GMoCA: Green Mobile Cloud Applications

Yan Gu, Verdi March, Bu Sung Lee

HP Laboratories
HPL-2012-79R1

Keyword(s):

Mobile computing; Cloud computing; Energy; Cost function;

Abstract:

Mobile cloud computing enables numerous associated mobile users to access the abundant cloud computing resources, thereby complements the resource constrain of mobile devices. A fundamental issue in the mobile application platform is to make deployment decision for individual tasks when the battery life of the mobile device is a major concern for the mobile user's experience. We propose a deployment scheme to offload expensive computational tasks from thin, mobile devices to powered, powerful devices on the cloud so that we could prolong battery life for mobile devices, meanwhile provide rich user experiences for such mobile applications. We envision that the scheme can be extended to other type of smart devices such as smart printers, smart TVs or sensors where offloading tasks is required to trade-off between performance and battery life.

External Posting Date: June 6, 2012 [Fulltext] Approved for External Publication
Internal Posting Date: June 6, 2012 [Fulltext]

GMoCA: Green Mobile Cloud Applications

Yan Gu
Services Platform Lab
Hewlett-Packard Laboratories
Singapore
chloe.yan.gu@hp.com

Verdi March
Services Platform Lab
Hewlett-Packard Laboratories
Singapore
verdi.march@hp.com

Bu Sung Lee
Services Platform Lab
Hewlett-Packard Laboratories
Singapore
francis.lee@hp.com

***Abstract*—Mobile cloud computing enables numerous associated mobile users to access the abundant cloud computing resources, thereby complements the resource constrain of mobile devices. A fundamental issue in the mobile application platform is to make deployment decision for individual tasks when the battery life of the mobile device is a major concern for the mobile user's experience. We propose a deployment scheme to offload expensive computational tasks from thin, mobile devices to powered, powerful devices on the cloud so that we could prolong battery life for mobile devices, meanwhile provide rich user experiences for such mobile applications. We envision that the scheme can be extended to other type of smart devices such as smart printers, smart TVs or sensors where offloading tasks is required to trade-off between performance and battery life.**

***Keywords*-Mobile computing; Cloud computing; Energy; Cost function;**

I. Introduction

Advances in mobile hardware and software techniques have enabled mobile phones to support rich multimedia capability such as image recognition, video/audio processing, natural language processing, etc. As estimated by Gartner, mobile devices are going to overtake the PC sales with more than 40 million units by 2013 [1]. According to the various market review and forecast, the landscape of ubiquitous smart devices such as smart printers, smart TVs or sensors is expanding even rapidly. A critical challenge of rich mobile applications is to maximize both the performance *and* battery life on executing resource-intensive tasks. This can potentially be addressed by complementing resource-starved mobile devices with cloud resources, whereby compute-intensive tasks in mobile devices are shifted to the cloud. The technologies for replicating and migrating execution among connected computing substrates, including virtual machine migration and incremental checkpoints, have matured. An intuitive solution for executing mobile applications is to shift all computationally expensive workload to the cloud and leave the relatively cheap tasks with mobile client, to guarantee the performance of application. However, task offloading incurs overhead of data transmission across mobile and the cloud in terms of time and power consumption on mobile devices [2]. Hence, the aggressiveness of offloading needs to be calibrated to prevent diminished (or even degraded) performance and power saving.

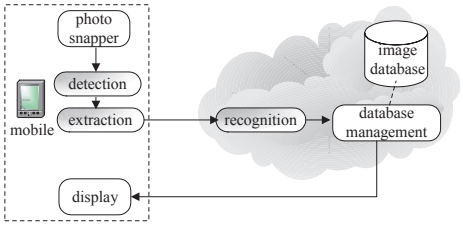


Figure 1. *extraction* component on mobile device.

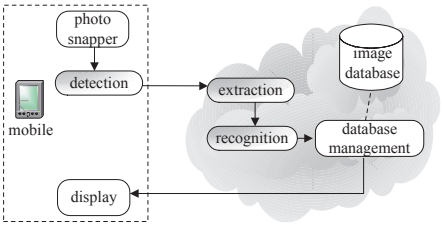


Figure 2. *extraction* component on the cloud.

In this paper, we propose GMoCA (**G**reen **Mo**bile **C**loud **A**pplication), a power-performance optimization scheme based on control theory for our existing μCloud platform [3]. A μCloud application is composed of *mobile*, *cloud* and *hybrid* components. Hybrid components can be deployed to the cloud or mobile devices. Figure 1 and 2 illustrate an example of face recognition application which takes a photo using the camera of a mobile phone, automatically detects the face in the captured photo and then retrieves the profile information associated with the detected face. Figure 1 shows the *hybrid* component - *extraction* is executed on mobile device, while it is migrated to the cloud as shown in Figure 2. Task migration attempts to opportunistically offload hybrid components from mobile devices to the cloud to reduce both the execution time and mobile power consumption. To ensure that the task migration is effective, we propose a control theory-based model to estimate the potential performance gain and power reduction due to task migration.

The rest of this paper is organized as follows. The next section discusses some related work for partitioning of mobile applications and deployment of application to the cloud. Section III describes the power-efficient scheme for task offloading. In Section IV, we evaluate the performance of our proposed scheme and finally we conclude in Section V by outlining some directions for future work.

II. Related Work

There are few existing work on dynamic offloading [4], [5], [6]. Liu et al. proposed a heuristic searching algorithm to determine the destination host for the application migration in [5]. In their migration scheduling engine, they employed the feature of live migration for Xen [7] virtual machine.

CloneCloud [4] automatically offloads part of the execution of mobile applications from mobile devices to the cloud. The right partition of a mobile application to be migrated to the cloud is determined by performing on-line static and dynamic profiling. They used a randomly chosen set of input data for the application to generate the workload profile. Each application is executed once on the mobile device and once on the cloud to generate the workload. However, the set of input could not represent the whole input space. It is hard to extrapolate the workload on cross-platform scenarios. As such, the scalability of such profiling is unclear.

MAUI [6] presents a platform to minimize power consumption of mobile devices by offloading tasks to the cloud. It requires the engineers' effort to analyze the code of application offline and annotate the methods for remote executions manually. In runtime, their profiler monitoring the resource predicts whether the method should be offloaded to the cloud. In the end, the platform reintegrates the results back into the mobile device. While our work provides a control theory-based prediction model, which generates much better results than history-based approach.

Usually, the control-theory based approaches adjust the system performance by periodically feeding the output back to the system. Such models are widely used to estimate the application workload, such as video decoding [8], [9], [10], [11], [12], [13], [14] and game applications [15].

Lu et al. presented a feedback-control scheduling algorithm for dynamic voltage scaling (DVS) in [8], [9], with the decoding application as as $M/M/1$ queue system. Wu et al. modelled multiple-clock-domain processors as queue system, in which queue occupancies are used to trigger DVS algorithm [10], [11].

Gu et al. is the first time to introduce the feedback control mechanism in the context of interactive games [15].

In our work, we bring in control theory to provide better estimation for hybrid components in the task offloading. To our knowledge, no such scheme has been proposed targeted towards mobile cloud applications.

III. Power-Efficient Component Offloading

GMoCA is our proposed scheme for dynamically offloading hybrid components that are computational intensive from mobile devices to the cloud.

Given the illustrated example of task migration in Figure 1 and Figure 2, the hybrid component - *extraction* is denoted as component i, the mobile component - *detection* is denoted as component h and the cloud component - *recognition* is denoted as component j in this section.

A. Objective Functions

To ensure that component migration yields significant performance improvement and mobile power reduction, we define an objective function that must be met in order for a particular component to be offloaded to the cloud. Let E_i and P_i denote the execution time and power consumption of component i on a mobile device, respectively. In addition, let E_i' and P_i' denote the execution time and power consumption of component i on the cloud. A task migration is effective, i.e., yields performance gain and power reduction, if objective function $(E_i' < E_i) \wedge (P_i' < P_i)$ is satisfied, where $\wedge$ denotes the logical AND. In practise, we could relax the objective function and further define two metrics to determine the component offloading.

- *Power Scalability* is to provide the minimal power consumption, given a fixed quality of application. In this case, we would migrate the component i to the cloud when the follow equations are satisfied.

$$\begin{cases} E_i' & \leq e_i \\ P_i' & < P_i \end{cases} \tag{1}$$

  where e_i is the threshold execution time for the component i.
- *Power-bounded Scalability* is to provide the maximum quality of application, given a fixed power consumption. In this case, we would migrate the component i to the cloud when the follow conditions are met.

$$\begin{cases} E_i' & < E_i \\ P_i' & \leq p_i \end{cases} \tag{2}$$

  p_i is the threshold power for the component i.

B. Cost Functions

In order to determine which components and when the components should be offloaded to the cloud, given the resource availability, e.g. mobile processor and network bandwidth, we formulate the problem by cost functions, detailed as follow.

$$E_i = \frac{D_{hi}}{B^m} + \frac{C_i}{F^m} + (\frac{D_{ij}}{B^{mc}} + L_{ij}^{mc}) \quad (3)$$

$$E_i' = (\frac{D_{hi}}{B^{mc}} + L_{hi}^{mc}) + \frac{C_i}{F^c} + \frac{D_{ij}}{B^c} \quad (4)$$

$$P_i = P_i + P_{ij} \quad (5)$$

$$P_i' = P_{hi} \quad (6)$$

where

- F^c and F^m denote the processor frequencies on the cloud and mobile device, respectively. As we know, the mobile processor is enabled with scalable frequency, we define F^{max} as the fixed maximal frequency of the processor. While F^c is a fixed frequency for a cloud processor.
- B^c denotes the I/O bandwidth within the cloud, B^m denotes the I/O bandwidth within a mobile device and B^{mc} denotes the available network bandwidth between a mobile device and the cloud.
- L^{mc} denotes the network latency from a mobile device to the cloud. The size of data affects the latency. Here we assume the latency among mobile devices and the latency among cloud servers are negligible.
- P_{ij} denotes the power required to transmit data from component i to component j.
- P_i denotes the power required to execute component i on a mobile device.
- D_{ij} denotes the data volume transmitted from component i to component j.
- C_i denotes the number of processor cycles required to execute component i.

Here, we assume I/O speeds of mobile and cloud server, i.e., B^m and B^c are constants. As we host the components on the *components repository*, whenever it is beneficial to migrate a component from a mobile device to a cloud server, the same component will be directly retrieved from repository to the cloud server. By this manner, we avoid the migration overhead for the component source codes. D_{hi}, D_{ij}, C_i, P_i, P_{ij}, P_{hi} are predicted by using our designed predictor, which are illustrated in section III-C.

C. Predictor

To predict the gain of migrating component i, in terms of performance improvement and power reduction, we need to derive E_i, P_i, E_i' and P_i' prior to the execution of i is started. As such, we propose a general proportional-integral-derivative (PID) controller (Equation 9) to predict the following parameters: D_{hi}, D_{ij}, C_i, P_i, P_{ij} and P_{hi}.

1) PID Controller Fundamentals: We describe a general proportional-integral-derivative (PID) controller for close-loop systems with an emphasis on the aspects that are important for understanding our scheme.

Feedback control is one of the fundamental mechanisms for dynamic systems to achieve equilibrium. In a feedback system, some variables – *measured process variables*, are monitored and measured by the feedback controller and compared to their desired values – *set points*. The differences (*errors*) between the controlled variable and the set point are fed back to the controller repeatedly. Corresponding system states are usually adjusted according to the differences to let the system variables approximate the set points as closely as possible.

A PID controller is a generic control loop feedback mechanism that is used to adjust system parameters based on the feedback from the recent error between a *measured process variable* and a desired *set point*. The measured variable usually reaches its set point and stabilizes within a short period. It involves three separate components – *Proportional Control*, *Integral Control* and *Derivative Control*. The proportional control determines the speed of the system in reacting to errors. The integral control is used to determine the accuracy of the system based on recent errors. Finally, the derivative control determines the system reaction based on the rate at which the error changes.

The PID feedback controller can be described in three major forms: the ideal form, the discrete form and the parallel form. Although the discrete form is often used in digital algorithms to keep tuning similar to electronic controllers, the parallel form is the simplest one where each control element is given the same error input in parallel. The output of the controller is given by

$$Output(t) = R_{contrib} + G_{contrib} + V_{contrib} \quad (7)$$

where $R_{contrib}$, $G_{contrib}$ and $V_{contrib}$ are the *proportional*, *integral* and *derivative* contributors of the PID controller.

$$\begin{cases} R_{contrib} &= K_R \cdot \epsilon(t) \\ G_{contrib} &= \frac{1}{K_G} \cdot \int \epsilon(t)dt \\ V_{contrib} &= K_V \cdot \frac{d\epsilon(t)}{dt} \end{cases} \quad (8)$$

where $\epsilon(t)$ is defined as the difference between a measured variable and a desired set point. K_R, K_G and K_V are the constants, which are defined as *proportional gain*, *integral gain* and *derivative gain* respectively.

By tuning the values of these gains, the PID controller can provide individualized control specific to process requirements involving error responsiveness, overshoot of the set point and system oscillation.

A high proportional gain results in a large change in the controller's output, given a changed system error. In contrast, a small proportional gain results in a small system response to a large system error, i.e., a less sensitive controller.

The integral contributor accelerates the movement of process towards set point and eliminates the residual steady-state error that occurs with a proportional-alone controller. However, since the integral contributor is responding to accumulated errors from the past, it can cause the present value to overshoot the set point (i.e. cross over the set point and then create a deviation in the other direction).

The derivative controller slows the rate at which the system output changes. Hence, it is used to reduce the magnitude of the overshoot produced by the integral contributor and improve the controller stability. However, differentiation of an input error amplifies noise in the error, and thus the derivative controller is highly sensitive to noise in the error and can cause a process to become unstable, if the noise and the derivative gain are sufficiently large.

2) PID Controller Design:

$$\begin{cases} \Delta\bar{\omega}(t) = K_R \cdot \epsilon(t) + \frac{1}{K_G} \cdot \sum_{G_c} \epsilon(t) + K_V \cdot \frac{\epsilon(t)-\epsilon(t-V_c)}{V_c} \\ \bar{\omega}(t+1) = \bar{\omega}(t) + \Delta\bar{\omega}(t) \end{cases} \tag{9}$$

Using the PID controller, each of the above-mentioned six parameters is estimated as the sum of the previously predicted value (i.e., $\bar{\omega}(t)$) and the controller feedback (i.e., $\Delta\bar{\omega}(t)$). The feedback is derived from the prediction errors of previous predictions. The error is periodically measured by the PID controller and is given by $\epsilon(t) = \omega(t) - \bar{\omega}(t)$, where t denotes the sample index, $\omega(t)$ denotes the actual value and $\bar{\omega}(t)$ denotes the predicted value. K_R, K_G and K_V are the proportional, integral and derivative coefficients, respectively. G_c and V_c are the tunable parameters of the controller. For example, G_c is set to 2, i.e., $\sum_{G_c} \epsilon(t) = \epsilon(t) + \epsilon(t-1) + \epsilon(t-2)$. We further define $\epsilon(t - V_c)$ as the prediction error at sample $t - V_c$. The output $\Delta\bar{\omega}(t)$ is fed back to the controller and regulates the next estimated value $\bar{\omega}(t+1)$ of the component.

IV. Preliminary Experiment

We implemented our proposed prediction schemes on HTC Nexus One mobile device. It runs on Android 2.3.4. The mobile device is running on ARM® processor which supports the frequency scaling with as many as 12 different levels. The preliminary experiments are conducted on an object feature detection component. In this section, we evaluate the performance of our predictor (Equation 9) with two parameters in the cost functions, C_i and P_i respectively. In order to measure the component execution time (C_i) on the mobile device, we fixed the frequency to the maximum $F^{max} = 998\ MHz$ by writing the system files and invoked Android clock functions from *SystemClock* class to collect the system time. The power consumption (P_i) to execute a component is measured by using PowerTutor [16]. As we fix the processor frequency to the maximum (F^{max}), the CPU utilization has linear correlation to the power consumption. Therefore, in the paper, the resultant CPU utilization is compared to indict the power consumption in the figures.

To illustrate the performance of the predictor (Equation 9), we define two metrics. (i) the *absolute prediction error* which is defined as the absolute difference in processor cycles or CPU utilization between the actual and predicted values, and (ii) the relative prediction error which is defined as the ratio (%) between the absolute prediction error and the actual value.

To compare the performance of PID predictor with other history-based predictors, we introduce another predictor based on an auto-regressive (AR) model [17]. AR model is a well-known approach in time series systems to predict an output of a system based on history outputs. An auto-regressive model $\omega(t)$ is given by a weighted sum of ρ previous values, where ρ is the order of the model. The model is given by $\bar{\omega}(t+1) = c + \sum_{\rho} \varphi(t)\omega(t) + \epsilon(t)$, where $\varphi(t)$, $\varphi(t-1)$, ..., $\varphi(t-\rho)$ are the parameters of the model and the summation of those parameters equals to 1. $\epsilon(t)$ is white noise and it is defined as prediction error in our scheme, i.e. $\epsilon(t) = \omega(t) - \bar{\omega}(t)$, where t is the index of sample. c is a constant.

Selecting an order for the model, in this context, is similar to that of selecting the sample size. As the system becomes more complex, we expect the value of ρ to steadily increase. However, an excessively large value of ρ will result in overfitting. This trend can be seen in Figure 3(a) and 3(b), which shows the accuracy of predicting changes significantly with different values of ρ. In our experiment, we select $\rho = 2$ for workload and CPU utilization prediction.

A. Tuning PID Parameters

In our experiments, we observe that the selection of values impacts the prediction results significantly. By manually tuning the parameters, we obtain the best prediction when $K_R = 0.5$, $K_G = 28$, $K_V = 0.001$ and chose $G_c = 2$ and $V_c = 1$ for the prediction of workload as shown in Figure 4(a). It compares the prediction errors for workload with the above tested values for K_R, respectively. Note that the PID model produces the best prediction when $K_R = 0.5$. The absolute prediction error is as high as 103121 cycles when K_R is set to 0.1, while its error drops to 49091 cycles with $K_R = 0.5$. Similarly, the relative prediction error drops significantly when K_R is set to 0.5.

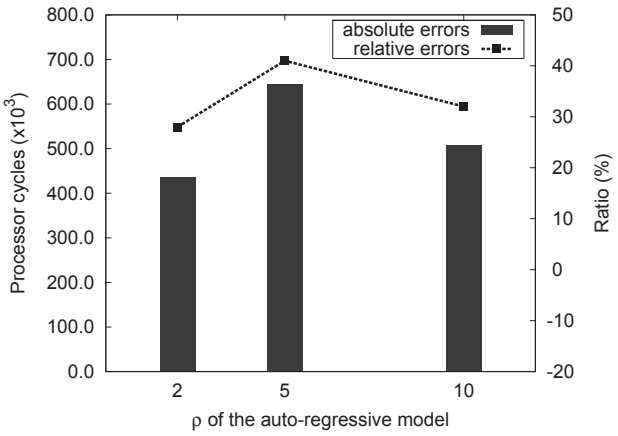
While for the prediction of CPU utilization, we obtain the best prediction when $K_R = 0.9$, shown in Figure 4(b). When K_R is set to 0.9, the absolute error drops to 5.9 and the relative error is decreased to 2.4 %.

Due to the limited space, we do not elaborate the comparison with different values for each parameter.

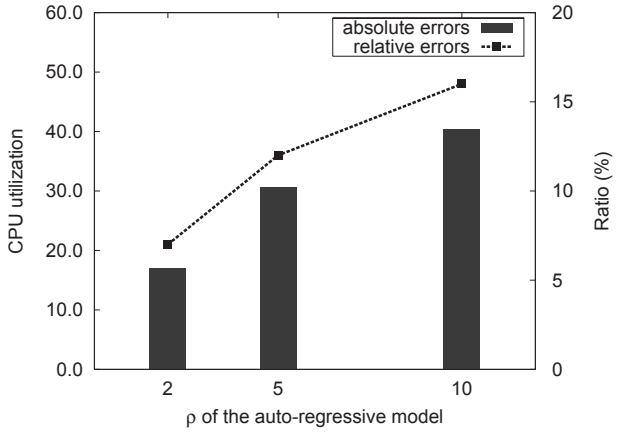
B. Results

The overhead of our proposed PID predictor and the AR model is negligible as both of them are based on linear operations.

Our initial results show that the PID model outperforms over AR model. In order to eliminate the effects in transient state when we start the prediction, we plot the results from index 5 to 100 when the prediction reaches the steady state. Figure 5 shows the comparison of workload prediction with the best setting for AR and PID models, i.e., $rho = 2$ and

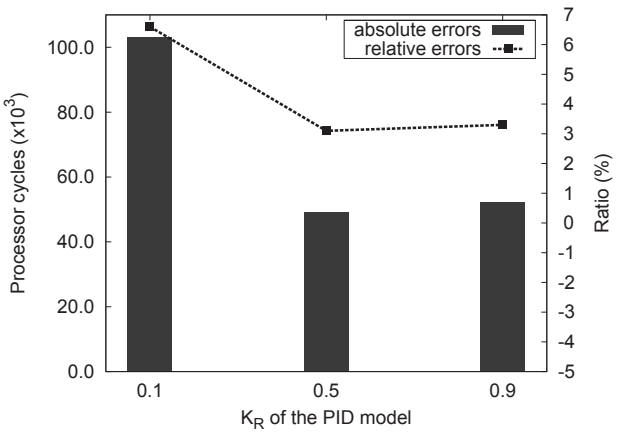


(a) Workload.

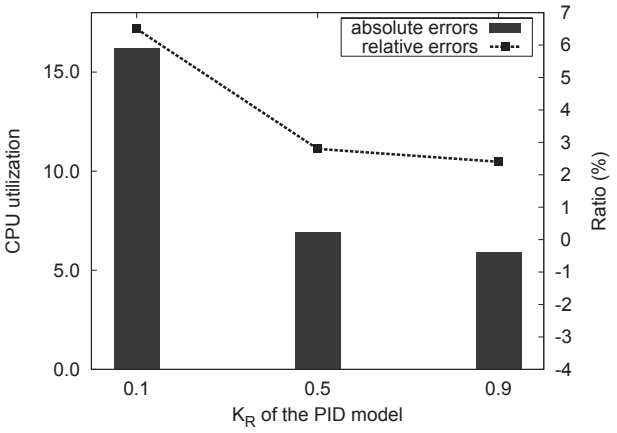


(b) CPU utilization.

Figure 3. Average absolute and relative prediction errors with different ρ in AR model.



(a) Workload.



(b) CPU utilization.

Figure 4. Average absolute and relative prediction errors with different K_R in PID model.

$K_R = 0.5$. While Figure 6 shows the prediction of CPU utilization with the best setting for AR and PID models, i.e., $rho = 2$ and $K_R = 0.9$. Both figures show that the predicted values with PID model match the real values much more closely than AR model.

The same predictor could be applied to estimate the other parameters in our cost functions and achieve good results.

V. Concluding Remarks

In the paper, we have implemented prediction method and conducted preliminary experiments on the Android devices for individual components. The results show that the control theory-based method gives better results comparing with history-based AR model, for workload and power consumption. Those results show that our proposed scheme could give much better estimation for other parameters in our cost functions. We are currently building the migration framework in which the proposed objective functions are examined to selectively offload the components across mobile device and the cloud.

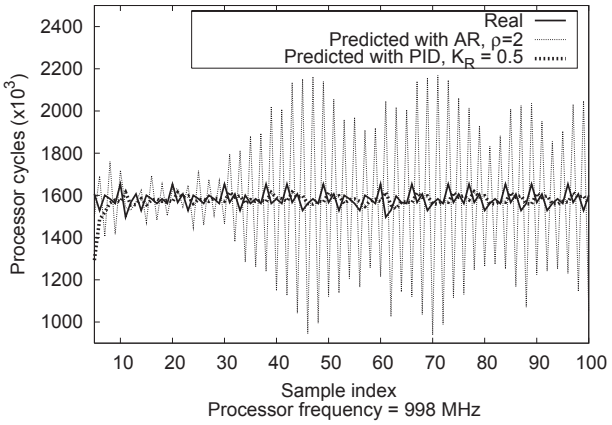


Figure 5. Prediction of workload with AR and PID models.

Moreover, although GMoCA is originated to assist the deployment of hybrid components, it is easily decoupled and extended to other scenarios where offloading tasks are

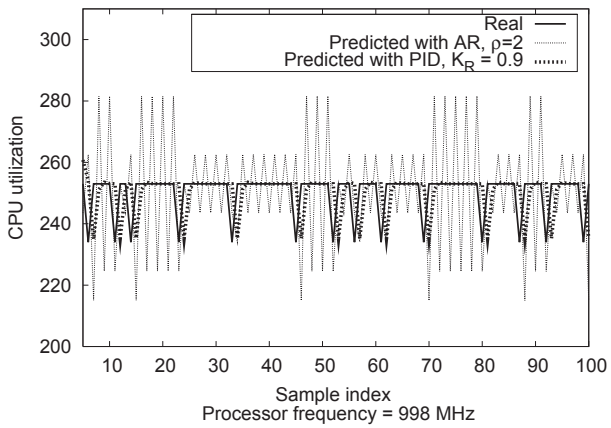


Figure 6. Prediction of CPU utilization with AR and PID models.

needed across smart devices, e.g. smart printers, smart TVs or sensors.

REFERENCES

[1] M. Walsh, "Gartner: Mobile to outpace desktop web by 2013," 13 January, 2010.

[2] D. Huang, "Mobile cloud computing," *IEEE COMSOC Multimedia Communications Technical Committee (MMTC) E-Letter*, 2011.

[3] V. March, Y. Gu, E. Leonardi, G. Goh, M. Kirchberg, and B. S. Lee, "uCloud: Towards a New Paradigm of rich mobile applications," in *Proceedings of the 8th International Conference of Mobile Web Information Systems (MobiWIS)*, 2011.

[4] B. Chun and P. Maniatis, "Augmented smart phone applications through clone cloud execution," in *Proceedings of the 12th Workshop on Hot Topics in Operating Systems (HotOS)*, 2009.

[5] L. Liu, H. Wang, X. Liu, X. Jin, W. B. He, Q. B. Wang, and Y. Chen, "Greencloud: a new architecture for green data center," in *Proceedings of the 6th International Conference Industry Session on Autonomic Computing and Communications Industry Session (ICAC-INDST)*, 2009.

[6] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2010.

[7] "Xen user manual http://bits.xensource.com/xen/docs/user.pdf."

[8] Z. Lu, J. Hein, M. Humphrey, M. Stan, J. Lach, and K. Skadron, "Control-theoretic dynamic frequency and voltage scaling for multimedia workloads," in *Proc. 2002 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*. Greenoble, France: ACM Press, October 2002, pp. 156 – 163.

[9] Z. Lu, J. Lach, M. R. Stan, and K. Skadron, "Reducing multimedia decode power using feedback control," in *Proc. 2003 International Conference on Computer Design (ICCD)*. San Jose, CA, USA: IEEE Press, October 2003, pp. 489–496.

[10] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark, "Formal online methods for voltage/frequency control in multiple clock domain microprocessors," in *Proc. 2004 International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Boston, MA, USA: ACM Press, October 2004, pp. 248 – 259.

[11] Q. Wu, P. Juang, M. Martonosi, L.-S. Peh, and D. W. Clark, "Formal control techniques for power-performance management," *IEEE Micro*, vol. 25, no. 5, pp. 52–62, 2005.

[12] J. A. Stankovic, C. Lu, S. H. Son, and G. Tao, "The case for feedback control real-time scheduling," in *Proc. 1999 Euromicro Conference on Real-Time Systems (ECRTS)*. University of York, York, UK: IEEE Press, June 1999, pp. 11 – 20.

[13] A. Varma, B. Ganesh, M. Sen, S. R. Choudhury, L. Srinivasan, and J. Bruce, "A control-theoretic approach to dynamic voltage scheduling," in *Proc. 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*. San Jose, California, USA: ACM Press, October 2003, pp. 255 – 266.

[14] Y. Zhu and F. Mueller, "Feedback EDF scheduling exploiting dynamic voltage scaling," in *Proc. 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. Toronto, Canada: IEEE Press, May 2004, pp. 84– 93.

[15] Y. Gu and S. Chakraborty, "Control theory-based DVS for interactive 3D games," in *Proc. 2008 Design Automation Conference (DAC)*, Anaheim, CA, USA, June 2008.

[16] "Powertutor http://ziyang.eecs.umich.edu/projects/powertutor/."

[17] G. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis: Forecasting and Control*, 3rd ed. Prentice Hall, 1994.