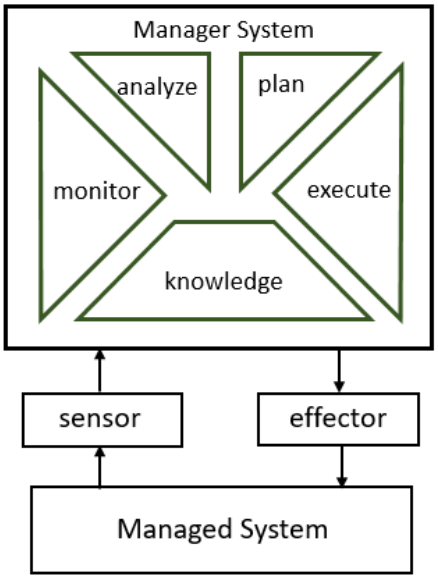

Figure 1: MAPE-K architecture

implementations. Both of them use prediction models to predict future behavior of the managed software system, but PLA describes the possible future behavoir of software system inculding tactic latency and its environment as an integrated Markov decision process (MDP) and tries to predict the future changes of the environment based on historical measurements while CobRA is aimed at introduce model predictive control directly based on specific assumptions. Cobra regards the system as a linear model and treats the changing environment as disturbance. Moreover, Both of them adopt the idea of receding horizon which calculates a sequence of adapation strategies for each loop but only selects the first one.

Despite of the different implementations, the two proactive self-adaptation approaches are all proven to be effective. When we are considering to implement a proactive self-adaptive system, which one should we choose? Gabriel et al. have compared the two approaches in the light of development costs and run-time performance and shown their similar adaptation effects on a benchmark [15]. But they did not analyse why PLA and Cobra can achieve the similar adapation effect and investigate the capabilities of their different implementations for handling uncertainties.

Therefore, based on their work, we try to give an explanations for similar adapatation effects, such as the constraints of model parameters and enumerate the different implementations of the two approaches for handling uncertainties and further analyze their advantages and disadvantages, providing a detailed guidance for approach selecting. We analyzed the differences between the two from three aspects, including 1) modeling environment, 2) asymmetric latency and 3) regular terms for optimization. We theoretically analyzed the reasons and effects of the differences, and follow the same experiment senarios of [15] to verify our conjecture.

In the remainder of the paper, Section II presents the adaptation scenario we used. Section III provides an overview of CobRA and PLA both on design and implementation. Next, Section IV gives an explanations for their similarities. Section V describes three different aspects of CobRA and PLA. Finally, Section VI presents our conclusions and the future work.

2 ADAPTATION SENARIO

The same as [15], we carry out our comparison on the RUBiS system [5], an open-source auction site prototype modeled after eBay.com which is widely used for evaluating self-adaptation and cloud computing [8, 17]. 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 [11] 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.

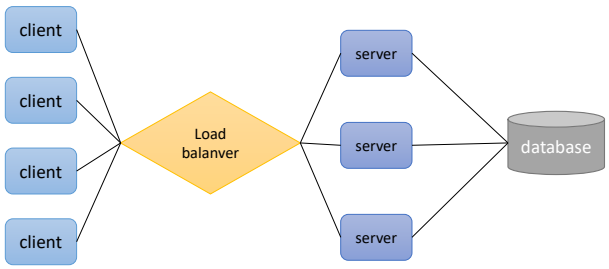

Figure 2: Architecture of RUBiS

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 r below the threshold T; the second one is the target system shall provide high quality of content, namely maximize dimmer 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:

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

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, 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) arrival rate of user requests and 2) latency of booting a server which can be measured aftertime and it provides two types of tactics to adjust itself to deal with these uncertainties:

- `Add/remove a server`: The two tactics are used to change the number of servers s. 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 d 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 METHODS: PLA AND COBRA

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

3.1 PLA

PLA 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 [14].

For each adaptation period τ, PLA uses an autoregresive (AR) time series predictor, such as RPS toolkit [6], to predict the distribution of each environment variable and applies the Extended Pearson-Tukey (EP-T) [9] three point approximation to construct probability trees for next environment prediction. As shown in Figure 3, 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.

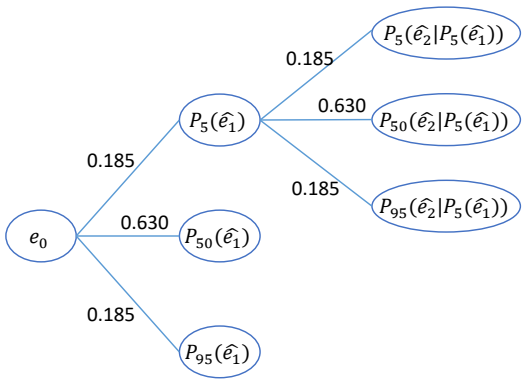

Figure 3: Probability Tree of Environment

The managed system accompanied with adaptation tactics is modeled as a Markov decision process (MDP) [16] which uses state transitions to express all possible execution of adaptation tactics for handling the environment predictions. We can use a probabilistic model checker [14], such as PRISM [12] 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 [19] to calculate the response time under specified configurations of server and dimmer. Obviously, the adaptation performance 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

CobRA 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:

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

where y represents the set of measurable requirement-related variables called *indicators* and u represents a set of adaptation parameters called *control parameters*. y^o_{k+i} and $\hat{y}_{k+i}$ 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 system, the indictors is the average response time r for user requests 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 calculating 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 [13] and a represents measurable environment variables, such as arrival rate of user requests, 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.

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

$$\begin{aligned} &minimize_{\Delta u_{k+i-1}} \quad J_t \\ &subject \quad to \quad u_{min} \leq u_{k+i-1} \leq u_{max}, \\ &\Delta u_{min} \leq \Delta u_{k+i-1} \leq \Delta u_{max}, \\ &\tilde{x_{k+i}} = \tilde{A} \cdot \tilde{x_{k+i-1}} + \tilde{B} \cdot \Delta u_{k+i-1}, \\ &y_{k+i-1} = \tilde{C} \cdot \tilde{x_{k+i-1}}, \\ &\quad i = 1, ..., H, \\ &x_k = x(k) \end{aligned} \tag{6}$$

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.

4 SIMILARITIES

In the work of G.A[15], they show that both of them can handle those uncertainties and maximize the system's utility. We reproduced their experiments and get conclusions even further. We explore the reasons why the two different methods can get similar adaptation control results.

In our experiment, we use the same workload: WorldCup[2] and ClarkNet[3] as Gabriel et al. did and system parameters and utility functions are follwing the settings in **??**.

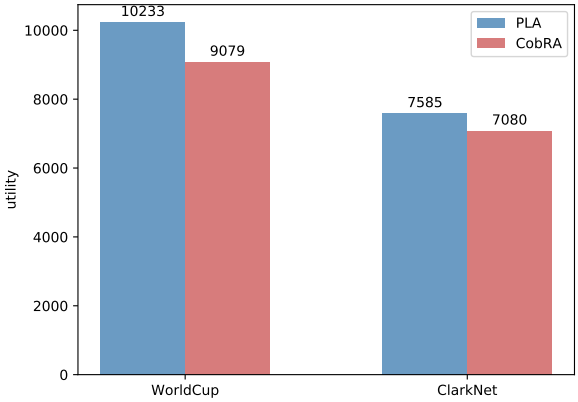

Figure 4: Results of utility

We show total utilities of PLA and CobRA under the two workloads in Figure 4 and Figure 5 show the detailed control results of the two methods under the two workloads. We get utility values more than theirs results and compared to CobRA, PLA can achieve better control results.

4.1 Feedback in PLA and CobRA

The reason for this is we find that both of the two methods heavily depend on the strength and rate of feedback. If we can adjust feedback parameters to a favourable value, we an get even better results of self-adaptation. Through feedback control, PLA can decrease the errors between model and real system to keep r below the threshold as far as possible. However, simple feedback control has some weaknesses resulting in CobRA's slight poor performance than PLA, which we discuss in section IV.

The important factor influencing CobRA's decision making is the setting of parameters in the cost-function and system model, which was also argued in the work of GA. CobRA relies heavily on the value of weight matrix, feedback adjustment parameters and setpoint value, which determine the optimal solution of cost-function.

Although PLA is an open-loop method in terms of architecture, that is, the control effect at current decision period does not affect the decision making at the next period. Open-loop methods can only handle disturbance in measurements and modeling. However, there are many uncertain disturbance in PLA. First, it is inevitable that there will be errors in environment prediction. Second, in implementation, PLA uses LPS queuing model[19] to simulate system's service mode and calculate the average response time of requests. Although LPS model is a pretty accurate description of limited process sharing system with round-roubin strategy, it's calculated

value is the indicators when the system gets a steady state. However, when the system's running, it is difficult to achieve stable states as a result of the load fluctuation. Therefore, there will be errors between the calculated value of queuing model and the actual response time of the system actually. As the system is exponential, when the threshold is triggered, the difference will be significant, which will affect PLA to make correct decisions. However, PLA can still achieve similar effects as CobRA under errors in modeling and prediction because PLA uses feedback in the implementation. In the work of GA[**?**], it is proposed that in the process of implementation, Kalman Filter is used to dynamically adjust the value of service rate according to the differences between the calculated value and the real value of response time. In fact, it adjusts the servicetime parameter *st* of the system, which is the value describing the service capability of the system and should not be adjusted with the change of workload. So this adjustment is just to make the model more realistic, which is not much reasonable. To explore the

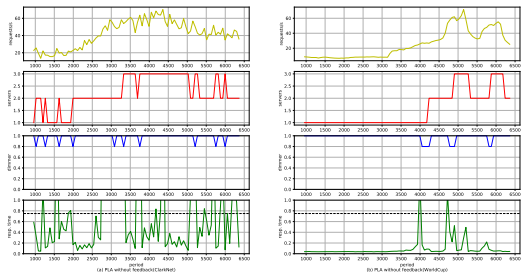

Figure 6: Control effects of nkPLA

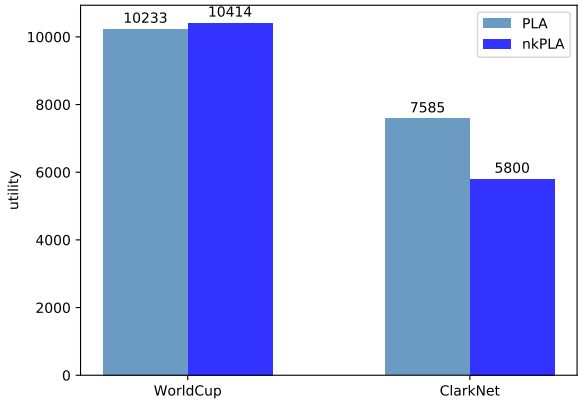

Figure 7: Comparation of utility between PLA and nkPLA

effect of feedback, we construct a variant of PLA, named as nkPLA, which does not use feedback and just depends on prediction. We compare the self-adaptation effects of PLA with nkPLA. Figure 6 shows the results of no feedback under two workloads and Figure 7 compares the utility with PLA. Under the ClarkNet, there are several period the response time exceeding the threshold T but PLA's controller does not make adaptation decisions to handle it. That is because the response time calculated by the queuing theory model is below the threshold value. However, PLA makes decisions based on the calculated value, and it is determined that no control decisions need to be made at this moment. Thus the utility is much lower than with feedback. However, the utility of nkPLA is slightly lower than PLA under WorldCup. That is because the characteristics of this workload and the natural defect of feedback. WorldCup is much stable and then meets a sudden ascend. Although without feedback PLA can't handle the model error, feedback will tend to make tactics conservativly to guarentee the system's stability. Thus the number of servers can not decrease timely, which lead to lower utility value.

4.2 Adaptation Decision

The decision-making of these two methods are actually two different constrained optimization problems. We assume that the two methods are excuting in the same situation. Under this precondition, we disscus the reasons why they can achieve similar control effects under so many differences by deducing their adaptation decision process. For convenience of comparison, we express the control action in the form of "augmented velocity form" of control variables Δu. CobRA's modeling for requirements can be described as a quadratic constrained optimization problem. In control theory, it is most common to use quardratic performance indices to achieve smooth and robust control results. It can be described as:

$$\begin{aligned} J_k = &\sum_{i=1}^{H_p}((y_{k+i}^2 - y_{k+i}))^T Q_i (y_{k+i}^2 - y_{k+i}) \\ &+ (\Delta u_{k+i-1}^T) P_i (\Delta u_{k+i-1})) \end{aligned} \tag{7}$$

We use H_p to denote the length of prediction horizon and H_u denote the control horizon. The control action $u_{k|k}$ at time k can be get from CobRA's optimization problem.

$$u_(k|k) = [I_q \quad 0 \quad 0 \cdots](\theta^T Q \theta + R)^{-1} \theta^T Q (r_k - \psi(\hat{x}_{k|k}) \tag{8}$$

where

$$\begin{aligned} &\psi = [(CA)^T (CA^2)^T \cdots (CA_p^H)^T]^T \\ &\theta = \begin{bmatrix} CB & 0_{n\times q} & \cdots & 0_{n\times q} \\ CAB & CB & \cdots & 0_{n\times q} \\ \vdots & & \ddots & \vdots \\ CA^{H_p-1}B & & \cdots & CA^{H_p-H_u-1}B \end{bmatrix} \\ &Q = diag(Q_1, \cdots, Q_{H_p}) \\ &R = diag(R_0, \cdots, R_{H_u-1}) \\ &r_k = [r_{k+1}^T \quad r_{k+2}^T \quad \cdots r_{k+H_p}^T \quad]^T \end{aligned} \tag{9}$$

and $u_(k|k)$ is the control input applying at time k and is extracted from u_k and is the solution of the optimization problem. $\hat{x}_(k|k)$ denotes the estimated value of state x_k at time k. We can get the $\hat{x}_(k|k)$ by using Kalman Filter as follows:

$$\begin{aligned}
&\hat{x}_{k|k} = \hat{x}_{k|k-1} + L_k(y_k - C\hat{x}_{k|k-1}) \\
&\hat{x}_{k+1|k} = A\hat{x}_{k|k-1} + K_k(y_k - C\hat{x}_{k|k-1}) \\
&L_k = P_k C^T (CP_K C^T + DD^T)^(- 1), K_k = AL_k \\
&P_{tk+1} = AP_t A^T - K_k(CP_k C^T + DD^T)K_k^T
\end{aligned} \tag{10}$$

For PLA, its modeling for utility can be translated into a linear constrained optimization problem. Maximizing utility in the control horizon at period k can be transformed into minimizing the minus of utility, u, which can be expressed as follows:

$$\begin{aligned}
min \quad u_k = \sum_{i=1}^{H_p} ((s_{k+i} + \Delta s_{k+i} * cost \\
- \frac{d_{k+i} + \Delta d_{k+i}}{a_{k+i}} * (R_O - R_M))),
\end{aligned} \tag{11}$$

under the circumstance that:

$$r_k \leq T \tag{12}$$

We use LPS model to calculate the average response time at period k $E[r]_k$ by knowing measured system states:

$$\begin{aligned}
&E[r]_k \approx (\frac{{c_a}_k^2 + {c_s}_k^2}{2} \times \frac{\beta_k}{1 - \rho_k} \times {d_p}_k \\
&+ \frac{{c_a}_k^2 + {c_s}_k^2}{1 + {c_s}_k^2} \times \frac{\beta_k}{1 - \rho_k} \times (1 - {d_p}_k)) \\
&where \\
&({d_p})_k \approx \rho_k^(\frac{1 + {c_s}_k^2}{{c_a}_k^2 + {c_s}_k^2} K) \\
&\rho_k = \beta_k \times \lambda_k \\
&\beta_k = d_k \times st_k + (1 - d_k) \times lst_k \\
&\lambda_k = \frac{1}{a_k \times s_k}
\end{aligned} \tag{13}$$

We consider that the environment is an upward trend. The calculated r of CobRA is lower than the real value because CobRA does not predict the trend of workload. Thus CobRA depends on Kalman Filter to catch the trend of workload and modify the system state. PLA can predict approximate future variation tendency of workload. In addition to prediction errors, queuing theory is apt at modeling systems at steady state. However, during running it is rather difficult for the system to get steady state. Workload fluctuates will lead to requests's backup in servers and result in the inconsistency between prediction and measurement. Therefore, Kalman Filter modify the *st* dynamically according to the distance between calculated and measured r, the same as CobRA. Turn up *st* reults in higher calculated r in next period.

Compared to simple proportional control which does not use receding horizon to predict future system but just use a certain proportion to reflect errors, proactive self-adaptations such as PLA and CobRA can deal with system tactics that have latency in launch. Because if the controller does not consider future state, when the workload is increasing and needs more servers to provide satisfying service, the controller can not predict to add a server ahead of time to deal with the latency.

We can get the conclusion that when there are inevitable errors in prediction and model, the way for PLA to solve these uncertainties is to use feedback. PLA can achieve satisfying control results under so much errors heavily depends on feedback, the same as CobRA.

We can get the conclusion that, the reasons that PLA and CobRA can get similar control result have two aspects:

- `Adopt ideas and structures of MPC.`: First, both of them use models to predict future system behavior, but the models of them are different. Second, both of the employ receding horizon to make the adaptation decision robust and accurate. In addition,
- `Heavily depend on feedback`: They both use feedback to decrease errors between model and the real system, and to reduce the impact from disturbance. Although from the perspective of architecture, PLA is an open-loop method. It needs feedback to deal with errors in prediction for environment and system behavior.

5 DIFFERENCES

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

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

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 kinds of workloads. Figure 8 shows the adaptation process of PLA without environment prediction and Figure 9 shows the utility of PLA with npPLA. As we can seen, the utility of npPLA is 10326 for WorldCup and 6668 for ClarkNet which is similar to the utility with prediction. 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.

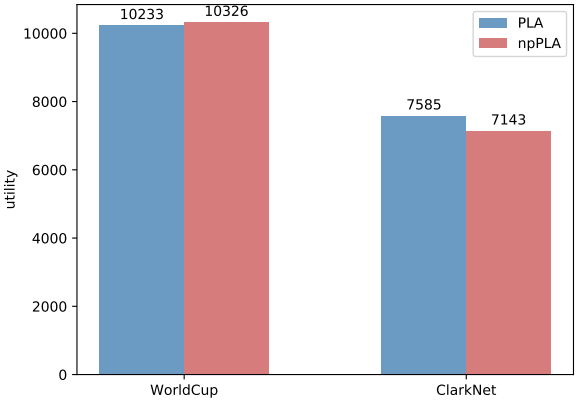

Figure 9: Utility of PLA and npPLA

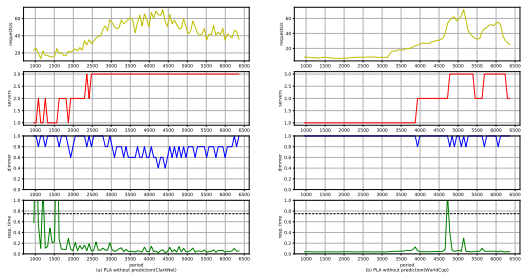

Figure 8: PLA without prediction for environment

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 example, it use LPS queuing model 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 [18]. 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 the highly nonlinear system, such as RUBiS which is an exponential system and sensitive to fluctuation of workload, a linear model may makes poor self-adaptation.

In this section, we want to discuss the contributions of PLA's precise model to a good self-adaptation. We design a variant PLA named linearPLA to discuss the effectiveness of precise model. The same as CobRA, linearPLA uses a time-invariant liear model to describe the system instead of exponential LPS model. 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.

However, the precise system model of RUBiS is a steady state approximation of LPS queue in heavy traffic [19] and there are significant deviations between predicted indictors and their transient measurements which is relied on feedback mechanism to offset. Figure **??** shows its adaptation process with the same two workloads and Figure **??** shows the utilities of PLA with a precise model and a linear model respectively. Although linear model is more inaccurate than LPS model,

they can get about the same adaptation effects because of large deviation of precise model prediction. In conclusion, if the precise system model and other predction models also have signficant model uncertainties, a linear model with a good designed feedback mechanism is a good choice to implementing self-adaptation.

5.3 Asymmetrical Latency

In RUBiS system, there is a certain latency when adding a server, but removing a server can be regarded to be 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 will cost the same time to shown the effect of increasing or decreasing the values of control parameters, namely the adding/increase or removing/decrease adaptation tactics 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 performance, 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:

$$r = k_1 * s + k_2 * d + k_3 * a \tag{14}$$

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:

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

That is:

$$r(k) = k_1 * s(k-2) + k_2 * d(k-1) + k_3 * a(k-1) \tag{15}$$

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, $A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$, 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

Based on the work of [15], we analyze and compare the similarities and differences between PLA and CobRA for guiding the approch selection when implementing software self-adaptation.

Gabriel et al. have validate that the two approaches can achieve similar adaptation effects. We explore the principle of this similarity. Then, we enumerate and analyze the major four differences between the two approaches. We find that the adaptation performance of PLA are high relied on feedback mechanism when there are significant devirations in model prediction and system precise modeling, leading to the similar adaptation performance of PLA without environment prediction and with linear modeling respectively. 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] Martin Arlitt and Tai Jin. 2000. A workload characterization study of the 1998 world cup web site. *IEEE network* 14, 3 (2000), 30–37.

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

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

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

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] Martin L Puterman. 2014. *Markov Decision Processes.: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

[17] S Joe Qin and Thomas A Badgwell. 2003. A survey of industrial model predictive control technology. *Control engineering practice* 11, 7 (2003), 733–764.

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

[19] 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.

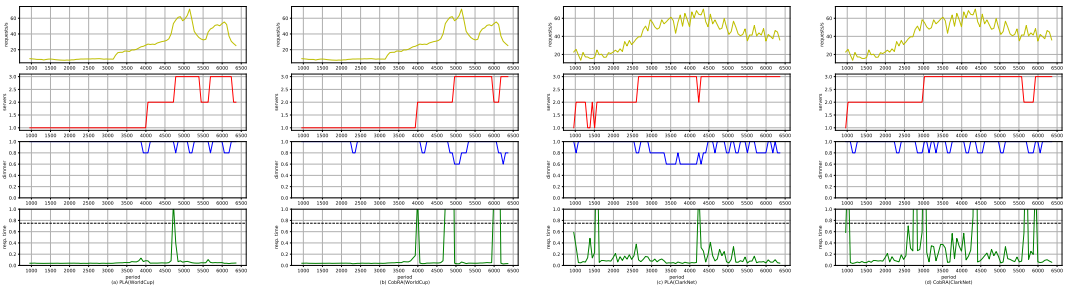

Figure 5: Control results of PLA and CobRA