

# 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 Thørvöld Group  
Hekla, Iceland  
larst@affiliation.org

Inria Paris-Rocquencourt  
Rocquencourt, France

## ABSTRACT

Software-intensive systems tend to face more and more uncertainties in running environment, which are difficult to engineer and expensive to consider at design time. Therefore, software systems need to have ability of self-adaptation, to monitor the change of uncertainties and adjust itself dynamically to keep approaching the target. As we consider which methods we should choose to use, we weigh their engineering costs and resulting effects. Currently, there are most advanced two proactive adaptive methods, PLA[7] and CobRA[2], both adopting ideas from MPC. Gabriel A. Moreno compared the two methods in aspects of development costs and run-time performance[4]. They spend different efforts on modeling and achieve different effects, but both of them can achieve satisfying adaptation control. In this paper, we follow the experimental scenario of G.A and explore the reasons for the similarities and differences between the two on the basis of their work. We find that the open-loop method is difficult to cope with uncertainties such as errors from modeling and prediction. PLA needs to combine feedback to achieve good control effect. We analyze and verify the major differences between them.

## CCS CONCEPTS

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

## KEYWORDS

self-adaptation, PLA, CobRA, model predictive control

## ACM Reference Format:

Ben Trovato and G.K.M. Tobin. 2019. Comparison of Two Proactive Self-adaptation Methods: PLA and CobRA. In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, 8 pages.

## 1 INTRODUCTION

Why do software systems need capabilities of self-adaptation? The main reason is that software-intensive systems face a lot of uncertainties while running, such as fluctuant running environment and changing requirements. While in the design time of software systems, considering all those uncertainties is not realizable and needs high cost. However, due to some properties of the software system, we can design an adaptive mechanism to tolerate these uncertainties. The main difference between traditional software engineering and self-adaptation software engineering is that, in traditional software, we will anticipate changes that the software system will face in the future and give the response to them. On the contrary, for self-adaptation, we do not need to design as carefully as possible the solution to all changes that maybe happen, we just need to know some feature of the system, then we can design an adaptation controller to handle these changes according to these features. Software self-adaptation can be defined as an ability of software system to monitor uncertainties during running and then make adaptive tactics to response to these changes in order to maximize its goal. As shown in figure??, software self-adaptation often bases on MAPE-K loop[5], consisting monitor, analyze, plan, and execute modules sharing a whole knowledge. The control manager gets measurements from managed system through sensors and then use effectors to execute adaptation tactics.

However traditional software self-adaptation are reactive, they only make adaptive tactics after changes happened, do not predict what will happen in the future. On the contrary, proactive self-adaptation can predict what will happen in the future, to make tactics ahead of time. Currently, there are works using control theory to design self-adaptation for software. There is a famous class of control methods in the control field called model predictive control(MPC) to explicitly deal with multivariable systems with constraints. MPC abstracts the system into a linear model of multiple inputs and outputs and then use this model to predict the behavior of systems in the future. It uses a receding horizon to get

---

\*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

the optimal control. Cobra is a typical example of applying control theory to software adaptation. Cobra is a requirement-based approach that employs traditional control theory ideas and regards the system like a black box. It abstracts the system into a simple linear model, modeling requirements into cost function, using feedback to smooth the effects of modeling error and disturbance. There is a noted self-adaptation method based on ideas of traditional software engineering called PLA. It does not model the system as linear, on the contrary, it regards the system as a white box and constructs precise MDP model to predict the future system and environment. The goal of the system is modeled into revenue function. Then we can use probabilistic model checkers[?] to exhaustive all can reach the path and get the solution which can maximize the revenue function. Although PLA and CobRA have different design ideas and implementation methods, they both adopt MAPE-K loop[5] and employ the idea from MPC. That is, 1) both use prediction models to predict future behavior. They establish different models for the system and environment. Cobra models the system as linear and treats the environment as disturbance, while PLA models the system and environment as an integrated MDP and predicts the future environment. 2) adopt the idea of receding horizon to get a optimal solution among the horizon. They compute a series of control actions but only commit the first one.

When we are considering which method to choose, there are some facts that we need to weigh. The most important one is whether the method can handle the problems and achieve the goal of object system. When we model the system, we are simplifying the system, thus inevitably there will have errors between the model and the real system. Unlike in the physical world, which can be modeled by exact mathematical equations, software systems are always more complex to model. Although it seems that MDP is more accurate than linear model, but it also exist errors and need more efforts while building. CobRA is not much precise, but it has many theory of guarantees of stability. If we can achieve similar results, obviously we want to use method that needs fewer effort.

Gabriel A. Moreno compared the two methods both in the light of development costs and run-time performance. While the two approaches perform comparably in general, their run-time behavior can be significantly different, both in terms of resource utilization and the ways in which they attempt to maximize performance. Both of them can get fine results after elaborate adjustment of feedback parameters. But they did not analyse the reasons why they achieve the same effect, or the reasons and consequences of their differences. Therefore, based on their work, we hope to further analyze the reasons why these two methods can achieve the same result, as well as the consequences of their differences under specific scenarios, so as to have a better understand of the applicability of these two methods.

In this work, we first analyze the similarities between the two, and deduce the reasons why the two can achieve similar effects from the perspective of their decision-making

process. Then, we analyzed the differences between the two from three aspects, including the three aspects of 1) modeling environment, 2) asymmetric latency and 3) regular terms for optimization. We theoretically analyzed the reasons and effects of the differences, and use the same experiment senario as G.A[4] to verify our conjecture through experiments.

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 details the reasons for their similarities. Section V describes three different aspects of CobRA and PLA. Finally, Section VI presents our conclusions and directions for future research.

## 2 TWO PROACTIVE SELF-ADAPTAION METHODS: PLA AND COBRA

Both of the two approach are based on MPC[3] idea. That is, 1) use models to predict behavior of system, 2) adopt idea of receding horizon. To achieve proactive features, both of them consider an adaptation *horizon*, which is some periods in the future. For our senario, we first define the target problems to be solved by the adaptive methods.

- **Adaptation uncertainties:** There are two kinds of uncertainties faced by self-adaptation controller. One is the uncertainty to be solved by the adaptive system, which is the external uncertainty. For RUBiS system, it includes the arriving rate of requests, the delay of how long it takes to start a server, and how long it takes to process a request, etc. The other type is the uncertainties faced in the design of controller. One is that there will be errors between the actual system and the model, as well as in the PLA's environmental prediction and data measurement. The solution of the first kind of uncertain reflects the control effect of the adaptive method itself. Tolerance for the second type of uncertainty reflects the properties of the controller, such as stability, which can be measured by SASO properties[?]. The second type of uncertainty will seriously affect the realization of the the non-functional requirements. In order to ensure the completion of their control goals, excellent controllers need to have the ability to tolerance uncertainties.
- **Adaptation goal:** The goal of adaption controller is to decide how to adapt to maximize the utility the system will accrue over the look-ahead horizon, with the monitoring, analysis, planning and executing part, dealing with the influence of uncertainties. And we expect the controller performance is robust, having the ability of resisting disturbance.

We describe PLA and CobRA in terms of design details and implementation.

## 2.1 PLA

The key idea of PLA is to let the adaptation decision underspecified through nondeterminism and construct a formal model of the adaptive system, then have the model checker resolve the nondeterministic choices to achieve the accumulated utility over the horizon is maximized. The adaptation decision is optimal over the decision horizon, and takes into account the inherent uncertainty of the environment predictions needed for looking ahead.

For PLA, decisions are made not only based on historical circumstances and current measurements, but also on the predicted future. We discretizes the execution of adaptation decision in periods of  $\tau$ , and the system behavior in the adaptation horizon is predicted by modeling the system into a MDP[?], which uses state changes to express the execution of adaptation tactics.

The environment can be modeled as a DTMC model in which the random variable representing the state of the environment has one change at each evaluation period  $\tau$ . PLA uses an autoregressive (AR) time series predictor[?] to predict future environment, such as RPS toolkit[?], which requires that the changes of environment should be predictable and regular. Because the predictor can provide the variance of the estimation, PLA uses the Extended Pearson-Tukey (EP-T)[?] threepoint approximation to construct probability trees for decision making. As shown in figure??, the root  $e_0$  of the tree is current environment, and  $e_1$  is the estimation distribution of environment in next period with probabilities 0.185, 0.630, and 0.185, respectively.

**Figure 1: Probability Tree of Environment**

Unfortunately, there will be large errors in the prediction of future environment. In addition, sudden fluctuations cannot be predicted, which will have a great impact on decision-making. This will be analyzed in detail later. Finally, the system and environment are built into a whole MDP model, and the goals of the system are constructed into the form of utility function. For RUBiS system, the models of the system and environment are stored in *knowledge*. At every beginning of adaptation period, monitor gets measurements from the system and then update the models.

We can use a probability model checker[?] to analyze the model, verify the attributes, exhaust all paths and find a strategy to maximize the revenue function. Because of the round-robin strategy of RUBiS, we use LPS queuing theory model[?] to calculate the response time of the system. In design time, PLA need plenty of effort to build an accurate model of the environment and the target system. It is a feed-forward method, thus the result largely depend on modeling.

## 2.2 CobRA

CobRA adopts more ideas of control theory. The non-functional requirements of RUBiS system are captured by the softgoals: High Performance, Low Cost, and High Optional Content

Availability. CobRA refers to adaptation goals as *indicators*, for RUBiS system, they are:  $r$  for average response time of requests,  $s$  for number of servers and  $d$  for dimmer value, and constitute the system's output. The parameters of the system that can be tuned in order to fulfil its goals are called *control parameters*. For RUBiS system, *control parameters* are the number of servers  $s$  and the dimmer value  $d$ , which constitute the system's input. As well as PLA, CobRA discretizes the execution timeline in adaptation periods of  $\tau$ , and solves the adaptation decision problem at the start of every period.

CobRA abstracts the relationship between the input and output of the system into a simple linear model. The model represents affects of control parameters to indicators over time. We can use a discrete-time linear dynamic model to describe the system behavior at the  $k$ -th decision period.

$$\begin{cases} x(k+1) = A \cdot x(k) + B \cdot u(k) \\ y(k) = C \cdot x(k) + D \cdot a(k) \end{cases} \quad (1)$$

where  $x$  is the state of the system,  $u$  is the set of control parameters,  $y$  is the set of indicators and  $a(k)$  is the average arrival rate of requests at the  $k$ -th decision period.

For the uncertain environment, CobRA does not predict future environment, instead, it treats the environment as disturbance to the system. In our scenario, the environment is uncertain arrival rate of requests  $a$ , that is used as a feedforward signal. The effect of environment's uncertainty is reflected by Kalman Filter on the system's states.

CobRA is requirement-based, for RUBiS system, the system's goal can be described as three output variables: the number of servers  $s$ , the dimmer value  $d$ , and the average response time of requests  $r$ . The goal of the system is that each indicator should be close to its target value, and the goal of the system is designed into the form of cost-function. CobRA make decisions by modifying the value of the control variables.

$$J_k = \sum_{i=1}^H [y_{k+i}^o - y_{k+i}]^T Q_i [y_{k+i}^o - y_{k+i}] + [\Delta u_{k+i-1}]^T P_i [\Delta u_{k+i-1}] \quad (2)$$

where  $Q_i$  and  $P_i$  are symmetric positive semi-definite weighting matrices.  $Q_i$  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.  $y_{k+i}^o$  are setpoints of indicator  $i$ , which means the goal value of this indicator.

The optimization problem can be defined as follows:

$$\begin{aligned} & \text{minimize}_{\Delta u_{k+i-1}} \quad J_t \\ & \text{subject to} \quad u_{\min} \leq u_{k+i-1} \leq u_{\max}, \\ & \quad \Delta u_{\min} \leq \Delta u_{k+i-1} \leq \Delta u_{\max}, \\ & \quad x_{k+i} = \tilde{A} \cdot x_{k+i-1} + \tilde{B} \cdot \Delta u_{k+i-1}, \\ & \quad y_{k+i-1} = \tilde{C} \cdot x_{k+i-1}, \\ & \quad i = 1, \dots, H, \\ & \quad x_k = x(k) \end{aligned} \quad (3)$$

CobRA makes adaptation decisions to keep indicators as close to setpoints as possible. The control scheme is represented as figure??

**Figure 2: Control Scheme of CobRA**

### 3 ADAPTATION SENARIO

Based on G.A's work, we carry out our comparison on the RUBiS system[10], an widely used open-source application for evaluation of self-adaption and cloud computing[8],[11]. To enable replicating experiments with the same conditions under control for the two approaches, avoiding unexpected fluctuation and uncertainties during real systems running, we ran our experiments on a simulated system of RUBiS, which is named SWIM[1].

RUBiS system consists of a load-balancer, a web server tier and a database tier. Clients use browsers to send requests to the server. Load balancer distributes requests among servers following a round-robin policy. Servers access the database to obtain the data required to render the dynamic content of the page. Figure ?? shows the architecture of RUBiS system. We introduce the characteristics of RUBiS system from the

**Figure 3: Architecture of RUBiS**

three aspects of uncertainties, tactics and utility:

- **Uncertainties:**

The main uncertainties of RUBiS system, which is need to sovled by the adaptation controller, can be divided into three categories:

- (1) **Environment** For RUBiS system, the only relevant property of the running environment that we consider is the arriving requests on the system. If the arrival rate of requests increases suddenly, the system need more resource to handle them.
- (2) **Latency of adding a server** For RUBiS system, there is a period time that before a new server can start working. We call it *latency* of adding servers.
- (3) **How long it needs to process a request** That is the service time in RUBiS system depending on the performance of the system and the nature of the specific requests.

- **Tactics:** For RUBiS system, the main uncertainty that self-adaptation want to handle is the uncertain environment. RUBiS system can execute two types of tactics to adjust itself in order to deal with uncertainties:

- **Add/remove server** There is a delay in adding a server, which we call it latency, whereas removing a server can be consider as no latency, because the latency for removing a server is negligible.
- **Increase/decrease dimmer value** RUBiS system follows the brownout paradigm[9], which can response to a request includes mandatory content and

optional content. The dimmer parameter indicates the proportion of a request including optional content(value between  $[0, 1]$ ).

- **Utility:** The goal of the target system can be described as two functional and three non-functional requirements. Functional requirements are tasks of the system, whereas we capture non-functional requirements formally in a utility function that enables us to quantify their satisfaction. The utility function is modeled as follows:

$$U_\tau = \begin{cases} U_R + U_C & r \leq T \wedge U_R = U_R^* \\ U_R & r \leq T \wedge U_R < U_R^* \\ \tau \min(0, \alpha - \kappa) R_O & r > T \end{cases} \quad (4)$$

– utility associated with *cost* per time interval:

$$U_C = \tau \cdot c \cdot (s^* - s) \quad (5)$$

– utility associated with revenue per time interval:

$$U_R = \tau \cdot \alpha \cdot (d \cdot R_O + (1 - d) \cdot R_M) \quad (6)$$

where  $\tau$  is the length of the interval,  $\alpha$  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.

The most important requirement is keeping the response time  $r$  below the threshold  $T$ . The second one is the target system shall provide high quality of service, which is described by dimmer value  $d$ . The third one is that the target system shall use little number of servers  $s$  to operate under low cost. The whole goal is to get higher revenue while generate lower cost.

### 4 SIMILARITIES

Although there are great differences between the two methods in terms of design and implementation, both of them can achieve a reasonable control effect similarly. In the work of G.A[4], they compared the performance of the two methods, including adaptive control performance and SASO property[?]. They compared the utility value of the two methods when they are running and concluded that in different scenarios, the two methods have different control effects on dimmer value and server numbers, but for the same scenario, they ultimately achieve similar utility value. We would like to explore the reasons why they can achieve similar control effects under so many differences.

The decision-making of these two methods are actually two different constrained optimization problems. For convenience of comparison, we express the control action in the form of "augmented velocity form" of control variables, similar to the description in CobRA. For PLA, it is a linear constrained optimization problem. Maximizing utility can be transformed into minimizing the minus of utility,  $u$ , which can be expressed

as follows:

$$\begin{aligned}
\min \quad & U = (s_0 + \Delta s) * cost - \frac{d_0 + \Delta d}{a} * (N - L), \\
\text{s.t.} \quad & r \leq T, \\
& -0.1 \leq \Delta d \leq 0.1, -1 \leq \Delta s \leq 1, \\
& 0 \leq d_0 + \Delta d \leq 1, 1 \leq s_0 + \Delta s \leq 3
\end{aligned} \tag{7}$$

Under the current measured system configuration and environment state, that is, known  $s_0$ ,  $d_0$  and  $a$ , by solving this optimization problem, we can get the control increment,  $\Delta s$  and  $\Delta d$  to minimize  $u$  while satisfying their constraints. Because of PLA's prediction for environment, in every decision period in the horizon,  $a$  is the prediction of future prediction, which is different from CobRA following.

For CobRA, it is a quadratic constrained optimization problem. In control theory, it is most common to use quadratic performance indices to achieve smooth and robust control results. It can be described as:

$$\begin{aligned}
\min \quad & J = (r' - r^o)^2 Q_{33} + (s_0 + \Delta s - s^o)^2 Q_{11} \\
& + (d_0 + \Delta d - d^o)^2 Q_{22} + (\Delta s)^2 R_{11} + (\Delta d)^2 R_{22} \\
\text{s.t.} \quad & -0.1 \leq \Delta d \leq 0.1, -1 \leq \Delta s \leq 1, \\
& 0 \leq d_0 + \Delta d \leq 1, 1 \leq s_0 + \Delta s \leq 3
\end{aligned} \tag{8}$$

In the current measured system configuration and environment state, CobRA hopes to find the control increments that can minimize  $J$  and satisfy the constraints at the same time. The environment state  $a$  here is the measured environmental value. However, CobRA will use Kalman Filter to adjust the system state to reflect the fluctuation of environment.

Obviously they are two different constrained optimization problems, can they get the same solution under the same input? We can obtain KKT conditions for two problems through Lagrangian multiplier method. 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. In our scenario, if we want to optimize the solution of CobRA to maximize the utility of the system, we think that the values of setpoints,  $s^o$ ,  $d^o$  and  $rt^o$ , should change in every decision period. However, in many related works, setpoints are rarely changed, and are always set to fixed values. Moreover, we think it is not reasonable to set setpoint  $rt^o$  to 0, because the response time of requests approaching 0 does not make the system optimal, but approaches a value below the threshold of the response time. Also, it is not reasonable to set  $rt^o$  as the threshold, because it is impossible for each indicator to reach the target value in many cases. Instead, it fluctuates around the target value according to the weight matrixes. However, the threshold value of response time seriously determines the revenue calculation of the system. Once the threshold value is exceeded, the revenue will be punished directly, so we should choose a more appropriate

value between 0 and the threshold value as the setpoint of response time.

We assume that PLA obtains a optimal solution  $s_0$  and  $d_0$ , to the problem in a given state. We set the setpoint in CobRA of  $s$  and  $d$  to  $s_0 + \Delta s$  and  $d_0 + \Delta d$ .

$$\begin{aligned}
\min \quad & J = (s_0 + \Delta s - s^o)^2 Q_{11} + (d_0 + \Delta d - d^o)^2 Q_{22} \\
& + (r' - r^o)^2 Q_{33} + \Delta s^2 R_{11} + \Delta d^2 R_{22} + \alpha_1(-\Delta d - 0.1) \\
& + \alpha_2(\Delta d - 0.1) + \alpha_3(-d_0 - \Delta d) + \alpha_4(d_0 + \Delta d - 1) \\
& + \alpha_5(-\Delta s - 1) + \alpha_6(\Delta s - 1) + \alpha_7(-s_0 - \Delta s + 1) \\
& + \alpha_8(s_0 + \Delta s - 3)
\end{aligned} \tag{9}$$

where

$$\begin{aligned}
r' &= k_1 * (s_0 + \Delta s) + k_2 * (d_0 + \Delta d) + k_3 * a, \\
\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7, \alpha_8 &\geq 0
\end{aligned} \tag{10}$$

We can get the KKT optimality conditions as follows:

$$\begin{aligned}
& 2(s_0 + \Delta s - s^o)Q_{11} + 2k_1(r' - r^o)Q_{33} \\
& + 2\Delta s R_{11} - \alpha_5 + \alpha_6 - \alpha_7 + \alpha_8 = 0, \\
& 2(d_0 + \Delta d - d^o)Q_{22} + 2k_2(r' - r^o)Q_{33} \\
& + 2\Delta d R_{22} - \alpha_1 + \alpha_2 - \alpha_3 + \alpha_4 = 0, \\
& \alpha_1(-\Delta d - 0.1) = 0, \alpha_2(\Delta d - 0.1) = 0, \\
& \alpha_3(-d_0 - \Delta d) = 0, \alpha_4(d_0 + \Delta d - 1) = 0, \\
& \alpha_5(-\Delta s - 1) = 0, \alpha_6(\Delta s - 1) = 0, \\
& \alpha_7(-s_0 - \Delta s + 1) = 0, \alpha_8(s_0 + \Delta s - 3) = 0
\end{aligned} \tag{11}$$

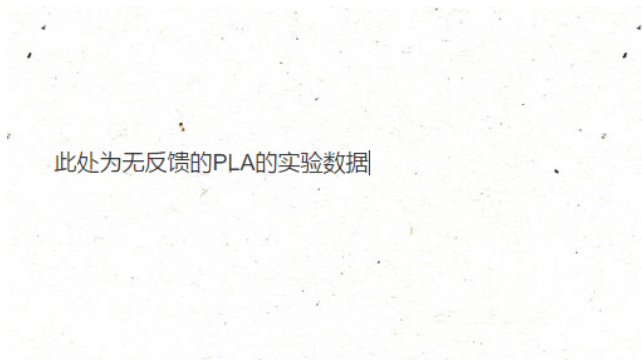
It needs to be discussed separately when  $s_0, \Delta s, d_0$  and  $\Delta d$  take different values. But we can get the conclusion when  $rt^o$  and weight matrixes  $Q, R$  meet some specific relationships, the optimization solution of CobRA can also get the same solution as PLA. In fact, the constraints on the variation of the control parameters are not the same here. The reason is that CobRA applies to continuous parameters while PLA applies to discrete parameters. In RUBiS system, before executing the tactic, we need to discretize the continuous variables obtained by CobRA, and the discretization process may produce the phenomenon that the obtained values are not exactly the same, but they can reach the same result after discretization.

Through the above analysis, we can know that under the same initial state, the two can obtain similar control decision results. However, at run time, it is difficult for them to implement the same control input for each decision, due to their different abstraction of the model and prediction for the environment. PLA is an open-loop method in terms of architecture, that is, the control effect at one decision period does not affect the decision making at the next period. However, they can still achieve similar effects because PLA also 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 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. There are many errors in PLA. First, it is inevitable that there will be errors in environment prediction. Second, in implementation, PLA uses LPS queuing model to simulate the system modeling, LPS queuing theory model is very accord with RUBiS system service mode. However, queuing theory models calculate system parameter values under steady state, but during running, as a result of the load fluctuation, systems are difficult to achieve stable states. Therefore, there will be a certain error between the calculated value of queuing model and the actual response time of the system. As the system is exponential, when the critical value is triggered, the difference will be significant, which will affect PLA to make correct decisions. In order to explore the influence of the feedback,

**Figure 4: Errors in response time calculation**

we remove the feedback in PLA. figure?? shows the actual response time and the calculation of response time under ClarkNet, there is a certain gap between leads to PLA method failed to under the condition of the response time is more than threshold make reasonable decisions. In figure??, we can see that there are several period the response time is exceeding the threshold but pla 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. PLA makes decisions based on the calculated value, and it is determined that no control decisions need to be made at this moment.



此处为无反馈的PLA的实验数据

**Figure 5: PLA without feedback**

We can get the conclusion that PLA and CobRA are facing a lot of errors at run time, PLA can tolerates errors because it introduces feedback into the implementation. When there are inevitable errors in prediction and model, the only way to solve this uncertainty is to use feedback.

## 5 DIFFERENCES

By analyzing the design ideas of PLA and CobRA, we can obtain some differences between them, which are mainly described in table 1. In GA's work, it has been analyzed that there are significant differences between the two in those aspects: artifact, prerequisite and specification. But we want to talk more about the context in which these differences make a significant difference, and how these differences affect the control effect. We discuss the differences from three aspects. They are the modeling of asymmetric delay, the difference of environment prediction, discuss how much prediction plays a role, and the constraint of regular term. We followed G.A's parameter settings and workloads, worldcup[?] and clarknet[6].

### 5.1 Prediction of Environment

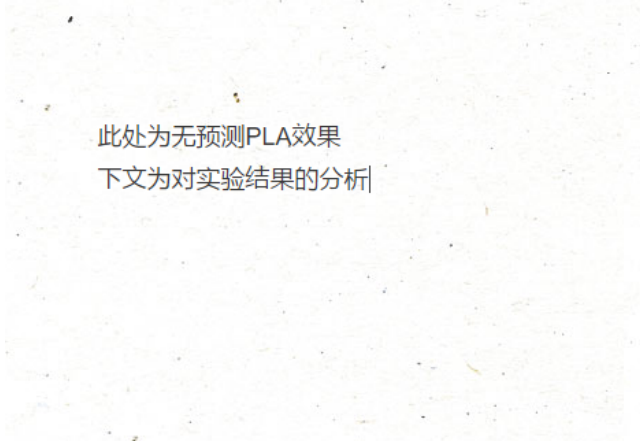
One of the most significant differences between the PLA and CobRA is that they model the environment differently. PLA regards the environment as an important uncertainty factor of the system and the target problem of the adaptive control. PLA believes that the future environment will change. At the beginning of every decision period, current and historical environmental data can be obtained. PLA uses known data to predict the future environment in the horizon, and predicts the future response time according to the predicted value of the environment, so as to make decisions. CobRA regards the environment as one of the disturbances of the system and does not predict the environment, but as a direct component adding to the state space function. Kalman Filter is used to achieve smoothing and feedback, so as to reflect the influence of the environment to the state change of the system. Both of the two methods take account of fluctuations in the environment, but they respond in different ways, one using prediction and the other using feedback.

Theoretically, the feedback is one-step slower than the current state, but prediction also have errors. In addition, although the two methods are different, they have the same assumption for the environment, which requires the environment to change slowly and regularly, so as to ensure that the prediction error is tolerable, and the feedback adjustment can produce appropriate effects. ??PLA still needs to use feedback to make up for the prediction error, although it makes prediction to the environment. If PLA does not predict the future environment, but only uses the feedback correction system state to reflect the changes of the environment, this is very similar to CobRA. We discuss whether similar results can be produced, so as to determine whether the prediction is necessary, and try to propose improved PLA and CobRA

PLA cobracobra We experiment with two wrokloads respectively, and compared the indicators and utility values. It can be found that if there is no prediction and only relying on the feedback, the experimental results depend on the feedback strength, that is, the Kalman Filter parameter setting, we can also tune a very good result. In fact, prediction is not necessary in PLA, because the error of prediction still needs to be compensated by feedback.

**Table 1: Differences between PLA and CobRA**

Features	PLA	CobRA
Method	Architecture-based	Requirement-based
Model for Environment	Predict and model to DTMC	Do not predict and regard as disturbance
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

**Figure 6: PLA without prediction for environment VS prediction**

We are also considering whether it would be better to introduce environmental prediction to CobRA. But the experimental results are not so noteworthy. The reason for this is that CobRA assumes that the environment is stable, so Kalman Filter is sufficient to deal with the stable environment.

## 5.2 Asymmetric Latency

Although RUBiS system is relatively simple, it has some features that are worth exploring for similarities and differences between the two methods. In RUBiS system, for the couple of decisions about server numbers, there is a certain latency when increasing the number of servers, but reducing the number of servers can be regarded as no delay. This is an asymmetric latency with a control parameter. PLA models each decision and the asymmetric latency is well described by the MDP model. However, in CobRA, asymmetric delay was not taken into account. CobRA's model considers that the delay was symmetric, that is, after modeling the latency of adding server, removing server was also considered having latency. This will cause CobRA to make the decision to reduce the server in advance, and increase dimmer to compensate

when reducing the server, reflecting the characteristics of concurrent.

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

$$r = k_1 * s + k_2 * d + k_3 * a \quad (12)$$

To translate this into a state space model, we use the state matrix  $A$  to reflect the latency of adding server. The matrixes

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:

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

Under this model, it can be found that at time  $k$ , the number of servers used is the measured value at time  $k-2$ , leading to a delay in removing servers. Therefore, when the decision is made to reduce the number of servers, in fact, the number of servers immediately decreases, which will lead to an error in the calculation of response time and affect the decision-making of the system.

At this point, CobRA decides that it needs to reduce the number of servers (why? As the system described in the model has a delay in reducing the number of servers, the controller considers that the number of servers cannot be reduced during this period. In order to maximize the revenue, it will choose to increase dimmer to compensate for the impact on the revenue during this period. This may cause the actual response time to exceed the threshold and, if not, reduce the load fluctuations the system can withstand.

latency

However, in the field of control, it has been proved that the MPC method can tolerate the imprecision of the model to a certain extent due to the function of feedback regulation. We ran an experiment to verify this conclusion. One way to modify CobRA to allow for asymmetric delays 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.

### 5.3 regularization

In CobRA's cost-function, it has constraints on the actuation of control parameters, prefer as little change as possible according to the weight matrixes. 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 maximum utility, CobRA prefer 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

GAPLA cobraGAPLAPLAPLAPLAPLAcobramppcobraPLAco-braPLAcobra Based on the work of GA, we analyzed and compared the similarities and differences between the PLA and CobRA. In the work of GA, it was proposed that the two could achieve similar control effects. We explored the reasons and found that the PLA's solution to the prediction error and model error was feedback adjustment. As PLA is an open-loop method, the errors caused by modeling and prediction cannot be solved when PLA method has no feedback regulating effect. Secondly, we analyzed the differences between the two in three aspects. Firstly, PLA can model asymmetric latency of adding servers, while CobRA does not model that. However, studies in the control field have proved that the MPC method can tolerate a certain degree of model inaccuracy. When the latency is larger than that, the control performance of CobRA will be greatly affected. The second aspect is the difference in environment modeling between the two methods. PLA predicts the future environment, while CobRA regards the environment as disturbance and does not predict, but reflects the impact of the environment to the state through feedback. With feedback from both, we discussed the necessity of prediction in the PLA. It was found that the effect of prediction was not obvious under the effect of feedback. The third point is that CobRA is limited in how much you can control. Under what circumstances does this make a clear distinction between the two. In my future work, I hope to make quantitative measurement and draw more general conclusions.

## REFERENCES

- [1] Association for Computing Machinery 2007. *ACM Visual Identity Standards*. Association for Computing Machinery. <http://identitystandards.acm.org>.
- [2] Rogério Brito. 2009. *The algorithms bundle*. <http://www.ctan.org/pkg/algorithms>.
- [3] David Carlisle. 2004. *The textcase package*. <http://www.ctan.org/pkg/textcase>.
- [4] Michael Downes and Barbara Beeton. 2004. *The amsart, amspdoc, and amsbook document classes*. American Mathematical Society. <http://www.ctan.org/pkg/amslatex>.
- [5] Michael Downes and Barbara Beeton. 2004. *The amsart, amspdoc, and amsbook document classes*. American Mathematical Society. <http://www.ctan.org/pkg/amslatex>.
- [6] Michael Downes and Barbara Beeton. 2004. *The amsart, amspdoc, and amsbook document classes*. American Mathematical Society. <http://www.ctan.org/pkg/amslatex>.
- [7] Cristophe Fiorio. 2015. *algorithm2e.sty—package for algorithms*. <http://www.ctan.org/pkg/algorithm2e>.
- [8] Axel Sommerfeldt. 2013. *The subcaption package*. <http://www.ctan.org/pkg/subcaption>.
- [9] Nicola L. C. Talbot. 2016. *User Manual for glossaries.sty v4.25*. <http://www.ctan.org/pkg/subcaption>.
- [10] UK TeX Users Group. 2019. UK List of TeX Frequently Asked Questions. <https://texfaq.org>.
- [11] Boris Veytsman, Bern Schandl, Lee Netherton, and C. V. Radhakrishnan. 2005. *A package to create a nomenclature*. <http://www.ctan.org/pkg/nomencl>.