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 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 changes and adjust itself dynamically to keep approaching the target. As we consider which methods we should choose to use, we focus on their engineering costs and resulting effects. Currently, there are most advanced two proactive adaptive methods, PLA[7] and CobRA[2], both adopting ideas from model predictive control(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 followe the experimental senario of G.A and explore the reasons for the similarities and differences between the two on the basis of their work. We find that PLA needs to combine feedback to cope with uncertainties such as errors from modeling and prediction. 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

*Both authors contributed equally to this research.

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

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

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 figure1, 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 tactices after changes happened but do not predict what will happen in the future. On the contrary, proactive self-adaptation considers behaviors of changes in the future, to make tactices 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)[3] 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.

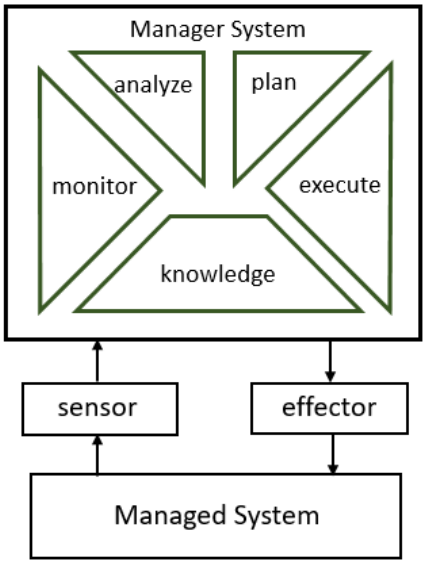

Figure 1: MAPE-K architecture

It uses a receding horizon to get 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 another 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 behavior of system and environment. The goal of the system is modeled into revenue function. Then we can use probabilistic model checkers[**?**] to exhaustive all pathes 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. Althogh 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 ADAPTATION SENARIO

The same as G.A, we carry out our comparison on the RUBiS system[11], an widely used open-source application for evaluation of self-adaption and cloud computing[9],[12]. 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 2: Architecture of RUBiS

three aspects: 1) the requirements that the system need to meet; 2) the uncertainties need to deal with while meeting the requirements; 3) the tactics that can be excuted to achieve self-adaptation control.

- `Requirements`: The functional requirements of RUBiS is the system shall response to every requests with mandatory and optional content. However, for adaptation controller, we only concern non-functional requirements of the system. The most important non-functional requirement of RUBiS 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. We capture non-functional requirements formally in a utility function that enables us to quantify their satisfaction. The utility function is modeled as follows:

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

  – utility associated with *cost* per time interval:

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

  – 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.

  Where *cost* is the cost of a server per unit of time and R_O, R_M is the revenue of a response to a request with the optional and only mandatory content respectively. We have $R_O > R_M > 0$.
- `Uncertainties`: Because of uncertainties while running, the non-functional requirements are difficult to meet. Thus we need self-adaptation control to cope with these uncertainties. The main uncertainties of RUBiS system resulting from two aspects:

  (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.
- `Tactics`: 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[10], 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 $[0, 1]$).

3 TWO PROACTIVE SELF-ADAPTAION METHODS: PLA AND COBRA

Both of the two approach are based on MPC[3] idea. To achieve proactive features, both of them consider an adaptation *horizon*, which is some periods in the future. Decisions are made not only based on historical circumstances and current measurements, but also on the predicted future. We discretizes the execution of adaptation decison in periods of τ, and the system and environment behavior in the adaptation horizon is presented by prediction models. 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.

3.1 PLA

The key idea of PLA is to let the adaptation decision underspecified through nondeterminism and construct a formal model for the adaptive system and then have the probabilistic model checker resolve the nondeterministic choices to find the way that maximize the accumulated utility over the horizon[7]. 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 future environment is be modeled as a DTMC in which the random variable representing the state of the environment has one change at each evaluation period τ. PLA uses an autoregresive (AR) time series predictor[**?**], such as RPS toolkit[**?**], to predict future environment, 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 3, the root e_0 of the tree is currunt environment, and e_1 is the estimation distribution of environment in next period with probabilities 0.185, 0.630, and 0.185, respectively. Unfortunately, there will be large

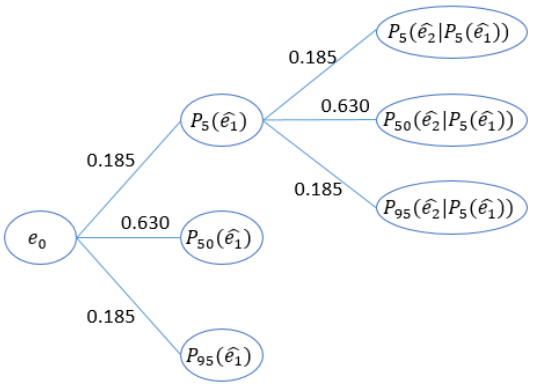

Figure 3: Probability Tree of Environment

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 utiity function. 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. Because of the round-robin strategy of RUBiS, we use LPS queuing theory model[**?**] to calculate the response time of the system. We can use a probabilistic model checker[**?**], such as PRISM[8] to analyze the model, verify the attributes, exhaust all paths and find a strategy to maximize the utility function.

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.

3.2 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 τ, 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.

$$\begin{cases} x(k+1) = A \cdot x(k) + B \cdot u(k) \\ y(k) = C \cdot x(k) + D \cdot a(k) \end{cases} \tag{4}$$

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, fot 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.

$$\begin{aligned} J_k = &\sum_{i=1}^{H} [y^o_{k+i} - y_{k+i}]^T Q_i [y^o_{k+i} - y_{k+i}] \\ &+ [\Delta u_{k+i-1}]^T P_i [\Delta u_{k+i-1}] \end{aligned} \tag{5}$$

where 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. y^o_{k+i} are setpoints of indicator i, which means the goal value of this indicator. The optimization problem can be definited

as follows:

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

CobRA makes adaptation decisions to keep indicators as close to setpoints as possible. The control scheme is represented as Figure 5

Figure 5: Control Scheme of CobRA

4 SIMILARITIES

Although there are great differences between the two methods in terms of design and implementation, they can achieve a reasonable control effect similarly. In the work of G.A[4], they compared the adaptive control performance and SASO property[**?**] of PLA and CobRA. We reproduce their implementation and achieve the control effects in their experiments, and get conclusions even further. In experiment, we followe G.A's parameter settings and two workloads of requests: WorldCup[**?**] and ClarkNet[6]

We show utilities under the two workloads in Figure 6. We get utility values are more than theirs results. In addition, compared to CobRA, PLA can achieve better control results. And the reason for that is both of the two methods heavily depend on the strength and rate of feedback, which we discuss later. Figure 4 show the control results of the two methods under the two workloads.

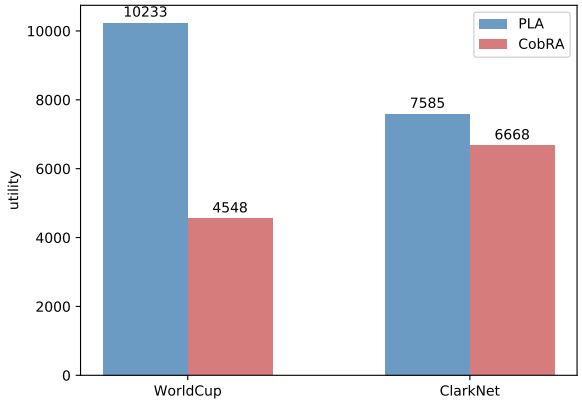

Figure 6: results of utility

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, similar to the description in CobRA. For PLA, its modeling for utility can be translated into a linear constrained optimization problem. Maximizing utility can be transformed into minimizing the minus of utility, u, which can be expressed as follows:

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

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, Δs 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. 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:

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

where

$$r' = k_1 * (s_0 + \Delta s) + k_2 * (d_0 + \Delta d) + k_3 * a, \tag{9}$$

is the expected response time after adjusting control parameters. 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.

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 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.

We can get the conclusion that under the assumption that they are excuting in same initial state, if CobRA's weight matrix meets those conditions, they can get same control results. However, in the running of the system, this assumption is difficult to be satisfied, the control effect is mainly depend on feedback. Although 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. Open-loop methods can only handle disturbance in measurements and modeling. 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. 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 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.

We compare the self-adaptation effects of PLA with feedback and without feedback. Figure 7 shows the results of no feedback under two workloads. We can see that there are several period the response time is 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. 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. The reasons that PLA and CobRA can get similar control result have two aspects: 1) both of the adopt ideas and structures of MPC, 2) they heavily depend on feedback to cope with modeling errors and disturbance.

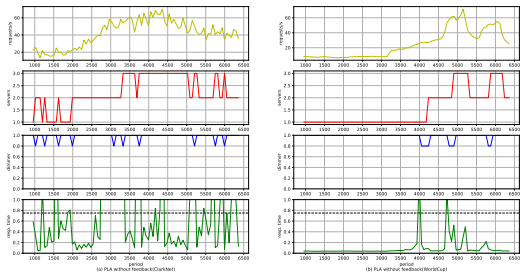

Figure 7: PLA without feedback

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. We mainly discuss three significant differences between them and considering how these differences affect the control effect. We talk about 1)the difference in modeling asymmetric delay of target system, 2)the difference of environment prediction to discuss how much the prediction plays a role, and 3)the constraints of regular terms.

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 source of uncertainty and the target problem of the adaptive control. At the beginning of every decision period, PLA uses current and historical environment to predict the future environment in the horizon to construct the probabilistc tree of environment. Then PLA calculates the future response time according to the model of the whole system and environment, so as to make adaptation decisions. However, CobRA regards the environment as one of the disturbances of the system and does not predict the future environment, but as a direct component adding to the state space function of the system. Kalman Filter is used to achieve smoothing and 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. But 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.

Theoretically, both of them have weaknesses: feedback control is always one-step slower than the current state while simple prediction can not tolerant unmeasured disturbance, nevertheless. In view of both of them can achieve similar effects, PLA needs feedback and prediction whereas CobRA only has feedback, we want to explore the effectiveness of prediction: whether it is necessary to use prediction in the presence of feedback. We design a variant of PLA: npPLA,

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

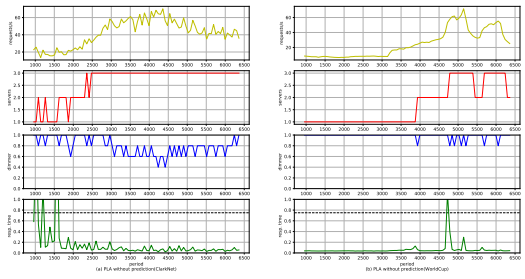

Figure 8: PLA without prediction for environment

which does not predict the future workload and just use feedback to cope with changing environment, the same as CobRA. We compared the control results of PLA and npPLA under the two workloads. The utility of PLA without prediction is 10326 for WorldCup and 6668 for ClarkNet, even a little more than the utility with prediction. Figure 8 shows the concrete results. 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 try to tune a pretty good result. In fact, prediction is not necessary for PLA, because the error of prediction still needs to be compensated by feedback. When the environment changes regularly and easy to predict, using predictions in PLA can get better control effect. Prediction can react to changes ahead of time and produce effective results quickly. However, feedback control does not possess foreseeability and works slowly. In addition, there are errors in feedback control certainly because feedback control only try to adjust the system after errors occur.

We are also considering whether it would be better to introduce environmental prediction to CobRA. Whereas M-PC can deal with inaccuracy to some extent and it has the precondition that environment is stable and changes slowly so that can be regarded as disturbance. Feedack is sufficient to deal with the stable environment. In conclusion, when the environment changes regularly and suitable to predict, prediction can offset the deficiency of just with feedback to some extent. In view of CobRA's assumption to the environment: changing slowly, it is superfluous to predict future environment for CobRA.

5.2 Asymmetrical 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 asymmetrical latency with a control parameter. PLA models each decision and the asymmetrical latency is well described by the MDP model. However, in CobRA, asymmetrical latency 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:

$$r = k_1 * s + k_2 * d + k_3 * a \tag{10}$$

To translate this into a state space model, we use the state matrix A to reflect the latency of adding 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:

$$r(k) = k_1 * s(k-2) + k_2 * d(k-1) + k_3 * a(k-1) \tag{11}$$

Under this model, it can be found that at time k, the number of servers used is the measured value at time $k-2$, also 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. cobraservercobraserverservercobradimmer 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 remove a server ahead of time and 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

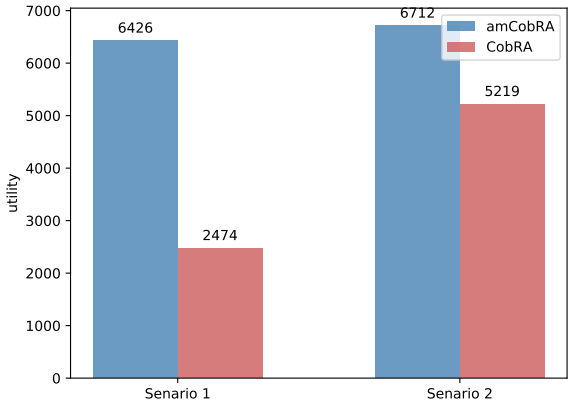

Figure 10: Utility of CobRA and amCobRA

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. 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. We construct a new CobRA controller with asymmetrical latency named amCobRA. We compare CobRA and amCobRA under two senario, where the two controller can make tactics to remove server. Figure 9 shows the results and Figure 10 shows the utility values. We can see that, in both senario, amCobRA can get better control results than CobRA.

5.3 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 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

Based on the work of G.A[4], we analyzed and compared the similarities and differences between PLA and CobRA. G.A proposed that the two could achieve similar control effects. We explored the reasons and found that the PLA also needs feedback to deal with errors in prediction and modeling. 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, amsproc, and amsbook document classes*. American Mathematical Society. http://www.ctan.org/pkg/amslatex.
[5] Michael Downes and Barbara Beeton. 2004. *The amsart, amsproc, and amsbook document classes*. American Mathematical Society. http://www.ctan.org/pkg/amslatex.
[6] Michael Downes and Barbara Beeton. 2004. *The amsart, amsproc, 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] Carsten Heinz, Brooks Moses, and Jobst Hoffmann. 2015. *The Listings Package*. http://www.ctan.org/pkg/listings.
[9] Axel Sommerfeldt. 2013. *The subcaption package*. http://www.ctan.org/pkg/subcaption.
[10] Nicola L. C. Talbot. 2016. *User Manual for glossaries.sty v4.25*. http://www.ctan.org/pkg/subcaption.

[11] UK TEX Users Group. 2019. UK List of TEX Frequently Asked Questions. https://texfaq.org.

[12] Boris Veytsman, Bern Schandl, Lee Netherton, and C. V. Radhakrishnan. 2005. *A package to create a nomenclature*. http://www.ctan.org/pkg/nomencl.

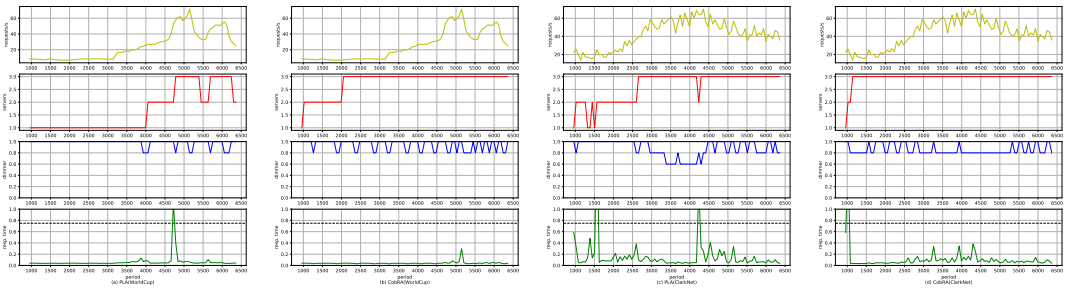

Figure 4: results of PLA and CobRA

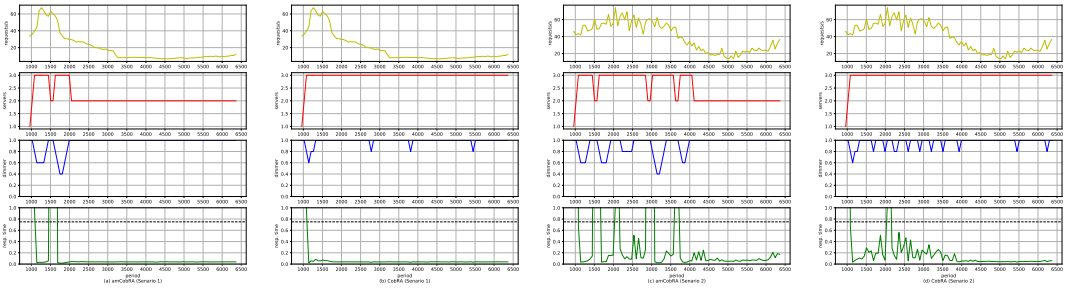

Figure 9: Control results of CobRA and amCobRA