

Online Adaptive Utilization Control for Real-Time Embedded Multiprocessor Systems

Jianguo Yao and Xue Liu
School of Computer Science,
McGill University,
Montreal, QC H3A2A7, CANADA
(jianguo,xueliu)@cs.mcgill.ca

Mingxuan Yuan, Zonghua Gu
Department of Computer Science and Engineering
Hong Kong University of Science and Technology,
P.R.CHINA
(csyuan, zgu)@cse.ust.hk

ABSTRACT

To provide Quality of Service (QoS) guarantees in open and unpredictable environments, the utilization control problem is defined to keep the processor utilization at the schedulable utilization bound, even in the face of unpredictable and/or varying task execution times. To handle the end-to-end task model where each task is comprised of a chain of subtasks distributed on multiprocessors, researchers have used Model Predictive Control (MPC) to address the Multiple-Input, Multiple-Output (MIMO) control problem. Although MPC can handle a limited range of model uncertainties due to execution time estimation errors, the system may suffer performance deterioration or even become unstable if the actual task execution times are much larger than their estimated values. In this paper, we present an online adaptive optimal control approach using Recursive Least Squares (RLS) based model estimator plus Linear Quadratic (LQ) optimal controller. We use simulation experiments to demonstrate the effectiveness of our controller compared with the MPC-based controller.

Categories and Subject Descriptors

C.4 [Computer Systems Organization]: Performance of Systems
--- Performance attributes

General Terms

Management, Performance.

Keywords

Feedback control, multiprocessors, real-time scheduling.

1. INTRODUCTION AND RELATED WORK

Traditional hard real-time systems execute in a closed environment and rely on worst-case analysis to make offline guarantees, i.e., by scheduling real-time tasks based on conservative estimations of the worst-case execution time of each task on a given hardware platform [1], [2]. But in recent years, there is a growing collection of real-time embedded systems that execute in an open and unpredictable environment yet still must make performance

guarantees, where task execution time may suffer large uncertainties due to input parameter variations or unpredictable environment. A key challenge is to provide real-time guarantees even though the workload cannot be accurately characterized *a priori*. The traditional approach of worst-case analysis for making real-time guarantees may cause under-utilization and wasted resources, since a task's WCET is often much bigger than its average execution time. We need more dynamic and adaptive scheduling algorithms that can adjust scheduling parameters according to runtime variations of task execution time while still making real-time guarantees. Feedback control is an effective technique to tackle these challenges [3], [4], [5].

To guarantee that the system is schedulable, that is, no deadline miss occurs, it is sufficient to keep each processor's utilization to be under the *schedulable utilization bound*. For example, if the scheduling algorithm is rate monotonic scheduling, where each task is assigned a fixed priority and a task with higher execution rate is assigned a higher priority, then the system is schedulable if the processor utilization does not exceed $m(2^{1/m} - 1)$, where m is the number of tasks on the same processor [6]. (This is often called the *Liu and Layland bound*.) If the scheduling algorithm is Earliest Deadline First (EDF), then the schedulable utilization bound is 1. Recently, researchers have applied feedback control techniques to keep processor utilization to be under the schedulable utilization bound through dynamic resource allocation in response to workload variations, in order to achieve high processor utilization while still meeting real-time constraints.

Lin et al applied feedback control techniques to real-time scheduling in the face of uncertain workloads in [4]. Goel et al proposed a feedback-based scheduling approach to meet real-time requirement [3]. Stankovic et al [5] applied feedback control to distributed real-time systems. Lu et al [7] presented a survey of feedback control techniques to ensure real-time guarantee under unknown execution time. Yong et al [8] used a distributed utilization feedback controller to handle system dynamics caused by load balancing for large-scale server clusters. Recent work on utilization control based on Model Predictive Control (MPC) was capable of handling execution time variations within a certain range [9], [10], [11]. Chen et al [12] presented a Multi-Parametric Rate Adaptation (MPRA) algorithm for discrete rate adaptation in distributed real-time systems with end-to-end tasks, where an adjustment tree is computed offline using mp-MILP (Multi-Parametric 0-1 Mixed Integer Linear Programming). When system variation happens at runtime, the controller searches this tree based on the variation to make online adjustments. All the above approaches based on feedback control require having an accurate model of the real-time system, which may be difficult to obtain for realistic systems.

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. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CODES+ISSS'08, October 19–24, 2008, Atlanta, Georgia, USA.

Copyright 2008 ACM 978-1-60558-470-6/08/10...\$5.00.

Some authors have designed centralized or decentralized MPC-based controllers to solve the processor utilization control problem [9], [10], [11] based on approximate models of the underlying systems and the tasks running on top of them. It was shown [9] that these approaches can perform well within a limited range of model uncertainties in terms of execution time estimation errors and/or variations. However, the system may suffer performance degradation or even become unstable if the task execution times are underestimated by a significant degree, i.e., the actual task execution times are much larger than their estimated values. (On the other hand, overestimation of task execution times may cause slow convergence to the set point, but will not cause instability.) In this paper, we present a CPU utilization control technique using Recursive Least Squares (RLS) based model estimator and Linear Quadratic (LQ) optimal controller [17], which is a self-tuning adaptive control algorithm without any need for prior knowledge of the system model. The system model is learned (estimated) and automatically adjusted online with the RLS method, and the control inputs are calculated at runtime by optimizing a quadratic cost function. Due to online model estimation, this control algorithm has good performance even in the face of large system model uncertainties.

The feedback control principle has found many applications in Systems-On-Chip (SoC). Lipsa et al [13] proposed a design methodology and framework for Autonomic Systems-On-Chip (ASoC), where an autonomic layer is added to monitor and control the functional layer to achieve self-calibration, fault tolerance or self-healing. Particularly relevant to this paper is Multiprocessor System-on-Chip (MPSoC), where multiple Processing Elements (PEs) are placed on a single chip and connected with a bus or network-on-chip. Each PE may be a dedicated ASIC or a programmable processor running several tasks using a real-time scheduling algorithm. Lankes et al [14] presented an extension to the SystemC-based simulator TAPES [15] to model and simulate an autonomic MPSoC system that can adapt to changing runtime conditions, although they did not use any formal control theory. Carta et al [16] presented a control theoretic approach to dynamic voltage/frequency scaling (DVFS) in a pipelined MPSoC architecture with soft real-time constraints. The techniques presented in this paper are rather generic and broadly applicable to any multiprocessor system, whether on-chip or not. For MPSoC, the processing and memory resources are typically more limited than a traditional distributed embedded systems, so it may be necessary to carefully evaluate runtime overheads of the control algorithms and their impact on control performance.

The remainder of this paper is organized as follows. Section 2 describes the system model and control architecture. Section 3 presents the details of RLS based model estimator and LQ optimal controller design. Section 4 presents performance evaluation results. Section 5 presents conclusions.

2. SYSTEM MODEL AND CONTROL ARCHITECTURE

We consider the same system model as [9]. The application is a group of end-to-end periodic tasks. Each task is composed of a chain of sub-tasks. A subtask is released when its predecessor task has finished execution. All the sub-tasks of one periodic task share the same execution rate. The hardware platform is composed of multiple processors. Each sub-task can be allocated on any processor. Each processor runs a group of periodic sub-tasks, and

the sub-task to processor mapping is predefined. Given a scheduling algorithm, a utilization bound can be determined such that the taskset on a processor is schedulable when the processor's actual utilization does not exceed the bound. Typically, higher task execution rates lead to higher Quality of Service (QoS) in terms of application utility, but also higher processor utilization, and vice versa. In order to achieve high QoS while keeping the system schedulable, the actual utilization should be controlled to be at or just below the utilization bound. In this paper, our control objective is to maintain each processor's utilization at the utilization bound (control set point) despite any execution time estimation errors and/or runtime variations by adjusting task execution rates dynamically, assuming that each task's execution rate can be adjusted within a certain acceptable range, which is true for most soft real-time applications.

The utilization model of each processor is as follows, the same as [9]:

$$y(k+1) = y(k) + B\Delta r(k), \quad (1)$$

where $y \in R^n$ represents the processor utilization vector with size n ; $\Delta r \in R^m$ represents the change to task execution rates for the m tasks running on the system; $B \in R^{n \times m}$ is defined as

$$B = GF, \quad (2)$$

Where F is the available subtask allocation matrix that records which tasks are running on which processors. $G = \text{diag}\{g_1, g_2, \dots, g_n\}$ is a diagonal matrix, where $g_i, i=1, 2, \dots, n$ are scalar values that denote the ratio between the change to the *actual* utilization of processor i and its estimation $\Delta r(k)$. The size of g_i measures the *estimation error*, i.e., how much the *actual* execution time of each task on processor i deviates from its *estimated* value, assuming that it is the same for all tasks on processor i . For example, if $g_i = 2$, then each task's actual execution time is twice the estimated value, hence the actual change to utilization caused by changing task rate is twice the estimated change. This may be due to uncertainties and runtime variations of actual task execution times, which may be quite different from their offline estimation. We will later show that our controller can handle larger values of g_i than the MPC-based controller in [9] under both steady and varying execution time conditions.

Example: We consider the distributed computing system from [9], with two processors and three periodic tasks, as shown in Fig. 2. Task 1 has a single subtask T_{11} running on Processor P_1 ; Task 3 has a single subtask T_{31} running on Processor P_2 . Task 2 has two subtasks: T_{21} running on P_1 and T_{22} running on P_2 , with the same execution rate. The system model is:

$$G = \begin{bmatrix} g_1 & 0 \\ 0 & g_2 \end{bmatrix}, F = \begin{bmatrix} c_{11} & c_{21} & 0 \\ 0 & c_{22} & c_{31} \end{bmatrix}, \quad (3)$$

$$y(k) = \begin{bmatrix} y_1(k) \\ y_2(k) \end{bmatrix}, \Delta r(k) = \begin{bmatrix} \Delta r_1(k) \\ \Delta r_2(k) \\ \Delta r_3(k) \end{bmatrix}, \quad (4)$$

The system model (1) with the parameters (3) and (4) can be rewritten as

$$\begin{aligned} y_1(k+1) &= y_1(k) + g_1(c_{11}\Delta r_1(k) + c_{21}\Delta r_2(k)) \\ y_2(k+1) &= y_2(k) + g_2(c_{22}\Delta r_2(k) + c_{31}\Delta r_3(k)) \end{aligned} \quad (5)$$

The control objective is to track the constant reference command y_{ref} , the processor utilization target vector, i.e., to minimize the error between the measured processor utilization and target utilization $e(k) = y(k) - y_{ref}(k)$. The controller task can be put on a separate processor, or it should be assigned the highest priority if it shares a processor with other applications. In the latter case, that execution time of the controller is assumed to be negligible compared to that of the application tasks. There are a utilization monitor and a rate modulator on each processor. The utilization monitor reports each processor's utilization in each sampling period, and the rate modulator is used to adjust the task execution rates. The inputs to the controller are each processor's utilization in the current sampling period, the utilization target vector and the rate adjustment range of each task. The outputs of the controller are the changes to task execution rates $\Delta r(k) = [\Delta r_1(k) \ \cdots \ \Delta r_m(k)]^T$. From the perspective of the plant, these outputs are called *control inputs*, and the rate modulators adjust the corresponding task rates according to $r_i(k) = \Delta r_i(k) + r_i(k-1), i=1,2,\dots,m$. This is a Multiple-Input, Multiple-Output (MIMO) control problem that cannot be handled with the traditional Proportional, Integral and Derivative (PID) control algorithm.

The actual gain matrix G is unknown because actual task execution times may be very unpredictable and deviate a lot from their estimated values. The MPC-based controller in [9] is designed with the assumption that the gain matrix elements $g_1 = g_2 = 1$, i.e. the controller assumes that the actual utilization change will be the same as the predicted utilization change. Even though the controller is designed under this assumption, the MPC-based controller in [9] is still stable when the *actual* g_i values fall within the range $0 < g_i < 5.95, i=1,2$ thanks to the feedback control principle.

Although these controllers are robust with small estimation errors, the control performance may deteriorate with large execution time estimation errors, i.e. large model uncertainties. In this paper, we employ online model estimation to better cope with large model uncertainties and/or runtime variations.

3. ONLINE ADAPTIVE OPTIMAL CONTROLLER DESIGN

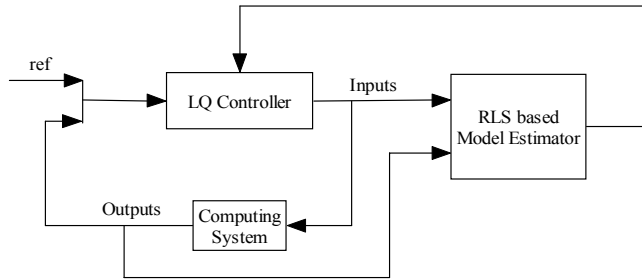


Fig. 1. Control architecture of RLS based model estimator and LQ optimal controller

Fig. 1 shows the overall control system architecture, same as [17]. The RLS-based model estimator is used to learn and update a linear model of the distributed computing system (plant), and the LQ optimal controller is used to keep the output at the desired set point by setting the control inputs by minimizing a quadratic cost function. We adopt a centralized control architecture in this paper, although it is possible to extend to decentralized control

architecture using the model decomposition technique in [11], which we leave as part of our future work.

A. RLS-based Model Estimator

The distributed computing system can be described with a general multiple-input-multiple-output (MIMO) model as follows:

$$A(q^{-1})y(k) = B(q^{-1})u(k) + d(k), \quad (6)$$

where $A(q^{-1})$ and $B(q^{-1})$ are matrix polynomials in the backward-shift operator

$$\begin{aligned} A(q^{-1}) &= I - A_1q^{-1} - \dots - A_lq^{-l}, \\ B(q^{-1}) &= B_0q^{-1} - \dots - B_{r-1}q^{-r}, \end{aligned} \quad (7)$$

where l is the order of the system, which is equal to 1 in our application. $d(k)$ is a sequence of independent, identically distributed n -dimensional random vectors with zero mean representing disturbances. We assume that $d(k)$ is independent of $y(k-j)$ and $u(k-j)$ for $j > 0$. $u(k) = \Delta r(k)$ is the control input, i.e., vector of task execution rate change, and $y(k)$ is the control output, i.e., vector of processor utilizations. (Note that the notation is slightly different from that in [9].)

We use the RLS estimator with exponential forgetting to estimate the coefficient matrices A_i and B_j online, where $0 < i \leq l$ and $0 \leq j < l$, since their values may change due to varying runtime conditions. (This is the key difference and advantage of this control scheme compared to the MPC or PID-based controllers, where the system model is fixed at runtime, especially in the face of large model uncertainties.)

For notational convenience, we rewrite the system model in the following RLS-friendly form, which we use in the remainder of the paper

$$y(k+1) = X(k)\phi(k) + d(k+1), \quad (8)$$

where

$$\begin{aligned} \phi(k) &= [u^T(k) \ \cdots \ u^T(k-l+1) \ y^T(k) \ \cdots \ y^T(k-l+1)]^T, \\ X &= [B_0, \dots, B_{l-1}, A_1, \dots, A_l], \end{aligned}$$

RLS estimator with exponential forgetting can be used to identify the time varying parameter matrix $X(k)$ online. This estimator has been applied extensively in adaptive control system design as it can converge fast and reject disturbance well. The estimator is described by the following equations

$$\varepsilon(k+1) = y(k+1) + \hat{X}(k)\phi(k), \quad (9)$$

$$\hat{X}(k+1) = X(k) + \frac{\varepsilon(k+1)\phi^T(k)P(k-1)}{\lambda + \phi^T(k)P(k-1)\phi(k)}, \quad (10)$$

$$P^{-1}(k) = P^{-1}(k-1) + (1 + (\lambda - 1) \frac{\phi^T(k-1)\phi(k)}{(\phi^T(k)\phi(k))^2})\phi(k)\phi^T(k), \quad (11)$$

where $\hat{X}(k)$ is the estimation of the $X(k)$; $\varepsilon(k)$ is the estimation error vector, $P(k)$ is the covariance matrix; λ is the forgetting factor ($0 < \lambda < 1$).

B. Linear Quadratic (LQ) Optimal Controller

The primary control objective is to let the system output track the reference command with small tracking error. At the same time, it is desirable to avoid large changes to the control inputs. We achieve

these goals by minimizing the quadratic cost function J defined as follows:

$$J = \left\| W(y(k+1) - y_{ref}(k+1)) \right\|^2 + \left\| Q(u(k) - u(k-1)) \right\|^2, \quad (12)$$

where W is a positive-semi-definite weighting matrix on the tracking errors, (a higher weight indicates higher importance value of the corresponding output variable). Q is a positive-definite weighting matrix to penalize large changes in control inputs. W and Q are defined as diagonal matrices, and their relative magnitude provides a way to trade-off tracking accuracy for smaller changes in control input. The control input that minimizes J can be obtained by setting the derivative $\partial J / \partial u(k) = 0$, and solving for $u(k)$ (The detailed derivation is described in [17] and omitted here):

$$u^T(k) = -(W^2 B_0^T B_0 + Q Q^T)^{-1} [W^2 (\hat{X}(k) \tilde{\phi}(k) - y_{ref}(k+1) B_0^T - Q Q^T u^T(k-1))]. \quad (13)$$

4. PERFORMANCE EVALUATION

In this section, we use simulation to compare the performance of the LQ optimal controller to that of the MPC-based controller.

A. Simulator and Experimental Setup

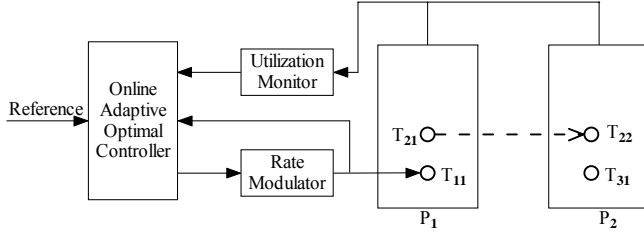


Fig. 2. Overall control system architecture.

We implemented a simulator for the utilization control of the 2-processor system introduced in Section 2 in Matlab [19]. Fig. 2 shows the overall control system architecture. We also implemented the MPC-based controller from [9] for performance comparison purposes.

Let $r_i(k)$ denote the invocation rate of Task i in the (k) th sampling period, then:

$$R_{\min,i} \leq r_i(k) \leq R_{\max,i}, i=1,2,3, \quad (14)$$

where $R_{\min,i}$ and $R_{\max,i}$ are the minimum and maximum execution rates of Task i , respectively, determined by the system designer from application domain knowledge.

The control input vector is:

$$u(k) = [\Delta r_1(k) \quad \Delta r_2(k) \quad \Delta r_3(k)]^T, \quad (15)$$

where $\Delta r_i(k) = r_i(k) - r_i(k-1)$ is change to the task rate $r_i(k)$, whose constraints are determined by the constraints of $r_i(k)$.

We assume rate monotonic scheduling on each processor, and let $y_{\max,i}$ denote the maximum utilization in processor i according to the Liu and Layland bound [6]:

$$y_{\max,i} = m_i(2^{1/m_i} - 1), i=1,2, \quad (16)$$

where m_i is the number of subtasks on processor i .

Since $m_1 = m_2 = 2$, $y_{\max,i} = 0.8284$, $i=1,2$. The runtime processor utilization must satisfy:

$$0 < y_i(k) < y_{\max,i}. \quad (17)$$

Parameter matrix F in Equation (2) and constraints in Equation (14) are as follows, same as [9]:

$$c_{11} = 35, c_{21} = 35, c_{22} = 35, c_{31} = 45, \\ R_{\max,1} = R_{\max,2} = \frac{1}{35}, R_{\min,1} = R_{\min,2} = \frac{1}{700}, \\ R_{\max,3} = \frac{1}{45}, R_{\min,3} = \frac{1}{900}.$$

As discussed in Section 2, the gain matrix G measures the estimation error. For simplicity, we assume the system gain $g_1 = g_2 = g$ in the simulation experiments, i.e., the execution time estimation errors on both processors are the same, and adjust the magnitude of g to see how well the controller handles task execution time estimation errors. The MPC-based controller in [9] was shown to be stable under the condition $0 < g_i < 5.95$, $i=1,2$.

We will show that the RLS-based LQ optimal control can tolerate a larger range of g while maintaining control stability, and when both controllers can maintain control stability, the RLS-based LQ optimal control shows better control performance, as measured by the *aggregate error*, defined in Equation (19) as the sum of squared errors between the target utilization and the actual measured utilization on each processor during a time interval while the system is in the steady state.

$$E = \left(\sum_{k=T_1}^{T_2} e_i^2(k) \right) / (T_2 - T_1), \quad (19)$$

where $e_i(k)$ is the tracking error of processor i 's utilization at time step k . T_1 and T_2 denote the simulation start and stop time steps for calculating the aggregate error. In our simulation experiments, we run the simulation for 1000 time steps, and the system goes into steady state before time step 200, so we set $T_1=200$ and $T_2=1000$.

B. Experiment 1: Steady Execution Times, Small System Gain

In this experiment, we set $g_1 = g_2 = 0.35$, i.e. the actual execution time is 35% of the estimated value. The initial task rates are assigned based on *estimated* execution times to make the utilization equal to the set point. Hence both processors are *actually* underutilized initially. The task rates are then increased gradually until the utilization of both processors converges to the set point of 0.8284. The objective is to keep the processor utilizations at the set point without knowledge of the actual task execution times while achieving good control performance as measured by the aggregate error defined in Equation (19).

Fig. 5 shows the processor utilization responses. (Note that the figure shows two curves for utilizations of the two processors, but they are quite close to each other so they look like one curve.) Although these two approaches both have acceptable performance in the steady state, the aggregate error of the LQ optimal controller in the steady state is slightly smaller than that of MPC controller, as shown in Table 1.

Table 1. Aggregate errors in Experiment 1

Algorithm	LQ Controller	MPC
CPU1	0.0173	0.0216
CPU2	0.0166	0.0202

C. Experiment 2: Steady Execution Times, Large System Gain

In this experiment, we set $g_1 = g_2 = 7$, i.e. the actual execution time is seven times the estimated value. Both processors are initially over-utilized. The task rates are then decreased gradually until the utilization of both processors converges to the set point of 0.8284.

Fig. 4 shows the processor utilization responses. For the LQ optimal controller, the utilizations exhibit some initial oscillations due to model estimation inaccuracies, but as model estimation becomes more accurate later, they converge to the utilization set point quickly. For the MPC-based controller, the estimation error falls outside of the stability range, and processor utilizations

oscillate significantly during the entire simulation time, which is obviously not acceptable for a real-time system. Intuitively, the *actual* utilization change caused by the rate modulator at each control step is much larger than (seven times) the *estimated* utilization change due to the grossly under-estimated task execution times, which caused the control instability. Table 2 shows the aggregate errors.

Table 2. Aggregate errors in Experiment 2

Algorithm	LQ Controller	MPC
CPU1	0.0167	0.4170
CPU2	0.0161	0.3719

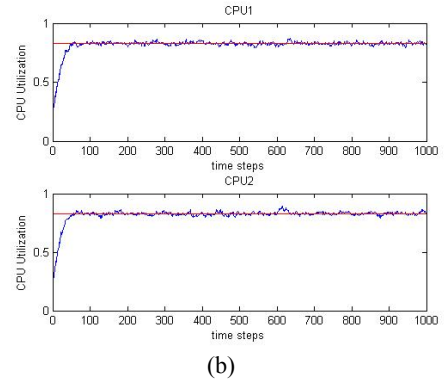
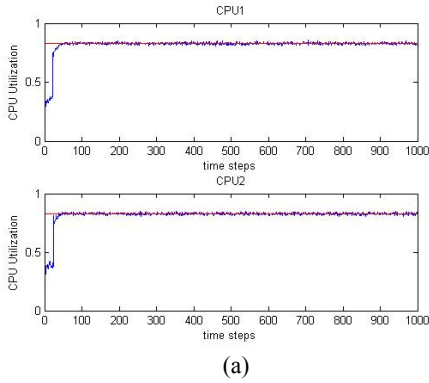


Fig. 3. Processor utilization response in Experiment 1 ($g=0.35$): (a). LQ optimal controller; (b). MPC-based controller

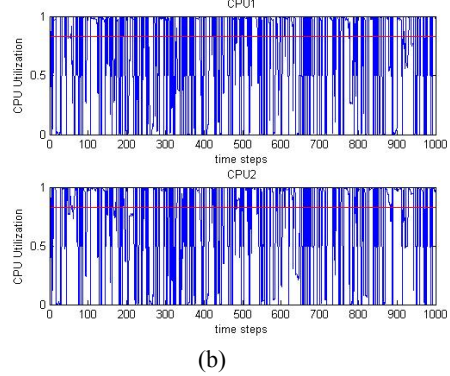
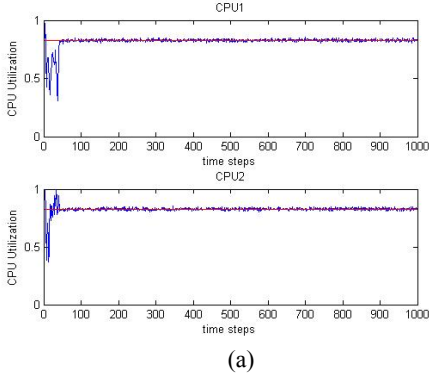


Fig. 4. Processor utilization response in Experiment 2 ($g=7$): (a). LQ optimal controller; (b). MPC-based controller

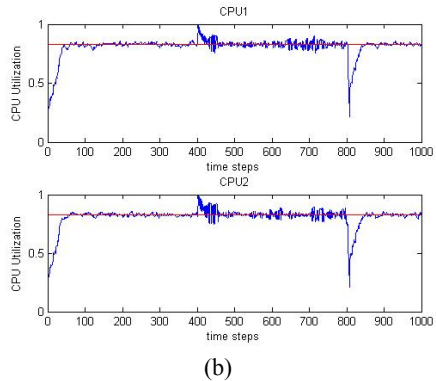
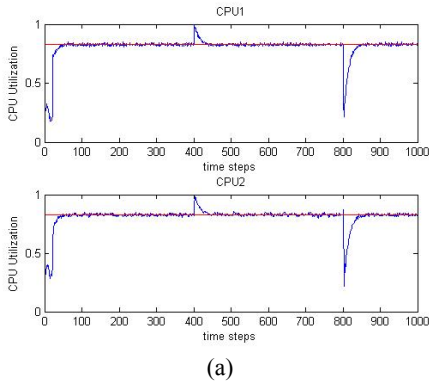


Fig. 5. Processor utilization response in Experiment 3 ($g=0.35, 2, 0.5$): (a). LQ optimal controller; (b). MPC-based controller

D. Experiment 3: Varying Execution Times

In this experiment, task execution times vary dynamically at runtime. We set $g_1 = g_2 = 0.35$ at simulation start time, $g_1 = g_2 = 2$ at the 400th time step, and finally $g_1 = g_2 = 0.5$ at the 800th time step. This experiment is designed to test the robustness of the controller in the face of runtime workload fluctuations.

Fig. 5 shows the processor utilization responses. When the workload is changed at the 400th and 800th sample steps, the LQ optimal controller keeps the utilizations at the desired set point with much smaller oscillation than the MPC-based controller, as confirmed by Table 3.

Table 3. Aggregate errors in Experiment 3

Algorithm	LQ Controller	MPC
CPU1	0.0185	0.0215
CPU2	0.0163	0.0324

5. CONCLUSIONS AND FUTURE WORK

In this paper, we address the processor utilization control problem in multiprocessor real-time embedded systems. In these systems, execution time may vary greatly, so an adaptive control approach is needed to keep the processor utilization at a given set point to ensure schedulability. MPC-based controller can handle execution time estimation errors within a certain range. To deal with large estimation errors, we employ Recursive Least Squares (RLS) based model estimator to estimate the system model online, and Linear Quadratic (LQ) optimal controller to keep the processor utilization at the desired set point. Simulation experiments shows that the LQ optimal controller has better system performance than the MPC-based controller under both constant and varying task execution time conditions, especially when the execution time estimation errors are large.

ACKNOWLEDGEMENTS

This work was partially supported by NSERC Discovery Grant #341823-07, Hong Kong RGC CERG #613506, and a National Study-Abroad Scholarship of P. R. China under Grant No. [2007] 3020.

REFERENCES

- [1] L. Sha and J.B. Goodenough, "Real-Time Scheduling Theory and Ada," *Computer*, vol. 23, no. 4, pp. 53-62, 1990.
- [2] A.J. Garvey and V.R. Lesser, "Design-to-time real-time scheduling," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 23, no. 6, pp. 1491-1502, 1993.
- [3] A. Goel, Walpole, and M. Shor. "Real-rate scheduling," in *proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 434-441, 2004.
- [4] S. Lin and G. Manimaran. "Double-Loop Feedback-Based Scheduling Approach for Distributed Real-Time Systems," in *proceedings of the High Performance Computing (HiPC)*, pp. 268-278, 2003.

- [5] J.A. Stankovic, T. He, T.F. Abdelzaher, M. Marley, G. Tao, S.H. Son, and C. Lu. "Feedback Control Real-Time Scheduling in Distributed Real-Time Systems," in *proceedings of the IEEE Real-Time Systems Symp*, 2001.
- [6] J. Liu, *Real-Time Systems*: Prentice Hall PTR 2000.
- [7] C. Lu, J.A. Stankovic, S.H. Son, and G. Tao, "Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms," *Real-Time Systems*, vol. 23, no. 1, pp. 85-126, 2002.
- [8] F. Yong, W. Hongan, L. Chenyang, and A.R.S.C. Ramu Sharat Chandra. "Distributed Utilization Control for Real-Time Clusters with Load Balancing," in *proceedings of the 27th IEEE International Real-Time Systems Symposium*, pp. 137-146, 2006.
- [9] C. Lu, X. Wang, and K. X., "Feedback utilization control in distributed real-time systems with end-to-end tasks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 16, no. 6, pp. 550-561, 2005.
- [10] X. Wang, Y. Chen, C. Lu, and X. Koutsoukos, "FC-ORB: A robust distributed real-time embedded middleware with end-to-end utilization control," *Journal of Systems and Software*, vol. 80, no. 7, pp. 938-950, 2007.
- [11] X. Wang, D. Jia, C. Lu, and X. Koutsoukos, "DEUCON: Decentralized End-to-End Utilization Control for Distributed Real-Time Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 7, pp. 996-1009, 2007.
- [12] Y. Chen, C. Lu, and X. Koutsoukos, "Optimal Discrete Rate Adaptation for Distributed Real-Time Systems," in *proceedings of the 28th IEEE International Real-Time Systems Symposium*, pp. 181-192, 2007.
- [13] G. Lipsa, A. Herkersdorf, W. Rosenstiel, O.A.B.O. Bringmann, and W.A.S.W. Stechele. "Towards a Framework and a Design Methodology for Autonomic SoC," in *proceedings of the Second International Conference on Autonomic Computing*, pp. 391-392, 2005.
- [14] A. Lankes, T. Wild, and J. Zeppenfeld, "System Level Simulation of Autonomic SoCs with TAPES," *ARCS* vol. 4934/2008, pp. 9-22, 2008.
- [15] T. Wild, A. Herkersdorf, and G.-Y. Lee, "TAPES - Trace-based architecture performance evaluation with SystemC," *Design Automation for Embedded Systems*, vol. 10, pp. 157-179, 2006.
- [16] C. Salvatore, A. Andrea, P. Alessandro, A. Andrea, and B. Luca, "A control theoretic approach to energy-efficient pipelined computation in MPSoCs," *ACM Transactions on Embedded Computing Sys.*, vol. 6, no. 4, pp. 27, 2007.
- [17] X. Liu, X. Zhu, P. Pradeep, Z. Wang, and S. Sharad. "Optimal multivariate control for differentiated services on a shared hosting platform," in *proceedings of the 2007 46th IEEE Conference on Decision and Control*, pp. 3792-3799, 2007.
- [18] M. Karlsson, X. Zhu, and C. Karamanolis. "An adaptive optimal controller for non-intrusive performance differentiation in computing services," in *proceedings of the International Conference on Control and Automation*, vol. 2, pp. 709-714, 2005.
- [19] *The MathWorks, MATLAB Function Reference*, 2007.