Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Response–Time Control of a Processor–Sharing System using Virtualized Server Environments ⋆

Martin A. Kjær * **Maria Kihl** ** **and Anders Robertsson** *

* *Department of Automatic Control, LTH, Lund University, Box 118, SE-221 00 Lund, Sweden, (email: {Martin.A.Kjaer,Anders.Robertsson}@control.lth.se)*
** *Department of Electrical and Information Technology, LTH, Lund University, Box 118, SE-221 00 Lund, Sweden, (email: Maria.Kihl@eit.lth.se)*

Abstract: Feedback in server systems has during last years gained much interest in order to fulfill still increasing demands on performance and optimization regarding, for example, quality of service (QoS) requirements. In this paper we expand a previously published feedback–based prediction scheme for controlling a single server queue together with a new control strategy. These control structures have the benefit over other previously suggested control structures that no off–line estimation of the required work is needed. In addition, our solutions maintain or improve the performance, regarding average response time and loss of computational resources.

1. INTRODUCTION

Server systems are used in most (tele)communication networks. The server system can process jobs from either other network entities or from clients. In the Internet, web servers send web pages to clients. Virtualized server environments have become popular within the server hosting for many reasons (isolation, platform independentness, and other). Being able to isolate the resource allocation and consumption between several applications now allow designs for better utilization of the resources than if the applications were to share the resources or, alternatively, hosted by separated physical resources. Virtualized server environments are still restricted by limited physically resources, which impose interesting control problems; how to divide resources optimally between the virtualized server environments, and how to guarantee that the virtualized server environments are ensure, and restricted to, the resources assigned to them. Some virtualization schemes allow a new form for actuation, allowing the designer to define the amount of computational resources dedicated to the virtual environment online, and thus, the amount dedicated to the server, see Xu et al. [2006], Wang et al. [2007]. Since a server system consists of CPUs, it has a non-linear behavior. A small increase in load can result in rapidly growing response times. Therefore, the design of the control mechanisms is not a simple task if optimized performance is desired. In recent years this field has gained a large research interest from both academia and IT vendors (Hellerstein et al. [2004]).

In this paper we focus on response-time control (the response–time T is defined as the time between the arrival of a job and the time when is has been fully served, also often denoted the system time, see Kleinrock [1975]); an objective directly coupled to the end-user, but with close relations to *e.g.* the more server oriented queue length control (Kihl et al. [2007]). However, the connection between the queue length and the response time depends on numerous factors such as the nature of the arriving traffic and the service times. Liu et al. [2006a] designed a response time control system based on admission control, where the admission probability was adjusted to obtain the desired response time. The control set–up was based on a measurement of the average required work. This was measured off–line under low load, and then applied in online control. The off–line identification of the average required work was also the procedure of the work by Henriksson et al. [2004] where feed–forward was used as part of control set–ups with resource allocation actuators. This method can result in serious degradation of performance if the average required work changes, as often happen in real applications. The required work is in general not easy to define in real server systems that often operate by processor sharing, and it can be hard to give any qualified estimates of this quantity.

In this paper, we expand a previously published control scheme (see Kjær et al. [2007]) to cope with periodical actuation based on computational resources allocation, as *e.g.* with server virtualization schemes. We also assume processor sharing serving instead of single server processing, and more bursty traffic. The prediction method is utilized for a new response–time control scheme. This controller is shown, through simulations, to maintain better desired robustness and transient properties than other published control strategies

2. SYSTEM DESCRIPTION AND MODELING

We consider a processor–sharing system. At a given time, the computational resources available to the processor are divided uniformly between the jobs present. Since

⋆ This work has been founded by the Swedish Research Council, project 621-2006-5522.

the processor will accept any incoming job, no queue is present. In virtualized server environments the CPU–resources are often shared between several applications. We treat our processor system as one application where the dedicated (reserved) share of the CPU is treated as an abstract computational resource, which can be set online at certain time intervals in the interval $p \in [0.05, 1]$. In the case where more computational resources are allocated (reserved) to the processor than the processor requires, the over–allocation is considered as a loss of computational resources, which could else have been utilized by other applications. Also, it is assumed that certain calculations are allowed at each departure, and that measurements of former response times, arrival times, and number of jobs, are available as measurements. For the development of controllers, no specific distributions for the arrivals and the required work are assumed. We do not consider the case where the processor can hold only a limited number of jobs. The arrival times and the required work are treated as disturbances, whereof only the arrival times are measurable.

2.1 Purpose of Control

Loss of computational resources occurs when resources are dedicated (reserved) to the processor, but no jobs are present to use the resources. Therefore, from a resource optimization point of view, it is desirable to ensure that there always are job to serve. Since jobs can not be invented by the processor, the only solution is to reduce the service rate, and thereby holding jobs longer in the processor. This has a cost on the application performance, since the response times will increase. The trade–off is then to ensure that there are jobs to serve, but at the same time, to maintain an acceptable response time. The goal of control is to ensure that the response time is as specified by T_{ref}, and minimizing the loss of computational resources. A third performance metric is the variation of the response times. The smaller it is, the better.

2.2 Modeling

In queuing theory it is well–known that a M/M/1 and a M/G/1–PS system have the same average response time, see Kleinrock [1967] and M. Sakata and Oizurnih [1971]. This means that a processor–sharing system without a queue behaves similarly (in terms of average response time) as a single server with an infinite queue. Modeling the system by a single server means that the jobs can not pass one another in time (the first job that enters is the first job to leave), which simplifies estimation significantly.

Since the system is an event driven system, we chose to model what happens when an event occurs. Consider the case where a job leaves the processor at time t_{now}, leaving N remaining jobs. This is illustrated in Fig. 1. Assuming that all jobs will have the same required work $\hat{w}$ and that the allocated computational resources p remain constant, a prediction of the average response time $\hat{T}$ is given by

$$\hat{T} = 1/N \sum_i (t_{now} - a_i) + \hat{w}(N+1)/2p \qquad (1)$$

where a_i is the arrival time of job i. The first term to the right of the equality sign represents the amount of time

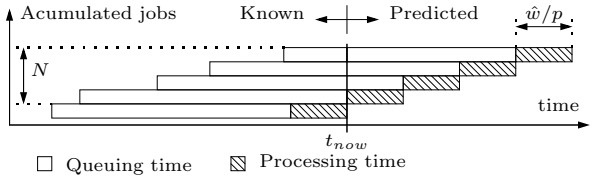

Fig. 1. Accumulated jobs in a single server queue.

that the jobs have already spend in the processor. This quantity is known by measurement, and is illustrated as the area left of t_{now} in Fig. 1. The second term of the equation represents the expected time that the jobs will have to stay in the processor for completion. This quantity is based on an estimate, and is illustrated as the area right of t_{now} in Fig. 1. (see Henriksson et al. [2004] and Kjær et al. [2007] for a more detailed explanation).

3. ESTIMATION

The model described above can be useful for online adjustment of the response time, but it relies on several measurements. The number of jobs in the system N, and their arrival times a_i are quantities often registered by a real system. If the prediction is executed at the departure, t_{now} is simply found by reading the clock. The average required work $\hat{w}$ is a lot harder to find. In a single server with a queue it could be estimated by measuring former service times, corrected with the allocated computational resources, but for processor sharing systems, this is not possible. Therefore, relaying on a measurement of the required work will reduce the applicability of the estimation. We therefore seek another strategy.

In classical linear estimation–problems models are used to estimate non–measured quantities, see for example Åström and Wittenmark [1997]. Often the measurable variables are compared to the estimated values, and a feedback mechanism tries to minimize the estimation error. We consider T as a measurable output and w as a state to be estimated. In Kjær et al. [2007] we proposed a redesign of the response–time prediction presented in Henriksson et al. [2004], resembling the structure of a classical observer. Because the dynamics of the system are not well known, we suggest to use a PI–controller due to its robustness. To stress that the estimator does not rely on measurements of w, we impose an artificial variable z to act as the input to the model. The estimator now takes the form

$$\hat{T} = \frac{1}{N} \sum_i (t_{now} - a_i) + \frac{(N+1)}{2p} z \qquad (2)$$

$$I_k = I_{k-1} + \frac{h_k K}{T_i}(T_k - \hat{T}_k) + \frac{h_k}{T_a}(v_{k-1} - z_{k-1}) \qquad (3)$$

$$v_k = K\,(T_k - \hat{T}_k) + I_k \qquad (4)$$

$$z = \begin{cases} v \;\; for \;\; v > 0 \\ 0 \;\; else \end{cases} \qquad (5)$$

where T_i and K are controller parameters. The variable h_k is the time between the previous and the current sampling. Using a varying sampling periods in the integrator has earlier been shown by Årzén [1999] to be superior to fixed sample–periods for some event based systems. Integrator anti–windup is included as the last term in (3), where $z(k)$

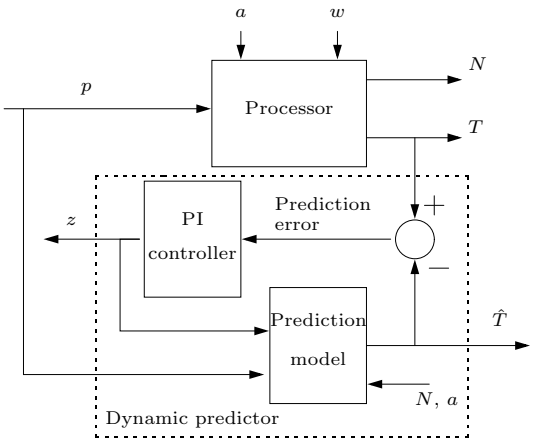

Fig. 2. Block diagram of predictor. The predictor can be interpreted as an observer with state z.

is the achieved control signal (z is not allowed to be negative). The parameter T_a determines the convergence rate of the anti wind–up, see Åström and Wittenmark [1997]. The parameters $T_i = 0.0005$, $K = 0.000001$, $T_a = 0.5$ were chosen for the investigations to be presented. The estimation scheme is also illustrated in Fig. 2. Equations (3) and (4) form a general PI controller, where the term z_{k-1} is exchanged with the relevant control signal.

4. CONTROLLER DESIGN

4.1 Suggested Controllers

The system suffers from a significant time delay; a change in p will only propagate to T after a certain time. In classical control theory, the performance of systems with delays can be improved by prediction techniques, such as the Smith predictor, see Åström and Hägglund [2005]. Inspired by this idea, two control strategies have been designed based on the estimator in (2). Due to lack of theoretical tools to analyze event–driven nonlinear systems, no analysis of the stability of the controlled systems have been performed. Inspiration to how this could be done can be seen in e.g. Kihl et al. [2007].

State Feedback Controller (*state FB*). The predictor in (2)-(5) is seen as an observer, and z as a state. A state feedback can now be formed by inverting the prediction model (2)

$$p_{state} = \frac{N+1}{2(T_{ref} - \frac{1}{N}\sum_i (t_{now} - a_i))}\, z \ . \qquad (6)$$

as suggested in Kjær et al. [2007]. To handle model errors, a periodic PI–controller as in (3) and (4) is added ($K = 10^{-4}$, $T_i = 0.0101$, $T_a = 1010.1$). The control structure is illustrated in Fig. 3.

PI with Predictive Feedback Controller (*PI–PFB*). A periodic PI controller similar to the one used in the *state FB* controller is used for feedback of the response time. This suffers from the time delay discussed earlier, and feedback from the predicted response time can allow much faster reaction. We chose to use a proportional controller

$$p_p = -K_p(T_{ref} - \hat{T}) \ . \qquad (7)$$

with $K_p = 0.2$. The structure is illustrated in Fig. 4.

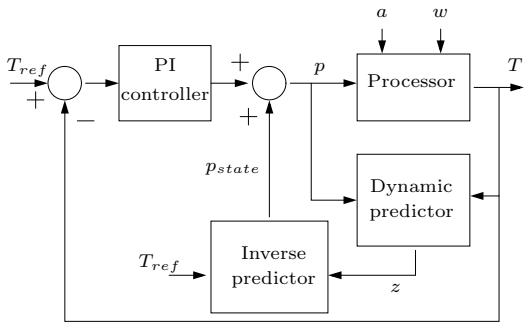

Fig. 3. Block diagram of the suggested *state FB* controller.

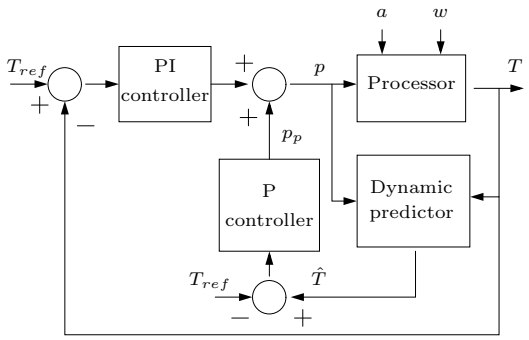

Fig. 4. Block diagram of the suggested *PI–PFB* controller.

4.2 Sample Periods

It is an assumption that the computational resources dedicated to the processor only can be changed at some fixed time period, T_s. However, the estimation is not restricted by the sampling period. The estimation is updated for each departure, ensuring that the estimate $\hat{T}$ and the state z always incorporate the newest measurements. When a sample incident occurs, the control signal can be based on data accumulated not just at the former sampling incidences, but also on data obtained in–between.

4.3 Controllers for Comparison

Earlier work presents solutions where feed–forward and feedback are combined as illustrated Fig. 5, see e.g Liu et al. [2006b], Lu et al. [2003], and Henriksson et al. [2004]. These all assume that an estimate of the average required work is available. An often used procedure to obtain the estimate is to measure the response times at very low arrival rates. Assuming that only one job is present, the response times can be used to estimate the required work. The estimate is then used online at higher arrival rates, assuming that it will remain constant. This method is not robust towards changes in the required work, as, for example, when the relative popularity within a certain website suddenly changes. We present two controllers based on this principle for comparison.

PI with Inverse Prediction Feed–Forward (*PI–IPFF*). A periodic PI controller as in (3) and (4) is used for feedback ($K = 1.4 \cdot 10^{-5}$, $T_i = 0.0101$, $T_a = 1010.1$). Equation (1) is rearranged, and the $\hat{T}$ is exchanged with the desired value T_{ref}

$$p_{ff} = \frac{N+1}{2(T_{ref} - \frac{1}{N}\sum_i (t_{now} - a_i))}\hat{w} \ . \qquad (8)$$

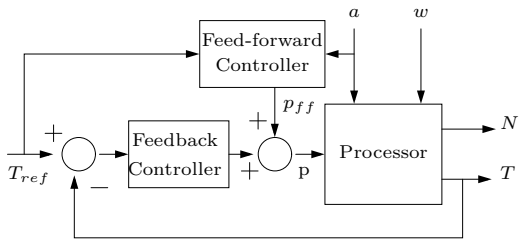

Fig. 5. Block diagram of a combined feedback, feed–forward setup for the processor system.

This is a slightly modified version of the feed–forward presented in Henriksson et al. [2004].

PI with Queuing–Theoretic Feed–Forward (*PI–QFF*). A periodic PI controller similar to that used in the *PI–IPFF* controller is used for feedback. The feed–forward is based on queuing–theoretic models. The average response time of a M/G/1-PS in stationary is given by (Kleinrock [1967], M. Sakata and Oizurnih [1971])

$$T = \bar{x}/(1 - \lambda\,\bar{x}) \ . \tag{9}$$

Equation (9), or varieties hereof, have formed the base for other feed–forward designs in the literature, *e.g.* Liu et al. [2006b], Lu et al. [2003]. In our case, $\bar{x} = \bar{w}/p$ if p is constant. Assuming that $\bar{w}$ is known (and exact), and λ is estimated by some windowing mechanism ($\hat{\lambda}$), a feed–forward signal can be formed as

$$p_{ff} = w\,\bar{(}1 + \hat{\lambda}\,T_{ref})/T_{ref} \tag{10}$$

4.4 Filtering Issues

The involved signals can be quite irregular, which can lead to irregular estimation and poor control performance.

All the tested periodic PI controllers use the comparison of the reference and a filtered response time T^p; $T^p_k = (T^p_{k-1} + T^s_k)/2$ where T^s_k is the average response time of the jobs that departed under the interval between sampling $k-1$ and k.

To update the estimator, the estimated response–time is compared to a filtered measured response–time; $T^f_i = 0.999\,T^f_{i-1} + 0.001\,T_i$, where i indicates the departing job number, and T_i is the response time of job i.

The response time estimate can also be quite irregular. An obvious idea is to apply a filter directly to the estimate $\hat{T}$. This has an undesirable effect due to the nonlinear structure. Linear filtering of the term $1/p$ would weight small values of p and could lead to wrong estimates. Also, a linear filtering of the term $\sum_i (t_{now} - a_i)$ would weight the jobs that have a long service time over those having a short response time, thus increasing the average estimate. The filtering must therefore be placed with care. The best results have been obtained by simply filtering N; $N^f_i = 0.999\,N^f_{i-1} + 0.001\,N_i$, which is an event–based filter.

The *state FB* controller and the *PI–IPFF* controller are based on inverse prediction. That is, any response time error is compensated in one update. A similar idea is used in classical minimum–variance control, which is known to have poor robustness properties, see Åström [2006]. In our case, the result is an undesirable irregular control signal, and some filtering is imposed. Applying a filter to the control signal would drive the average control signal off due to the nonlinearity of the fraction in (6) and (8). Therefore, the numerator and denominator are filtered separately;

$$P_i = 0.999\,P_{i-1} + 0.001\,(N+1)\,z \tag{11}$$

$$Q_i = 0.99\,Q_{i-1} + 0.01\,\frac{1}{N}\sum_i (t_{now} - a_i) \tag{12}$$

$$p_{state} = P_i/2(T_{ref} - Q_i) \tag{13}$$

for the *state FB* controller. For the *PI–IPFF* controller, p_{state} is substituted for p_{ff}, and z for $\bar{w}$.

4.5 Simulations

We investigated the controllers with simulations. We used a discrete-event simulation program written in Java. The Java-program included classes for the traffic generator, the processor, the observer, and the controller.

Steady state and transient simulations were performed. All steady–state results were evaluated after all transients had been removed. Transient behavior was investigated after convergence to steady state, and was allowed to run for sufficiently long time for the transient to settle.

To achieve sufficient burstiness, we used second order hyper–exponential (H_2)–distributions to generate inter–arrival times and required work. A H_2–distributed variable x is with probability α a realization of logarithmic distributed variable with expected value a_1, and with probability $(1-\alpha)$ a realization of logarithmic distributed variable with expected value a_2. We used the parameters:

$$\alpha = (C^2 - 1)/(C^2 + 161) \tag{14}$$

$$a_1 = 0.1\,\bar{x}\ ,\ a_2 = \bar{x}(1-\alpha)/(1-10\alpha), \tag{15}$$

where C^2 and $\bar{x}$ were the variance coefficient and average value of the H_2–distributed sequence, respectively. The value of C^2 were chosen equal for the inter–arrival times and the required work distributions, and had the value $C^2 = 5$ unless stated differently.

Average values and confidence intervals were evaluated on down–sampled data to remove correlation between the measurement points. The size of the 95%–confidence intervals for the response time did not exceed 10% of the mean value. The size of the confidence interval for the loss of computational resources did not exceed 1% of the maximum available computational resources. To evaluate the capability to maintain low variation of the response times the variation–cost function was defined as

$$J_s = \frac{1}{N}\sum_{k=1}^{N}(\bar{T} - T_k)^2 \quad , \quad \bar{T} = \frac{1}{N}\sum_{k=1}^{N} T_k \tag{16}$$

In the transient investigation we could not rely on long simulation runs to obtain sufficient accurate results when evaluating performance. Therefore, the transient periods were averaged over a large number M of experiments. To compare the performance of the simulation runs, we used the cost functions

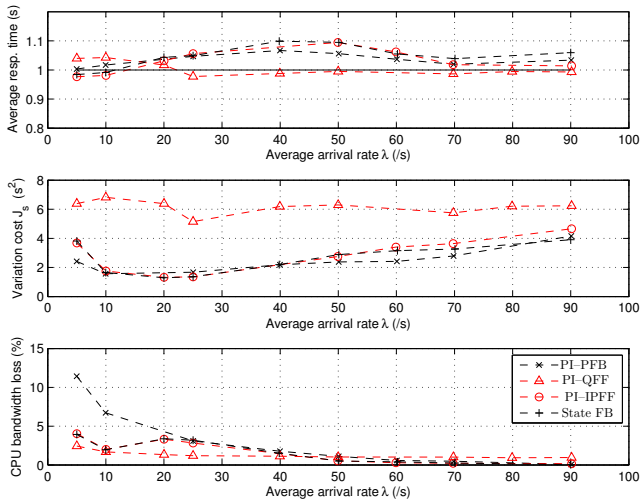

Fig. 6. Averaged steady state simulations for different arrival rates λ. $C^2 = 5$, $\bar{w} = 0.01\,s$, $\hat{w} = 0.01\,s$, $T_s = 1\,s$, $T_{ref} = 1$.

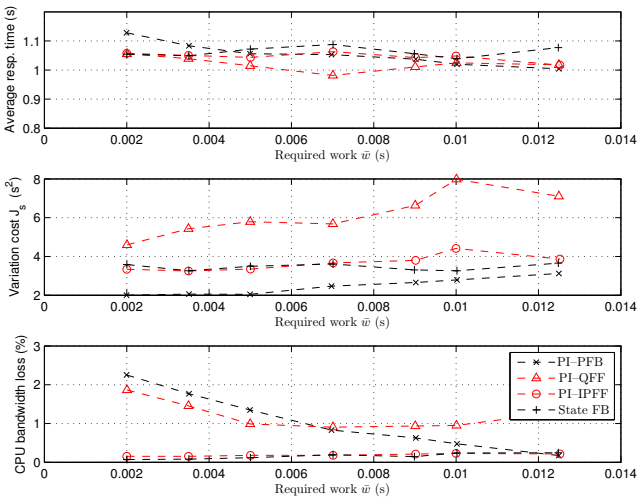

Fig. 7. Averaged steady–state simulations for different average required work $\bar{w}$. $C^2 = 5$, $\lambda = 70\,s^{-1}$, $\hat{w} = 0.01\,s$, $T_s = 1\,s$, $T_{ref} = 1\,s$.

$$J_T(M) = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{T_t}\sum_{k\in T_t}(T_{ref} - T_k)^2 \tag{17}$$

$$J_p(M) = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{T_t}\sum_{k\in T_t}(q_k)^2 \tag{18}$$

where T_t was the transient period, and q_k was the loss of computational resources.

5. RESULTS AND DISCUSSION

5.1 Steady State Simulations

Traffic Load The traffic is often described by two traffic quantities; the arrival rate (λ) and the nominal service rate ($1/\bar{w}$). Often, the traffic is quantified by the traffic factor $\rho = \lambda\bar{w}$. A low value of ρ means a lightly loaded system, where as values close to, but below, one means a heavy loaded system. If ρ exceeds one, the system lack resources to serve incoming requests, and the system is overloaded.

In the following, the effect of the arrival rate and the service rate is investigated. Fig. 6 shows the performance metrices when the arrival rate was varied in a range corresponding to $\rho = 0.05 - 0.90$. It indicates that all the controllers managed to keep the average response time near the reference, however, small off–sets were observed. All controllers performed best when the arrival rate was relative high, since the loss of computational resources was small. Especially the *PI–PFB* controller seemed to perform poorly at very low arrival rates. In the simulations, the estimate of the required work $\hat{w}$ corresponded exactly to the average required work $\bar{w}$ for the controllers for comparison. The suggested controllers estimated this parameter online, but showed no significant degradation in performance because of this. The *PI–QFF* showed a higher variation–cost J_s, which indicates a less smooth response time than the other controllers.

Fig. 7 shows the performance metrices when the required work was varied in a range corresponding to $\rho = 0.14 - 0.875$. It indicates that all the controllers managed to keep the average response time near the reference, however, small off–sets were observed. Also here, the queuing based prediction showed to perform rather poorly over the full range, since it yield a higher loss of computational resources, but also a large variation–cost J_s. In the simulations, the estimate of the required work $\hat{w}$ did not corresponds exactly to the average required work $\bar{w}$ for the controllers for comparison. Despite this, the inverse prediction feed–forward controller performed well in steady state because of the robustness of the PI–controller.

Response–Time Reference By ensuring that there always are jobs to process, the loss of computational resources will be minimal according to the principle behind the designs. This is obtained by allowing a higher average response time than what might be possible with more allocated resources. Fig. 8 shows that there was a limit where it was not worth to increase the response time reference any further. In the given case, response–time references above approximately one did not result in any significant reduction in the loss of computational resources. It does not matter, for the loss of computational resources, whether there were few (but always some) or many jobs in the system. Thus, a higher response–time reference would only generate higher average response time, but no improvements.

When the response times became too short, the risk of an empty system becomes significant, which introduces loss of computational resources. This is clearly indicated for the low response–time reference of Fig. 8.

Traffic variance Traffic might be more bursty than in the examples presented above. Fig. 9 shows how the suggested controllers performed under other types of traffic, at different arrival rates. The figure indicates that generally, the controllers did not depend much on the burstiness of the traffic. Only the *state FB* controller seemed to have a higher loss of computational resources at low burstiness and low arrival rate. Generally, the variation–cost J_s increased with the burstiness, but this is a natural trend, originating from the traffic itself.

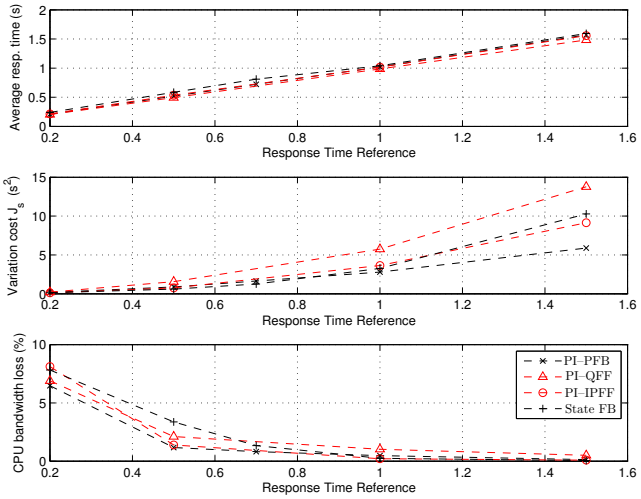

Fig. 8. Averaged steady–state simulations for different response time references T_{ref}. $C^2 = 5$, $\lambda = 70\,s^{-1}$, $\bar{w} = 0.01\,s$, $\hat{w} = 0.01\,s$, $T_s = 1\,s$.

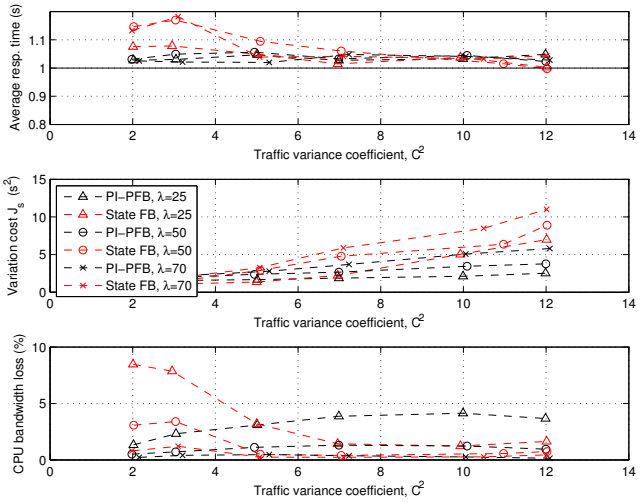

Fig. 9. Averaged steady–state simulations for different arrival rates and different variance coefficient C^2. $\bar{w} = 0.01\,s$, $\hat{w} = 0.01\,s$, $T_s = 1\,s$, $T_{ref} = 1\,s$.

5.2 Transient Simulations

One strong argument to use feedback in the control is the robustness towards changes in the environment. The above simulations were all performed with steady–state environments, which means that both the required work and the arrival rate had constant distributions. It is of high relevance to investigate the behavior of the controlled system under changes in the environment.

The top diagram of Fig. 11 illustrates cost functions for the transient behavior of the system when exposed to change in the required work. In the beginning of the experiment the load was relatively low with $\rho = 0.4$ ($\lambda = 50\,s^{-1}$, $\bar{w} = 0.008\,s$). At time $t = 1000$ the average required work was doubled, such that the system was exposed to high–load traffic, $\rho = 0.8$. Fig. 10 and the middle and bottom of Fig. 11 illustrates the results for the transient behavior of the processor when the arrival rate was changed. Initially, the system was exposed to a low–load traffic with $\rho = 0.35$

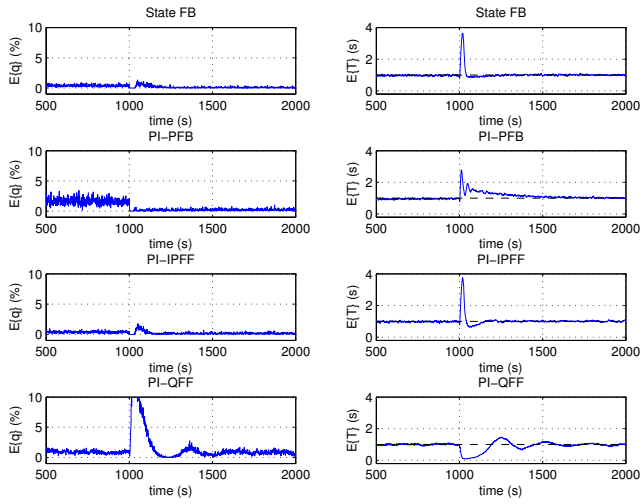

Fig. 10. Time–domain transient results with changing arrival rate and high traffic burstiness (C^2=5). Each plot represents an average of 250 simulation runs.

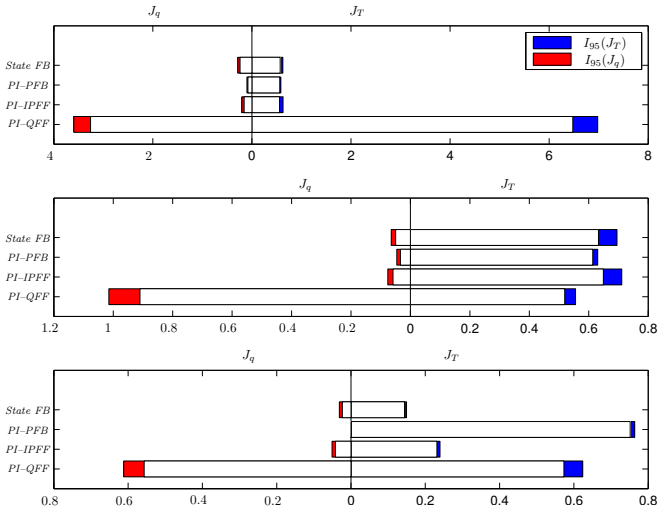

Fig. 11. Cost–function results. Top: Transient simulations case with changing required work (M=150, T_p=1700 s). Middle: Transient simulations case with changing arrival rate and high traffic burstiness. (C^2=5, M=250, T_p=300 s). Bottom: Transient simulations case with changing arrival rate and low traffic burstiness. (C^2=1.1, M=150, T_p=300 s).

($\lambda = 50\,s^{-1}$, $\bar{w} = 0.007\,s$). At time $t = 1000\,s$ the arrival rate doubled, such that the system was exposed to high–load traffic, $\rho = 0.7$. An incorrect estimate of the required work for *PI-IPFF* and *PI–QFF* controllers was used for all the transient simulations ($\hat{w} = 0.01$). The experiment presented in the middle and bottom of Fig. 11 had traffic variance coefficients C^2=5 and C^2=1.1, respectively

A first observation is that the queuing based feed–forward control responded poorly to changes; in the case of increasing required work the feed–forward did not do any difference, since it only considered the arrival rate (and a wrong, static, estimate of $\bar{w}$). Therefore, the periodic PI–controller had to handle the change, which was rather slow, resulting in a large deviation of both the response time and a large loss of computational resources. In the

cases where the arrival rate changed, the queuing theory based feed–forward over–compensated, resulting in a large loss of computational resources.

For change in required work, the *PI–IPFF* and the *state FB* controller showed no significant difference in performance. Immediately after the change, none of the controllers had a good estimate of the required work. The *state FB* controller adapted to the change, but because the prediction controller was rather slow, this did not have any significant effect during the investigated transient. A difference between the two controllers was observed when the arrival rate was changed and the burstiness was low (the bottom of Fig. 11). Here, the *state FB* controller had a valid estimate of the required work under the transient, and was thus capable of handling the transient better than the *PI–IPFF* controller, which had a false estimate. When more bursty traffic was imposed, the difference between the *state FB* controller and the *PI–IPFF* controller vanished (the middle of Fig. 11).

For all high–burstiness simulations our proposed *PI–PFB* controller showed superior transient response.

6. CONCLUSIONS

In this paper we have presented design of two controllers for a processor sharing system where the dedicated computational resources could be set at fixed sample times. The performance of the two suggested controllers was compared to two other controllers from the literature.

A general trend in all the investigation was the poor performance of the *PI–QFF* controller. It was based on a fixed, off–line estimated, required work, and only considered long–term averages in the feed–forward part. Only with low arrival rate, where the stochastics of the traffic became dominating, this controller performed similarly, or a bit better, than the other controllers. The *PI–IPFF* and the *state FB* controller differ in the estimate of the required work; off-line estimation and online estimation, respectively. The investigations showed that the dynamic estimation had most impact on the transient behavior. In the steady state, the feed–forward based on a poor estimate of the required work was compensated by the PI–controller. In the transient simulations, the dynamic estimator showed better performance if the traffic was not too bursty. In bursty conditions, the stochastics became so dominating that the prediction inverse model was too inaccurate to ensure proper control. In many of the experiments, our suggested *PI–PFB* controller showed the best performance. The transient behavior was superior to all the other controllers. Only when the traffic was lightly bursty ($C^2 \sim 1$) the *state FB* controller and the *PI–IPFF* showed a little better performance, since the inverse model was relative correct. In the steady–state analysis the *PI–PFB* controller showed to be sensitive to small required work and low arrival rate (that is, sensitive to low load).

A restriction on the response–time reference was put on both of the suggested controllers, which might cause restriction to their use in practical implementations.

The suggested controllers did not rely on any off–line estimation of the required work and showed similar or better performance than the controllers for comparison.

REFERENCES

K-E. Årzén. A simple event-based PID controller. In *Preprints 14th World Congress of IFAC*, Beijing, P.R. China, January 1999.

K.J. Åström. *Introduction to Stochastic Control Theory*. Dower publications. Inc, Mineola, NY, 2006.

K.J. Åström and T. Hägglund. *Advanced PID Control*. ISA - The Instrumentation, Systems, and Automation Society, Research Triangle Park, NC, 2005.

K.J. Åström and B. Wittenmark. *Computer–Controlled Systems*. Printice Hall, Upper Saddle River, NJ, 1997.

J.L. Hellerstein, Y.Diao, S. Parekh, and D.M. Tilbury. *Feedback Control of Computing Systems*. Wiley-Interscience, 2004. ISBN 978-0471266372.

D. Henriksson, Y. Lu, and T. Abdelzaher. Improved prediction for web server delay control. In *Proc. 16th Euromicro Conf. on Real–Time Systems (ECRTS'04)*, Catania, Italy, June 2004.

M. Kihl, A. Robertsson, M. Andersson, and B. Wittenmark. Control-theoretic analysis of admission control mechanisms for web server systems. *The World Wide Web Journal, Springer*, 11(1-2008):93–116, August 2007. online Aug 2007, print March 2008 (DOI 10.1007/s11280-007-0030-0.

M.A. Kjær, M. Kihl, and A. Robertsson. Response–time control of single server queue. In *Proc., 46th IEEE Conf. Decision and Control*, New Orleans, LA, December 2007.

L. Kleinrock. Time–shared systems: A theoretical treatment. *Jon. Association for Computing Machinery*, 14 (2):242–261, April 1967.

L. Kleinrock. *Queuing Systems*. John Wiley & Sons, Inc, New York, 1975. ISBN 0-471-49110-1.

X. Liu, J. Heo, L. Sha, and X. Zhu. Adaptive control of multi-tiered web application using queueing predictor. In *Proc. of the 10th IEEE/IFIP Network Operations and Management Symposium*, Vancouver, Canada, 2006a.

X. Liu, J. Heo, L. Sha, and X. Zhu. Adaptive control of multe–tiered web applications using queuing predictor. In *Proc.* 10^{th} *IEEE/IFIP Network Operation & Mangement Symp.*, Vancouver, Canada, April 2006b.

Y. Lu, T. Abdelzaher, C. Lu, L. Sha, and X. Liu. Feedback control with queuing–theoretic prediction for relative delay guarantees in web servers. In *Proc. 9th IEEE Real–Time and Embedded Technology and Application Symp. (RTAS'03)*, Toronto, Canada, May 2003.

S. Noguahi M. Sakata and J. Oizurnih. An analysis of the M/G/1 queue under round–rubin scheduling. *Operations Research*, 19(2):371–385, Mar.-Apr. 1971.

Z. Wang, X. Liu, A. Zhang, C. Stewart, X. Zhu, T. Kelly, and S. Singhal. AutoParam: Automated control of application-level performance in virtualized server environments. In *Proc. Second IEEE International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID'07)*, pages 2–7, Munich, Germany, 2007.

W. Xu, X. Zhu, S. Singhal, and Z. Wang. Predictive control for dynamic resource allocation in enterprise data centers. In *Proc.* 10^{th} *IEEE/IFIP Network Operation & Mangement Symp.*, Vancouver, Canada, April 2006.