7-2010

Robust fuzzy CPU utilization control for dynamic workloads

Can Basaran

Mehmet H. Suzer

Kyoung-Don Kang
Binghamton University--SUNY, kang@binghamton.edu

Xue Liu

Robust Fuzzy CPU Utilization Control for Dynamic Workloads

Can Basaran, Mehmet H. Suzer, Kyoung-Don Kang*

Department of Computer Science, State University of New York at Binghamton

Xue Liu

School of Computer Science, McGill University

Abstract

In a number of real-time applications such as target tracking, precise workloads are unknown a priori but may dynamically vary, for example, based on the changing number of targets to track. It is important to manage the CPU utilization, via feedback control, to avoid severe overload or underutilization even in the presence of dynamic workloads. However, it is challenge to model a real-time system for feedback control, as computer systems cannot be modeled via physics laws. In this paper, we present a novel closed-loop approach for utilization control based on formal fuzzy logic control theory [1], which is very effective to support the desired performance in a nonlinear dynamic system without requiring a system model. We mathematically prove the stability of the fuzzy closed-loop system. Further, in a real-time kernel, we implement and evaluate our fuzzy logic utilization controller, the PI utilization controller [2], and the model predictive utilization controller [3] for an extensive set of workloads. Our approach supports the specified average utilization set-point, while showing the best transient performance in terms of utilization control among the tested approaches.

1. Introduction

Real-time systems are deployed in mission critical applications such as target tracking, traffic control, and electric grid management where the workload may dynamically vary [4, 5]. For example, the execution times of real-time tasks for target tracking or traffic control may vary significantly when the number of targets or traffic density dynamically changes. In these systems, traditional real-time scheduling techniques [6] requiring *precise a priori* knowledge of the workload are not directly applicable to support timing constraints. Thus, it is critical to continuously measure and control the utilization in a feedback loop to avoid severe underutilization or overload in real-time systems operating in dynamic environments.

Linear PID (proportional, integral, and differential) control techniques [7] have been applied to manage real-time performance in dynamic environments [2, 8]. However, PID controllers and their variants,

*Corresponding Author

Email addresses: {cbasaran,msuzer,kang}@cs.binghamton.edu (Can Basaran, Mehmet H. Suzer, Kyoung-Don Kang), xueliu@cs.mcgill.ca (Xue Liu)

e.g., P, PI, or PD controllers, usually approximate the system dynamics in a piecewise linear fashion [2, 9]. PID controllers are guaranteed to support the set-point only if system dynamics do not deviate from a specific operating range derived offline. If the workload varies dynamically exceeding the operating range, PID controllers and their variants, may largely fail to support the set-point [9].

Model predictive control theory [10] is applied to manage the utilization in dynamic environments by continuously modeling the system behavior online [3, 11]. However, approximate models are often used to reduce the complexity of online predictive modeling of the controlled real-time system. For example, the authors of [3, 11] assume that the actual execution times of real-time tasks are equal to their estimated execution times to decrease the complexity of system modeling. Also, the predictive system model derived online may have non-trivial errors when workloads change fast [12].

In this paper, we apply formal *fuzzy logic control theory* [1] to adapt workloads, if necessary, to make the utilization converge to the specified set-point even given dynamic workloads. Unlike PID and model predictive control techniques, fuzzy control is not tied to a mathematical model of the controlled system or an operating range. Because of the model-free nature of a fuzzy logic controller, there is less risk of introducing design errors due to, for example, statistical inaccuracies existing in a black-box plant model [7, 10].

Rather than relying on an approximate system model, we develop novel fuzzy closed-loop system to control the utilization based on the logical understanding of the relation between the workload and utilization changes. Intuitively, it is clear that the utilization increases as the load increases before it saturates at 1 and vice versa. After the utilization saturates at 1, any further load increase does not affect the utilization. In this paper, we develop a fuzzy logic utilization controller based on the logical understanding of the nonlinear relation between utilization and load changes. We prove the stability of our fuzzy logic controller via the Lyapunov direct method [1, 12]. By leveraging the stability analysis result, we also tune the fuzzy logic controller to avoid repetitive tuning and testing.

For fair and realistic performance evaluation, we extend the Real-Time Application Interface (RTAI) for Linux kernel [13] to implement our fuzzy logic utilization controller (FLC), the PI utilization controller (PIC) designed via an offline piecewise linear approximation of system dynamics [2], and the advanced model predictive utilization controller (MPC) [3]. By performing extensive experiments, we thoroughly compare their performance with each other. Among the tested approaches, the FLC shows the smallest deviation from and the fastest convergence to the specified utilization set-point when the system is in a transient status. Further, it only consumes 0.53% CPU utilization and a small amount of memory to store fuzzy rules and a few control variables.

Despite the effectiveness of fuzzy logic control theory, little prior work has been done to apply it to manage the performance of real-time systems [14, 15]. A summary of the key contributions of this paper follows:

- This paper presents a new closed-loop approach to supporting the specified set-point utilization even in the presence of dynamic workloads. Especially, we directly manage the nonlinear relation between the load and utilization via formal fuzzy logic control theory that is very effective to support the desired performance in nonlinear, dynamic systems [1].

- Unlike the most existing work on fuzzy control of real-time performance [14, 15], we do formal stability analysis to prove that the utilization converges to the specified set-point in our fuzzy

closed-loop system.

- Different from [2, 3, 11, 14] based on simulations, we compare the performance of our approach to PIC and MPC in a real-time kernel. Although Wang et al [16, 17] have implemented and evaluated their approaches based on model predictive control theory [10] for utilization control in a real-time middleware, we are not aware of any prior work that thoroughly compares the performance of fuzzy logic, model predictive, and PI control approaches for performance management in a real-time kernel.

The remainder of this paper is organized as follows. The problem formulation of fuzzy logic control is given in Section 2. The design of our fuzzy closed-loop system is described in Section 3. In Section 4, the stability of our fuzzy logic controller is proved. Performance evaluation results are discussed in Section 5. Our work is compared to the current state of art in Section 6. Finally, we conclude the paper and discuss future work in Section 7.

2. Problem Formulation

In this section, the key objective of our fuzzy closed-loop approach, real-time task model, and QoS adaptation approach taken in this paper are described.

2.1. Objective and Real-Time Task Model

- **Goal:** In this paper, we aim to ensure that the utilization converges to the specified set-point even in the presence of dynamic workloads. In this way, the real-time system controlled by our fuzzy closed-loop scheme is desired to avoid overload or underutilization as much as possible.

- **Average and Transient Performance:** A real-time system operating in a dynamic environment may suffer transient overload or underutilization. Therefore, it is necessary to monitor and control not only the long-term average utilization but also the transient utilization in a closed-loop.

- **Task Model:** In this paper, we assume that there are N periodic real-time tasks in the system. Task τ_i ($1 \leq i \leq N$) is described by (C_i, T_i, $T_{i,min}$, $T_{i,max}$) where C_i is the *estimated* execution time and T_i is the period. Thus, τ_i's estimated utilization $U_i = C_i/T_i$. In this paper, we assume that τ_i's relative deadline $D_i = T_i$. A job τ_{ij} is the j^{th} instance of the periodic task τ_i. We assume that every task starts at time 0. Therefore, τ_{ij}'s absolute deadline $D_{ij} = jT_i$ where $j \geq 1$.

  In our approach, τ_i's period T_i can be adapted at run-time, if necessary, to support the utilization set-point within the specified lower and upper bounds, similar to [18]. Hence, T_i always meets the following condition:

$$T_{i,min} \leq T_i \leq T_{i,max} \tag{1}$$

  where the minimum period, $T_{i,min}$, and maximum period, $T_{i,max}$, are determined by the application of interest. For example, $T_{i,min}$ and $T_{i,max}$ may determine the highest and lowest QoS provided by τ_i for target tracking or traffic monitoring, respectively. Also, based on the relative importance of tasks, different tasks can be assigned different minimum and maximum periods, similar to [18].

- **Workload Estimation:** In this paper, we assume that only the estimated task execution times are known but the accurate execution times are unknown, because it is very difficult, if at all possible, to know precise workloads *a priory* in real-time systems operating in dynamic environments as discussed before. Therefore, we can only compute the estimated load $L = \sum_{i=1}^{N} U_i$.

- **Scheduling:** In this paper, real-time tasks are scheduled in an earliest deadline first (EDF) manner [6]. As the schedulable utilization bound of EDF is 1, a set of real-time tasks are admitted to the system, if $L \leq U_s \leq 1$ where U_s is the utilization set-point. As tasks are admitted and scheduled based on the estimated load, the system can be overloaded (or underutilized), if the execution times are underestimated (or overestimated). Thus, a system administrator is recommended to set $U_s < 1$ to leave headroom to meet as many deadlines as possible even in the presence of dynamic workloads unknown a priori. Note that our approach is not tied to EDF, but it is generally applicable to a class of real-time scheduling algorithms designed to meet deadlines by controlling the CPU utilization to be below the schedulable utilization bound. Thus, another real-time scheduling algorithm such as rate monotonic [6] can be used instead.

2.2. High-Level System Architecture

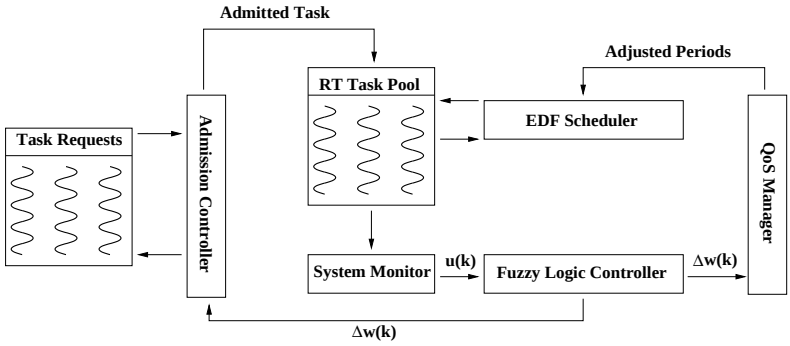


Figure 1: System Architecture

The overall structure of our closed-loop real-time system is shown in Figure 1:

- Upon the arrival of a real-time task τ_i with the *estimated utilization* U_i, the admission controller admits τ_i if $U_i + L \leq U_s$. Otherwise, it rejects τ_i and returns the system busy message to the client that submitted the task. To reduce the overhead for admission control, our admission test is performed on the task basis rather than considering individual task instances. Once a task is admitted, its periodic instances (i.e., jobs) are executed, even though the task execution period can be adapted, if necessary, to support U_s even given dynamic workloads.

- At the k^{th} sampling point, the system monitor measures the current utilization $u(k)$ and provides it to the fuzzy closed-loop controller that computes the required workload adjustment $\Delta w(k)$ to support U_s.

- According to $\Delta w(k)$, the QoS manager adjusts the periods of all the real-time tasks in the system within their specified minimum and maximum period bounds, if necessary, to support U_s. If $\Delta w(k) < 0$ (or $\Delta w(k) > 0$), the periods of all the tasks in the system are increased (or decreased) in proportion to $|\Delta w(k)|$ within their bounds specified by the application administrator. In this way, we aim to avoid an unfair case in which one task's period is increased (or decreased) substantially within its minimum and maximum bounds, while others are not. At the same time, our approach ensures that an important task with a small maximum period receives a higher QoS than the other tasks with the large maximum periods.

- The QoS manager informs the application and scheduler of the potential period adaptation, if any. Thus, the application is aware of potential QoS adaptation. At the same time, the scheduler can schedule the tasks using the updated periods.

2.3. Overview of QoS Adaptation in the Fuzzy Closed-Loop System

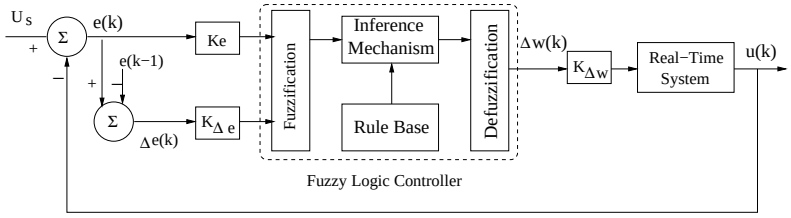


Figure 2: Fuzzy Logic Control System

Figure 2 shows the structure of the fuzzy closed-loop system designed in this paper. Let SP stand for the sampling period for control. We use the same SP for the PIC, MPC and FLC for fair performance comparisons in Section 5. The utilization $u(k)$ is measured at the k^{th} sampling point, i.e., time kSP, for the jobs executed in the k^{th} sampling period, i.e., the time interval $[(k-1)SP, kSP)$.

Given the current utilization $u(k)$, the fuzzy logic controller computes the required workload adjustment to support the utilization set-point U_s such as 0.7. The system is considered to be overloaded, if $u(k)$ exceeds the set-point and vice versa. The fuzzy control signal becomes negative (positive) when the system is overloaded (underutilized). Accordingly, the QoS manager in the real-time system determines how much to increase the task periods under overload and vice versa within certain bounds. A more detailed description of the procedure follows.

In the fuzzy logic closed loop system, the error, $e(k)$ in Figure 2, is defined as follows:

$$e(k) = U_s - u(k) \tag{2}$$

where U_s is the utilization set-point. Also, we monitor the change in error:

$$\Delta e(k) = e(k) - e(k-1) \tag{3}$$

Based on the measured error and change in error, we directly manage the utilization rather than relying on a black-box model that may involve non-trivial statistical errors, if the load changes fast

[7, 10]. Based on $e(k)$ and $\Delta e(k)$, the FLC in Figure 2 computes the required workload adjustment $\Delta w(k)$ for the next sampling period. The fuzzification interface converts $e(k)$ and $\Delta e(k)$ to linguistic values such as negative small (NS) and positive small (PS). The inference mechanism looks up the knowledge base that has IF-THEN rules to find the corresponding control signal. For example, an IF-THEN rule for utilization control may state that if error is NS and change in error is PS, then the control signal is NS. This rule dictates the QoS manager to reduce the load by a small amount. The defuzzification[1] interface converts the linguistic control signal to a crisp control signal $\Delta w(k)$ expressed as a real number such as -0.25. A detailed discussion of fuzzy control is given in Section 3.

Given the control signal $\Delta w(k)$, the QoS manager computes the period adaptation factor $F_e(k+1)$ for the next sampling period:

$$F_e(k+1) = F_e(k) \cdot (1 - K_{\Delta w} \Delta w(k)) \tag{4}$$

Note that the control signal in Eq 4, i.e. $\Delta w(k)$, is derived based on the potentially nonlinear relationship between the load and utilization as described before. $K_{\Delta w}$ in Eq 4 is the control gain that needs to be tuned to support the stability of the closed-loop system. (The stability of our closed-loop system is analyzed in Section 4.)

As the control signal $\Delta w(k)$ is inverted in Eq 4, $F_e(k+1) > F_e(k)$ and the periods of real-time tasks will be increased to reduce the utilization if $\Delta w(k) < 0$ due to overload conditions and vice versa. If the system is overloaded at the k^{th} sampling point, the period of τ_i $(1 \leq i \leq N-1)$ is increased for the next sampling period; that is, $T_i(k+1) > T_i(k)$. Thus, the estimated load L is decreased by $\frac{C_i}{T_i(k+1) - T_i(k)}$. Assuming the tasks are sorted in descending order of the importance, QoS adaptation is applied to τ_{i+1} and the next task(s) until the sum of the estimated load adaptation becomes equal to $K_{\Delta w} \Delta w(k)$ or no task period can be increased any further. Similarly, task periods are decreased according to the control signal, if the system is underutilized.

Using the adaptation factor, the QoS manager in the real-time system computes:

$$\widehat{T}_i(k+1) = T_i(k) \cdot F_e(k+1) \tag{5}$$

for an arbitrary task τ_i in the real-time system and determines τ_i's period for the $(k+1)^{th}$ sampling period, $T_i(k+1)$, as follows:

$$T_i(k+1) = \begin{cases} \widehat{T}_i(k+1) & \text{if } T_{i,min} \leq \widehat{T}_i(k+1) \leq T_{i,max} \\ T_{i,min} & \text{if } \widehat{T}_i(k+1) < T_{i,min} \\ T_{i,max} & \text{if } \widehat{T}_i(k+1) > T_{i,max} \end{cases} \tag{6}$$

Given $\Delta w(k)$, the QoS manager in Figure 1 increase or decrease the period of **every** task in the system according to Eq 5 and Eq 6, if necessary, to support the utilization set-point via QoS adaptation that is fair to every task as discussed in Section 2.2. Further, the QoS manager informs the scheduler and application of the new periods (as described in Section 2.2). Hence, the required workload to support

[1] Fuzzification and defuzzification are standard terms in fuzzy control theory [1].

utilization set-point U_s for the next period is calculated as:

$$w(k+1) = w(k) + K_{\Delta w}\Delta w(k) \tag{7}$$

From Eq 1 and Eq 6, we observe that it may not always be possible to adapt the task period as much as indicated by $K_{\Delta w}\Delta w(k)$. This is especially a problem when the system is currently overloaded and no task period can be extended anymore. In this case, newly incoming tasks, if any, are rejected. Also, the least important tasks in the system are temporarily suspended to fully enforce the control signal. In the $(k+1)^{th}$ sampling period, the admission controller in Figure 1 will accept incoming tasks, if the sum of the estimated utilization values of the tasks arriving at and already in the system does not exceed U_s as discussed before.

In this paper, a certain load that lets the system to converge to the set-point is called the convergent load W. The difference between W and the current workload is formulated as:

$$\tilde{w}(k) = W - w(k) \tag{8}$$

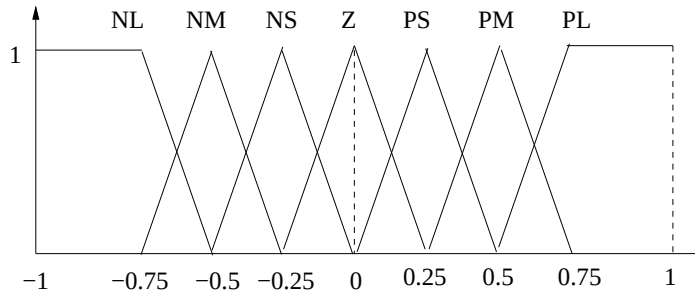
In reality, W is unknown and it may vary in time depending on execution time estimation errors. Thus, the purpose of fuzzy control is to adapt the workload based on $e(k)$ and $\Delta e(k)$ to support U_s by minimizing $|\tilde{w}(k)|$, i.e., the absolute value of $\tilde{w}(k)$.

3. Fuzzy Logic Control

In this section, the key components of the FLC and control signal computation process are described. Further, a detailed discussion of our rule-base design is given.

3.1. Fuzzy Logic Control Components for Control Signal Derivation

In this subsection, we describe standard fuzzy control terminologies [1] and describe how to derive the control signal.



NL: Negative Large, NM: Negative Medium, NS: Negative Small,
Z: Zero, PS: Positive Small, PM: Positive Medium, PL: Positive Large

Figure 3: Input/Output Membership Functions

The universe of discourse is the domain of an input (output) to (from) the FLC [1]. Figure 3 shows the universe of discourse for the utilization error, change in error, and control output. In this paper,

		$\Delta e(k)$						
		NL	**NM**	**NS**	**ZE**	**PS**	**PM**	**PL**
$e(k)$	**NL**	NL	NL	NL	NL	NM	NS	ZE
	NM	NL	NL	NL	NM	NS	ZE	PS
	NS	NL	NL	NM	NS	ZE	PS	PM
	ZE	NL	NM	NS	ZE	PS	PM	PL
	PS	NM	NS	ZE	PS	PM	PL	PL
	PM	NS	ZE	PS	PM	PL	PL	PL
	PL	ZE	PS	PM	PL	PL	PL	PL

Table 1: Fuzzy Rule-Base

the universe of discourse for $e(k)$ and $\Delta e(k)$ is $[-1,1]$, while the universe of discourse for the control output is set to [-0.75, 0.75] to bound the range of the control signal.

Linguistic variables describe the input/output variables in fuzzy control. For instance, two inputs to the fuzzy controller at time *kSP* are *error*, i.e., fuzzified $e(k)$, and *change in error*, i.e., fuzzified $\Delta e(k)$. Also, the output from the FLC is called *control signal*−the required workload adjustment expressed linguistically.

Linguistic variables are associated with linguistic values to describe characteristics of the variables. A linguistic variable *error*, for example, could be associated with linguistic values Large, Small, or Zero at a sampling point. Figure 3 shows linguistic values for the linguistic variables *error*, *change in error*, and workload *control signal* used in this paper.

A set of IF *premise* THEN *consequent* linguistic rules are used to map the inputs to output(s) of a FLC. For example, if *error* $= NL$ and *change in error* $= NM$ at the k^{th} sampling point, i.e., time *kSP*, then the system is overloaded and the degree of overload is increasing considerably according to Eq 2 and Eq 3. Thus, the corresponding rule in Table 1 generates a NL signal that dictates the real-time system to significantly reduce the load to achieve U_s. The *rule-base* in Table 1 has a set of IF-THEN rules stating how to achieve the utilization set-point according to the current *error* and *change in error*. (The design of the rule-base in Table 1 is discussed in Section 3.2).

A membership function (MF) in Figure 3 quantifies the *certainty* an $e(k)$, $\Delta e(k)$, or $\Delta w(k)$ value to be associated with a specific linguistic value. Specifically, the horizontal axis of Figure 3 represents $e(k)$, $\Delta e(k)$, or $\Delta w(k)$, while the vertical axis indicates the membership value. For MFs (except for the leftmost or rightmost ones), we use symmetric triangles of an equal base and 50% overlap with adjacent MFs, similar to [19, 1].

Unlike traditional set theory, in fuzzy set theory underlying fuzzy control theory, set membership is not binary but continuous to deal with uncertainties [20, 21, 1]. Thus, a fuzzy input or output may belong to more than one sets−maximum two adjacent sets in Figure 3−with different certainty values. For example, if $e(k) = -0.25$, then $e(k)$ belongs to the fuzzy set NS in Figure 3 with certainty 1, which is expressed as: $\mu_{NS}(-0.25) = 1$. If $\Delta e(k) = 0.0625$, $\mu_{ZE}(0.0625) = 0.75$ and $\mu_{PS}(0.0625) = 0.25$.

Based on the fuzzified $e(k)$ and $\Delta e(k)$, the inference mechanism in Figure 2 determines which rules to apply at the k^{th} sampling point. Thus, in the previous example, the IF-THEN rules, rule(NS,ZE) = NS and rule(NS, PS) = ZE, in Table 1 apply. To compute the certainty value(s) of the corresponding IF

premise THEN *consequent* rule(s), we take the minimum between the certainty values of the premise, i.e., $e(k)$ and $\Delta e(k)$, because the consequent cannot be more certain than the premise [1, 19, 22]. Thus, $\mu(NS,ZE) = min\{1, 0.75\} = 0.75$ and $\mu(NS,PS) = min\{1, 0.25\} = 0.25$ in the previous example.

Note that maximum four rules apply at a sampling point, since the error or change in error can belong to up to two MFs in Figure 3. Thus, the worst case time complexity of our fuzzy logic control is O(1). Also, storing the rule-base (Table 1) consumes little memory.

Finally, the control signal is computed via defuzzification. Let i and j ($1 \leq i, j \leq 7$) represent the row and column indexes in Table 1. Further, let $\mu(i, j)$ denote the certainty of the corresponding $rule(i, j)$ in the table derived as described before and let $c(i, j)$ denote the center of the MF of the $rule(i, j)$'s consequent. For triangle MFs, the center is the middle of the triangle's base and the fuzzy utilization control output is [1]:

$$\Delta w(k) = \frac{\sum_{i,j} c(i, j) \cdot \mu(i, j)}{\sum_{i,j} \mu(i, j)} \tag{9}$$

In Figure 3, the center of NS and ZE is -0.25 and 0.0, respectively. Thus, in the previous example, $\Delta w(k) = ((-0.25) \cdot 0.75 + (0.0) \cdot 0.25)/(0.75 + 0.25) = -0.1875$.

3.2. Fuzzy Rule-Base Design

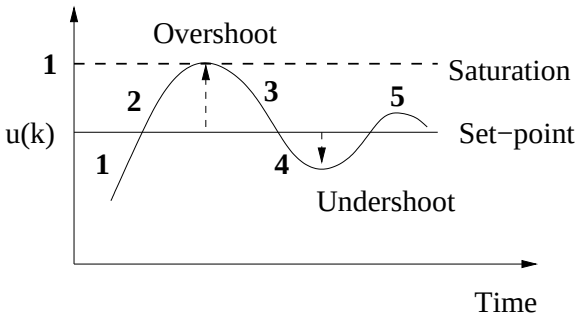


Figure 4: Utilization Control Characteristics

As shown in Figure 4, there are five zones that characterize dynamic real-time system's behaviors from which we derive the rule-base for utilization control in Table 1.

Zone 1. $e(k) \geq 0$ and $\Delta e(k) \leq 0$: In this zone, the actual utilization is smaller than the set point, but it comes closer to the set point[2]. The control signal to be applied is carefully determined by comparing the magnitude of "$e(k)$" and "$\Delta e(k)$" where "$e(k)$" and "$\Delta e(k)$" represent the fuzzified $e(k)$ and $\Delta e(k)$, since the current workload may be lower than, equal to, or higher than the convergent load W.

- If $|"e(k)"| > |"\Delta e(k)"|$ then $\tilde{w}(k) \geq 0$ in Eq. 8; that is, the current load is lower than W. For example, if "$e(k)$" $\in PM, PL$ and "$\Delta e(k)$" $\in NS$, then the current load is lower than W. In this

[2]Note that, in Zones $1-4$, $e(k)$ and $\Delta e(k)$ are not both zero at the same time. In only Zone 5, $e(k)$ and $\Delta e(k)$ can be zero at the same time.

case, the utilization is increasing too slow. Thus, the controller should apply a positive signal to further increase the load. As a result, $\Delta w(k) > 0$.

- If $|"e(k)"| = |"\Delta e(k)"|$, then $\tilde{w}(k) = 0$ in Eq. 8. For example, if $"e(k)" \in PS$ and $"\Delta e(k)" \in NS$, then the current load is equal to W ($\tilde{w}(k) = 0$). Thus, $\Delta w(k) = 0$.
- If $|"e(k)"| < |"\Delta e(k)"|$, then $\tilde{w}(k) < 0$. For example, if $"e(k)" \in PS$ and $"\Delta e(k)" \in NM, NL$, then the current load is higher than W. In this case, the utilization increases too fast. Thus, the controller applies a negative signal, $\Delta w(k) < 0$, to avoid an overshoot.

Zone 2. $e(k) < 0$ and $\Delta e(k) \leq 0$: In this zone, the utilization is higher than the set-point and it is further increasing. It indicates that the current load is higher than W; that is, $\tilde{w}(k) < 0$. Hence, the controller applies $\Delta w(k) < 0$ to reverse the current trend.

Zone 3. $e(k) \leq 0$ and $\Delta e(k) \geq 0$: In this zone, the utilization is higher than the set point, but it comes closer to the set point. The control signal should be carefully determined by comparing the magnitude of $"e(k)"$ and $"\Delta e(k)"$ as the current workload value may be lower than, equal to, or higher than W value.

- If $|"e(k)"| > |"\Delta e(k)"|$, then $\tilde{w}(k) < 0$. For example, if $"e(k)" \in NM, NL$ and $"\Delta e(k)" \in PS$ then the current load is higher than W; that is, $\tilde{w}(k) < 0$. As the utilization is decreasing too slow, the controller should apply a negative signal to further reduce the load.
- If $|"e(k)"| = |"\Delta e(k)"|$, then $\tilde{w}(k) = 0$. For example, if $"e(k)" \in NS$ and $"\Delta e(k)" \in PS$, then the current load is equal to W. Thus, $\Delta w(k) = 0$.
- If $|"e(k)"| < |"\Delta e(k)"|$, then $\tilde{w}(k) > 0$. For example, if $"e(k)" \in NS$ and $"\Delta e(k)" \in PM, PL$, then the current load is lower than W. The utilization is decreasing too fast in this case. Thus, the controller should apply a positive signal to increase the load to support U_s, i.e., $\Delta w(k) > 0$.

Zone 4. $e(k) > 0$ and $\Delta e(k) \geq 0$: In this zone, the actual utilization is lower than the set-point and it is further decreasing. It indicates that the current workload is lower than W, i.e., $\tilde{w}(k) > 0$. Thus, $\Delta w(k) > 0$.

Zone 5. $|e(k)| \leq \varepsilon$ and $|\Delta e(k)| \leq \varepsilon$ where ε is a small predefined real number: In this case, the real-time system is in the steady state. $\Delta w(k) = 0$, as the current workload is equal to W, i.e., $\tilde{w}(k) = 0$. In Section 4, we prove that the fuzzy closed-loop system asymptotically convergences to the ε neighborhood of the set-point.

To summarize, the relationship between the control output and inputs in Table 1 can be formulated in linguistic terms:

$$"\Delta w(k)" = "e(k)" + "\Delta e(k)"$$

The linguistic value of $"\tilde{w}(k)"$ can be determined from these five zones. Our fuzzy logic rule-base containing the five zones implies the following linguistic equation:

$$"\tilde{w}(k)" = "\Delta w(k)" \tag{10}$$

which can be validated by inspecting the rule base and explanation of the fuzzy control actions in the five zones. In our rule-base, the sign of $\Delta w(k)$ is equal to the sign of $\tilde{w}(k)$. This is because, in each zone, the sign of $\Delta w(k)$ is determined based on the sign of $\tilde{w}(k)$ as discussed earlier in this subsection. Also, the control signal's magnitude is proportional to the difference between W and current load.

4. Stability Analysis and Tuning

In this section, the stability of the closed-loop system is analyzed and the control gains, i.e., $Ke, K_{\Delta e}$ and $K_{\Delta w}$ in Figure 2, are tuned.

4.1. Stability Analysis

In this paper, we prove the stability of our fuzzy closed-loop system via the Lyapunov Direct Method [12, 1].

Theorem 4.1 *Lyapunov Direct Method [12, 1]. If the following conditions are true for an arbitrary function $V(x(k)) : R^n \rightarrow R$ where $n \geq 1$,*

$$\begin{aligned} &V(x(k)) = 0, \; if \; x(k) = 0 \\ &V(x(k)) > 0, \; if \; x(k) \in R^n - \{0\} \\ &V(x(k+1)) - V(x(k)) < 0 \end{aligned}$$

then $V(x(k))$ is a Lyapunov candidate function (LCF) in some region $D \in R^n$ which contains the origin. $V(x(k))$ guarantees the asymptotic stability around zero. (Any nonzero equilibrium point can be transformed to the origin via change of variables.)

We apply Theorem 4.1 to prove the stability of our closed loop fuzzy control system. Specifically, we choose the LCF function as:

$$V(\tilde{w}(k+1)) = \tilde{w}^{\,2}(k+1). \tag{11}$$

Theorem 4.2 *If the $V(\tilde{w}(k+1))$ has the LCF function properties, then the closed loop fuzzy control system is asymptotically stable around the set point.*

Proof The LCF function has the following properties:

$$\begin{aligned} &V(\tilde{w}(k+1)) = 0, \; if \; \tilde{w}(k+1) = 0 \\ &V(\tilde{w}(k+1)) > 0, \; if \; \tilde{w}(k+1) \in R - \{0\} \end{aligned}$$

To meet all the requirements to be a LCF, this function should also have the following property:

$$V(\tilde{w}(k+1)) - V(\tilde{w}(k)) = \tilde{w}^{\,2}(k+1) - \tilde{w}^{\,2}(k) < 0 \tag{12}$$

Using Eq. 7 and Eq. 8, we get:

$$\begin{aligned}\tilde{w}(k+1) &= W - w(k+1) \\ &= W - [w(k) + K_{\Delta w}\Delta w(k)] \\ &= W - w(k) - K_{\Delta w}\Delta w(k) \\ &= \tilde{w}(k) - K_{\Delta w}\Delta w(k) \end{aligned} \tag{13}$$

From Eq. 12 and Eq. 13, we derive that:

$$\begin{aligned}V(\tilde{w}(k+1)) - V(\tilde{w}(k)) &= [\tilde{w}(k) - K_{\Delta w}\Delta w(k)]^2 - \tilde{w}^{\,2}(k) \\ &= K_{\Delta w}\Delta w(k)\,[K_{\Delta w}\Delta w(k) - 2\tilde{w}(k)] < 0\end{aligned}$$

To ensure this inequality, the following constraints should be met:

$$sign(\tilde{w}(k)) = sign(\Delta w(k)) \tag{14}$$

$$|\Delta w(k)| < \frac{2}{K_{\Delta w}}|\tilde{w}(k)| \tag{15}$$

The first constraint (Eq. 14) is met, since "$\tilde{w}(k)$" = "$\Delta w(k)$" (Eq. 10). As W and thus $\tilde{w}(k)$ are not measurable directly, we can change the second constraint (Eq. 15) by replacing $\tilde{w}(k)$ with a small positive real number ε:

$$|\Delta w(k)| < \frac{2}{K_{\Delta w}}\varepsilon,\ \varepsilon \in R^{+} \tag{16}$$

If this inequality holds for $\tilde{w}(k) \geq \varepsilon$, then $w(k)$ will asymptotically converge to an ε neighborhood of the convergent load. Specifically, $0 < K_{\Delta w} < 2/0.75$ since $\Delta w(k) = [-0.75, 0.75]$ as discussed in Section 3.1. This concludes the proof of the stability of our fuzzy closed-loop system. ∎

4.2. Fuzzy Controller Tuning

We need to tune $Ke, K_{\Delta e}$ and $K_{\Delta w}$ in Figure 2 for good performance. To support the stability of the fuzzy closed-loop system, we must meet the condition that $0 < K_{\Delta w} < 2.6$ as derived in Theorem 4.2. $K_{\Delta w}$ of a larger value reduces the settling time, but it may cause a higher overshoot. In this paper, we set $K_{\Delta w} = 1$ to balance the settling time and overshoot, while focusing slightly more on reducing potential overshoots. Generally, $K_{\Delta w}$ has the largest effect on the system performance, because it directly affects the stability in addition to the settling time and overshoot. On the other hand, K_e and $K_{\Delta e}$ do not directly affect the stability according to Theorem 4.2. In this paper, K_e is set to 1 so that the controller can utilize the whole rule base for the error input. On the other hand, we set $K_{\Delta e} = 0.1$ to damp potentially jittery change-in-error values. Generally, a large $K_{\Delta e}$ reduces the settling time, but increases the overshoot.

5. Performance Evaluation

In this section, a description of the experimental set-up for evaluating the FLC, MPC, and PIC is given. Also, the performance evaluation results are discussed.

5.1. Experimental Settings

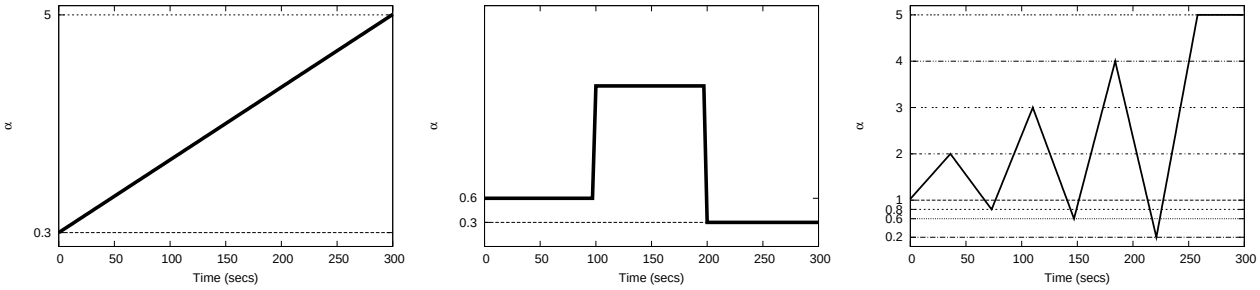
We have implemented the FLC, MPC, and PIC in the RTAI 3.6 [13]. The Linux kernel version 2.6.22 is installed on a 2.3GHz Pentium 4 machine with 1 GB RAM. We have modified the real-time scheduler provided by RTAI to collect performance statistics and implement the controllers. We have implemented and tuned the PIC as described in [2]. Also, we have implemented the MPC described in [3] with the prediction horizon $P = 2$ and control horizon $M = 1$.

Name	**Value**
Set-point (U_s)	0.7
Sampling period (SP)	1 second
Algorithm	EDF
Deadline semantics	Firm
Run length	300 seconds
Runs per load profile	10
Load profiles	Ramp, Step & Sawtooth

Table 2: System parameters

As described in Section 2, all the controllers output the period adaptation factor F_e in Eq. 4 used to adapt the periods of real-time tasks in the system. Each controller is invoked at every sampling point to compute the required workload adjustment to support the utilization set-point U_s. In this paper, SP is set to 1s and U_s is set to 0.7 as shown in Table 2.

Tasks are scheduled according to the EDF (Earliest Deadline First) algorithm. The deadlines are firm; that is, a task instance is canceled as soon as it misses its deadline. Each job is associated with an actual execution time: $AET_{ij} = \alpha \cdot EET_{ij}$ where EET_{ij} is the estimated execution time of job τ_{ij} in the system and α is the *execution time factor*, similar to [2, 3]. In this way, fair performance comparisons are possible among the PIC, MPC, and FLC.



(a) Ramp (b) Step (c) Sawtooth

Figure 5: Tested Workloads

Note that the scheduler and controllers are unaware of actual execution times. When $\alpha > 1$, they may underestimate execution times. As a result, they may overload the system, missing deadlines. On the other hand, when $\alpha < 1$, they may underutilize the system. Thus, we evaluate how closely the

FLC, MPC, and PIC can support U_s when α varies. To this end, we have created several different experimental load profiles summarized in Table 2. For each profile, 10 runs are executed and the average of the 10 runs is reported. Each run executes a random task set for 300s.

In Figure 5, the *ramp* load continuously increases as α increases from 0.3 to 5 over 300s. The *step* load tests the robustness of the controller given a sudden load increase and decrease in a step manner. There are five variations of the step load. Each of them starts with $\alpha = 1$ and an initial load of 60%. At 100s, α is increased to 2, 3, 4, and 5 for Step-2, Step-3, Step-4, and Step-5, respectively. Further, α is decreased to 0.3 at 200s. The ramp and step workloads are widely used to evaluate control performance [7, 9, 2, 3]. We use them for fair performance comparisons between the fuzzy controller and the PIC [2] and MPC [3]. In addition, we consider the *sawtooth load* that concatenates multiple ramp loads to stress the real-time system by increasing or decreasing α at a constant rate.

At the beginning of an experimental run, each task τ_i runs at its minimum period $T_{i,min}$. To satisfy Eq 1, the maximum period of a task τ_i is:

$$T_{i,max} = xT_{i,min} \tag{17}$$

For the set of experiments presented in Sections 5.2.1 $-$ 5.2.3, we set $x = 4$ and use the ramp, step, and sawtooth workloads described above. For the experiments presented in Section 5.2.4, we fix α to 2 and randomly choose x in Eq 17 in the range $[2,6]$ for each task. Also, we increase the number of tasks by six times at 100s in a step manner. This workload is called TASKx6 workload in this paper. In Section 5.2.5, we use a different workload, called Step5-Random, where x for each task is randomly selected in the range $[2,6]$. Moreover, α is abruptly increased to 5 at 100s and decreased to 0.3 at 200s to further stress-test the performance of the tested approaches to utilization management.

5.2. Experiment Results

In this section, the performance evaluation results of the FLC, MPC, and PIC for the ramp, step, and sawtooth workloads are discussed. In our experiments, all the tested approaches admitted all real-time tasks, because the *estimated total utilization* computed based on the estimated execution times is smaller than 1$-$the schedulable utilization bound of the EDF scheduling algorithm [6]. Also, no task was suspended in this paper.

In our experiments, all the tested closed-loop approaches successfully supported the *average* utilization set-point for most of the experiments by adapting task periods according to the feedback control signal as directed in Eq 6. Therefore, we focus on the *transient* performance results in the following. Note that it is critical to manage not only the long-term average but also transient performance in a mission-critical real-time system.

5.2.1. Ramp Workload

The results for the ramp load are given in Figure 6. The PIC has non-zero steady state errors that do not decay until the end of the experiment at 300s as shown in Figure 6. Thus, we observe that the PIC clearly fails to support the set-point. In contrast, the MPC cancels an initial utilization overshoot. MPC's settling time is approximately 40s. As shown in Figure 6, the FLC's settling time is only about 10s. Also, it shows the smallest overshoot. From these results, we observe that the FLC achieves the best performance among the tested approaches for the ramp load.

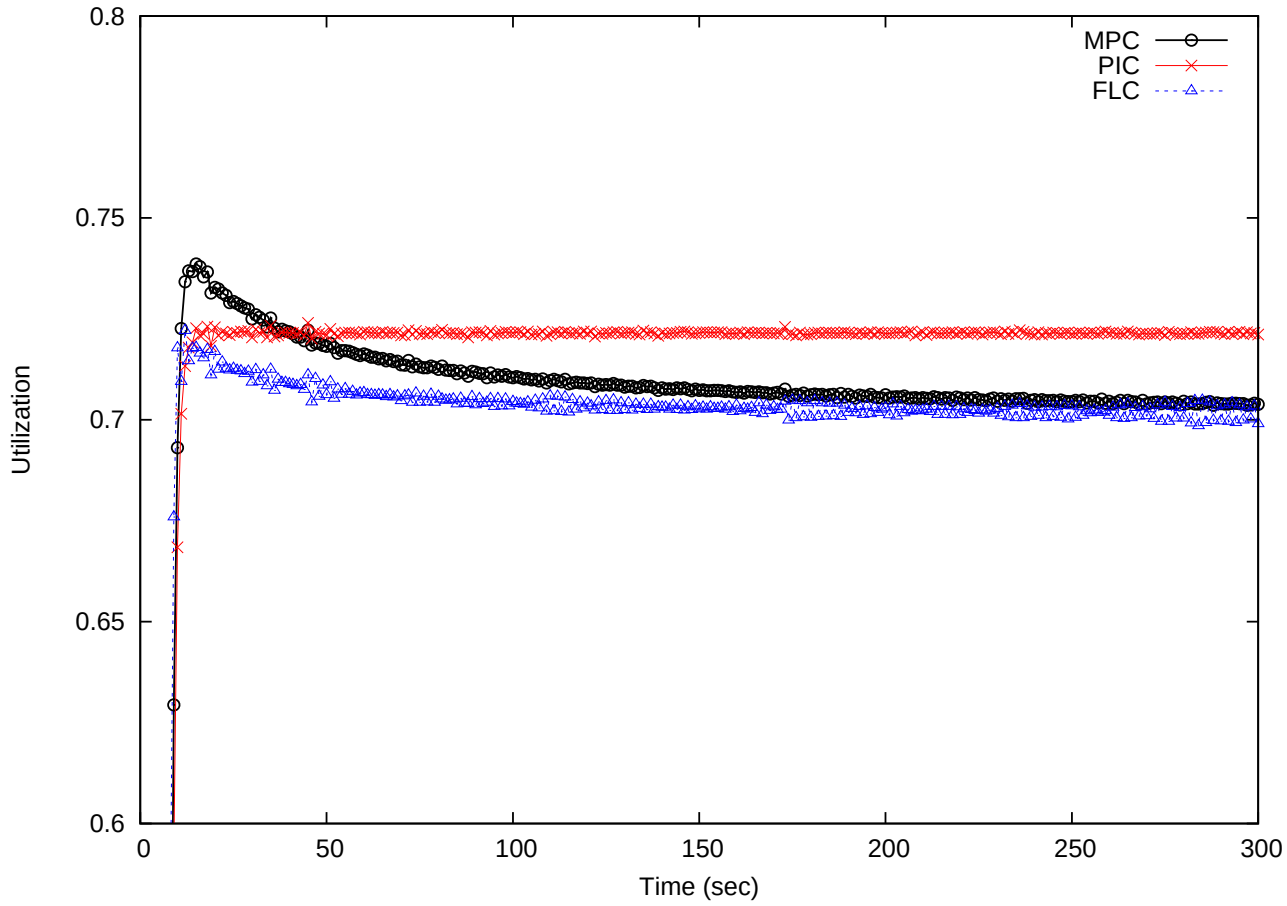


Figure 6: Ramp Results

To further analyze the set-point tracking performance, we define the aggregated error E_{agg}:

$$E_{agg} = \sqrt{\frac{1}{n}\sum_{k=1}^{n}(U_s - u(k))^2} \tag{18}$$

where n is the number of the sampling points in one experimental run. For the ramp workloads, the FLC reduces E_{agg} by 56% and 74% compared to the MPC and PIC. Specifically, $E_{agg} = 0.0056$ for the FLC, while $E_{agg} = 0.0127$ and $E_{agg} = 0.0215$ for MPC and PIC, respectively. Overall, the FLC supports the smallest deviations from the set-point and shortest settling times for the ramp load.

For all the tested workloads, the FLC, MPC, and PIC adjusts the task periods in a similar fashion. The average period adaptations achieved by them are almost equal. However, the transient period adaptation of the FLC is faster than the others. This result shows the higher adaptivity of the FLC to dynamic workloads.

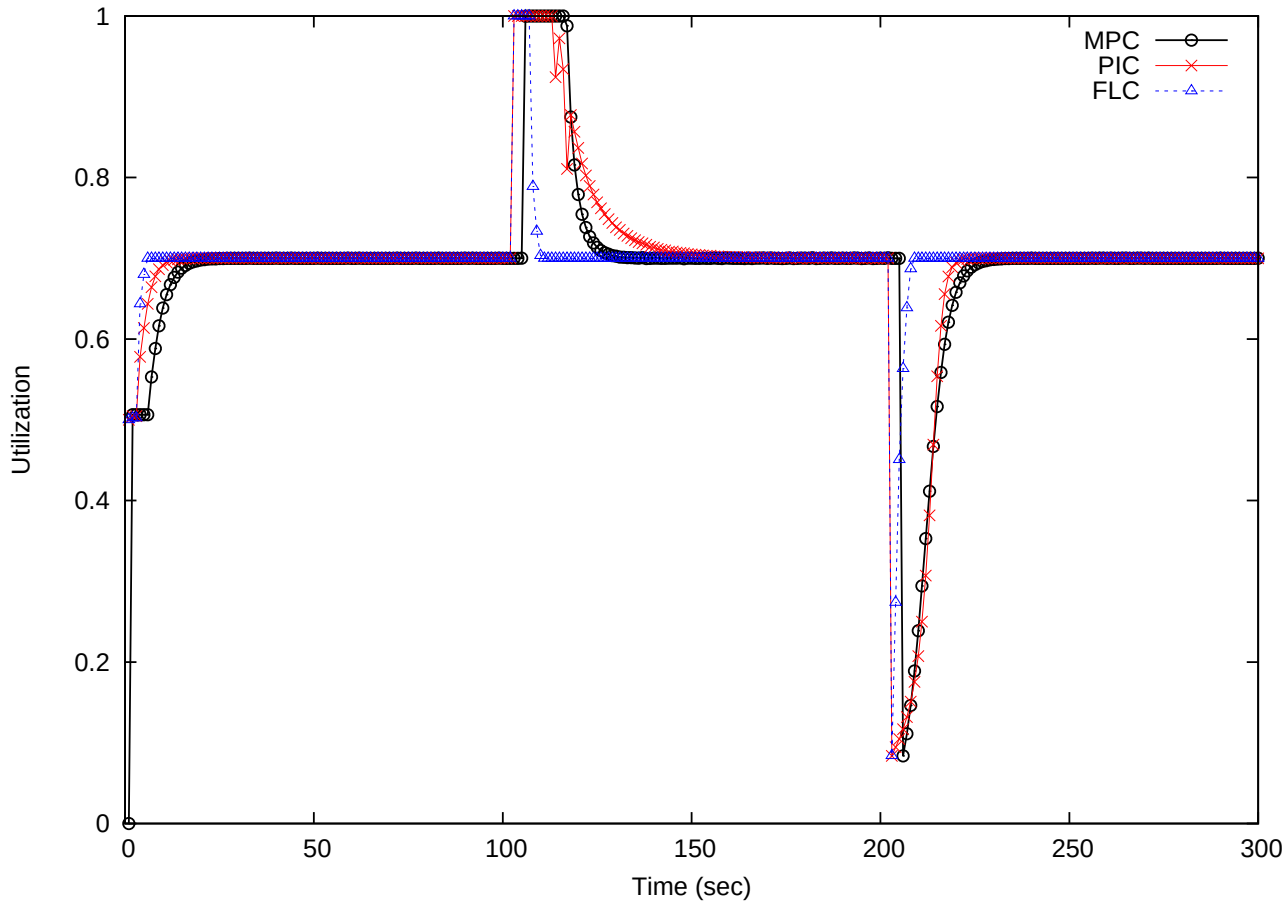


Figure 7: Step-5 Results

5.2.2. Step Workloads

For the clarity of presentation without repetitive discussions, we only show the results for the Step-5 load under which the real-time system is most stressed. Figure 7 shows the results for the Step-5 load that tests the robustness of the controllers against abrupt changes in task execution times. Since the α value suddenly jumps from 1 to 5 at 100s, all the tested approaches show utilization overshoots. As a result, the utilization saturates at 1 at 100s in Figure 7. However, the FLC's settling time is only about 7s as shown in Figure 7, which is less than half the settling time of the MPC. The FLC's settling time is approximately seven times shorter than the PIC's settling time of 55s.

Further, the FLC achieves the smallest E_{agg}. Specifically, $E_{agg} = 0.0611, 0.0714$, and 0.1227 for the FLC, MPC, and PIC, respectively. Thus, the FLC reduces E_{agg} by approximately 50% compared to the PIC. Furthermore, it reduces E_{agg} by more than 14% compared to the MPC with the less complex controller design than the MPC.

5.2.3. Sawtooth Workload

The performance results for the sawtooth load are shown in Figure 8. Similar to the ramp and step load results, the FLC shows the most reliable performance.

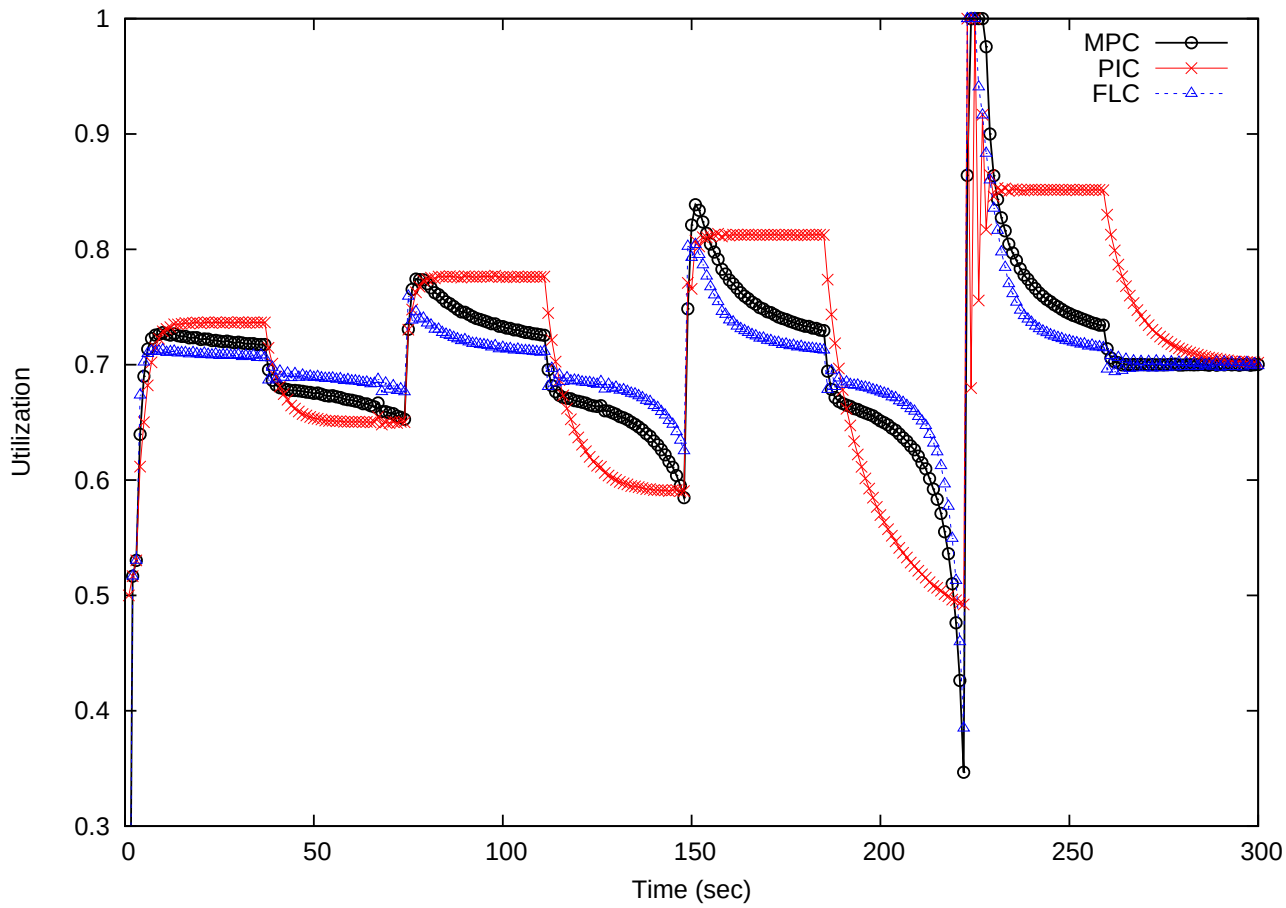


Figure 8: Sawtooth Results

In Figure 8, the FLC shows the substantially smaller utilization overshoots and undershoots than the MPC and PIC. The FLC achieves the fastest convergence to the set-point, even though it is difficult to numerically compare the settling time of the tested approaches due to the highly dynamic system behavior as shown in Figure 8.

Moreover, $E_{agg} = 0.0568, 0.073,$ and 0.0994 for the FLC, MPC, and PIC. Thus, the FLC reduces E_{agg} by more than 22% and 42% compared to the MPC and PIC, respectively. These results demonstrate the effectiveness and robustness of fuzzy control.

5.2.4. TASKx6 Workload

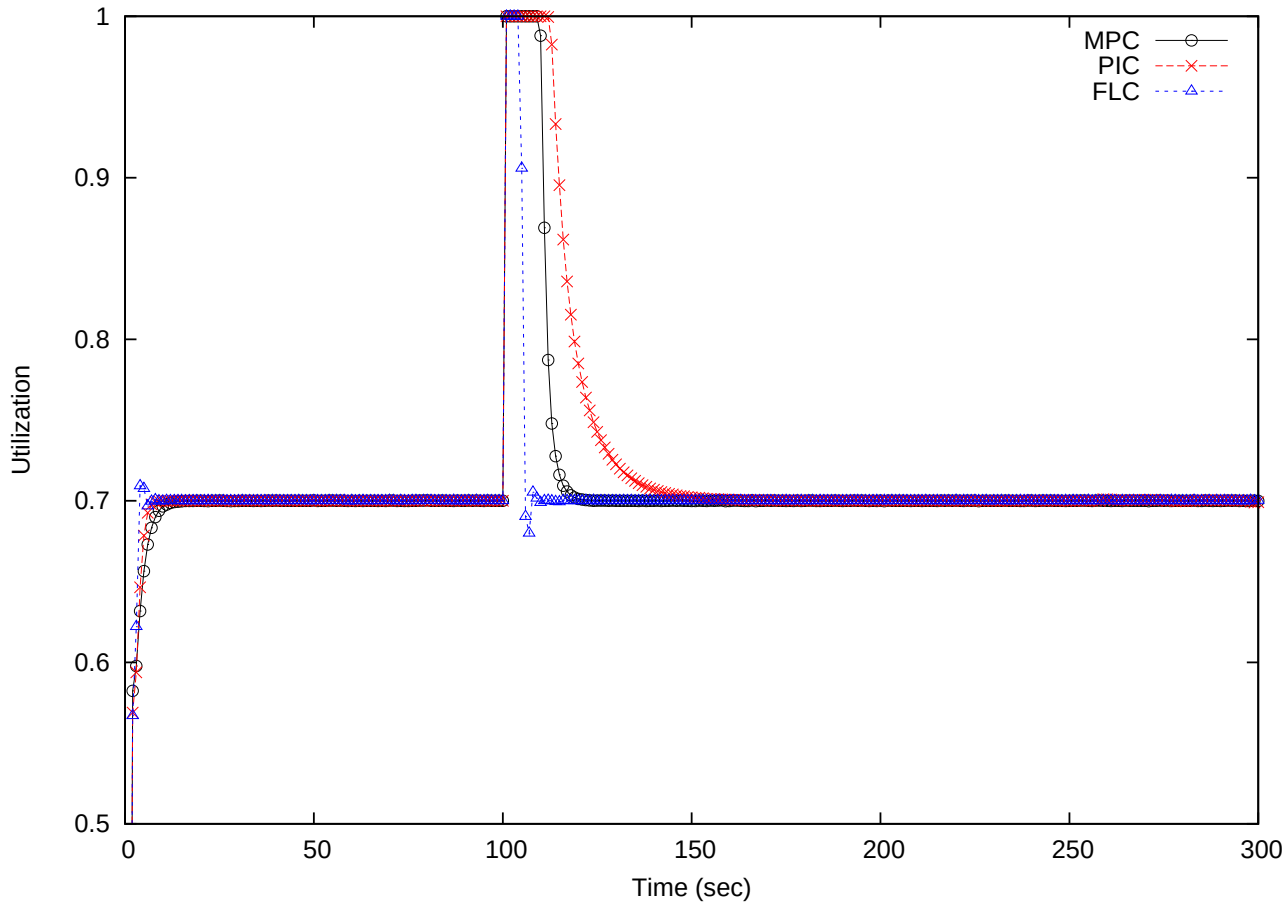


Figure 9: TASKx6 Results

In the TASKx6 workload, we abruptly increase the number of real-time tasks in the system rather than increasing the α value, which is kept fixed at $\alpha = 2$. The number of tasks in the system is increased by 6 times at 100s. As a result, the load increases from 70% to 420% at 100s. Also, the x value in Eq. 17 is randomly selected within the range $[2, 6]$ for each task. As shown in Figure 9, FLC improves the settling time by 29% and 72% compared to MPC and PIC, respectively. Also, it decreases E_{agg} by 17% and 32%, while reducing the total number of deadline misses by 50% and 46% over MPC and PIC, respectively.

5.2.5. Step5-Random Workload

In this experiment, x value is randomly selected in the range [2, 6]. In addition, α is increased to 5 at 100s and reduced to 0.3 at 200s. The results are presented in Figure 10. FLC decreases the settling time by 63% and 84%, while reducing E_{agg} by 29% and 35% over MPC and PIC, respectively. Further, FLC reduces the number of total deadline misses by 59% and 52% compared to MPC and PIC.

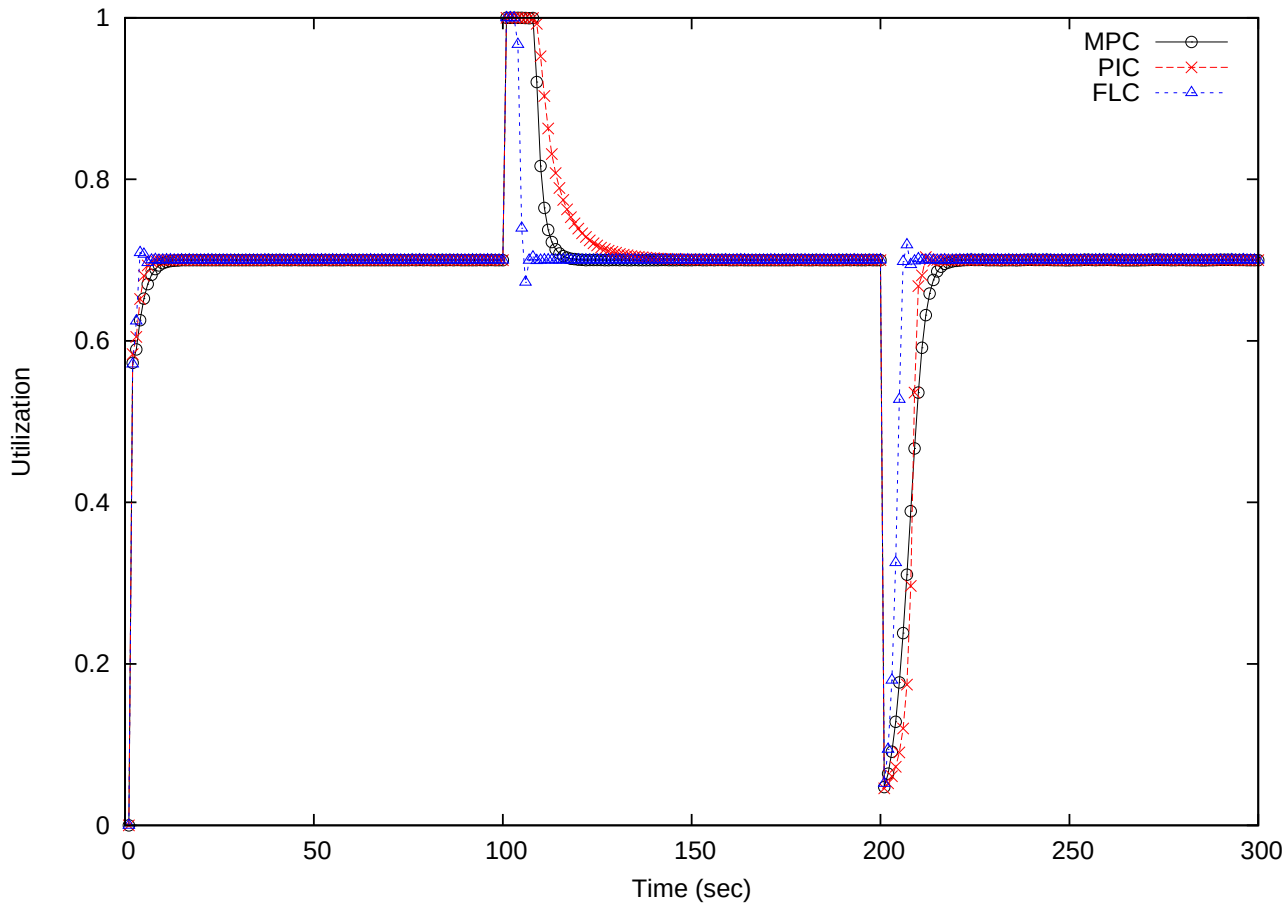


Figure 10: Step5-Random Results

Overall, the FLC achieves the most robust control performance based on the logical understanding of the system behavior requiring no mathematical modeling of the underlying controlled system, which is tied to an operating range or subject to modeling errors due to simplified approximations or online/offline statistical modeling errors. Especially, the FLC is more robust than the PIC and MPC when the load changes fast.

Table 3 summarizes the settling time, E_{agg}, and total number of deadline misses normalized to the FLC. In this way, we measure the performance of the tested approaches in terms of the accuracy of set-point tracking, the timeliness of system adaptation under overload or underutilization conditions, and completion ratio. We consider these features, because they are very important to avoid and recover quickly from overload or underutilization conditions, if any, in a real-time system. From Table 3, we

Load	Approach	Norm. Settling Time	Norm. Error	Norm. Total Misses
Ramp	FLC	1 (10s)	1 (0.0056)	0
	MPC	4	2.27	0
	PIC	1.1 (persistent e_{ss})	3.84	0
Step 2	FLC	1 (4s)	1 (0.0415)	1 (2716)
	MPC	1.25	1.12	2.60
	PIC	3.25	1.4	1.31
Step 3	FLC	1 (6s)	1 (0.0490)	1 (6994)
	MPC	1.33	1.21	2.65
	PIC	3.66	1.72	1.87
Step 4	FLC	1 (7s)	1 (0.0559)	1 (9477)
	MPC	1.29	1.19	2.54
	PIC	4.28	1.88	1.96
Step 5	FLC	1 (7s)	1 (0.0611)	1 (11198)
	MPC	2.87	1.17	2.32
	PIC	5.71	2	1.96
Sawtooth	FLC	1 (1s)	1 (0.0568)	1 (7430)
	MPC	3	1.29	1.40
	PIC	24	1.75	3.46
TASKx6	FLC	1 (7s)	1 (0.0286)	1 (4058)
	MPC	1.40	1.21	2
	PIC	3.57	1.47	1.84
Step5-Random	FLC	1 (7s)	1 (0.0733)	1 (23531)
	MPC	2.71	1.41	2.45
	PIC	6.14	1.54	2.08

Table 3: Performance Summary

observe that the FLC decreases the settling time by up to 75% and 96%, E_{agg} by up to 56% and 74%, and number of total deadline misses by up to 62% and 71% compared to the MPC and PIC.

Controller	**CPU Utilization**	**Code Size**
PIC	0.25%	3 lines
FLC	0.53%	100 lines
MPC	0.95%	600 lines

Table 4: Control Overhead Comparisons

Finally, Table 4 shows the overhead of the tested controllers. All the controllers are lightweight and consume less than 1% CPU utilization for the sampling period of 1s. The PIC has the lowest overhead while the MPC has the highest overhead due to the complexity. The FLC consumes approximately 0.5% CPU utilization and a small amount of memory.

6. Related Work

Feedback control has been applied to manage the real-time system performance in dynamic environments. A number of existing approaches for feedback control of real-time performance such as [2, 8] mathematically model real-time system behaviors via difference equations. To apply classical linear control theory, real-time system behaviors are approximated in a piecewise linear manner. As control gains are determined offline, however, these approaches may fail when workloads or system behavioral characteristics deviate from the ones used for offline modeling. Many existing linear control theoretic approaches share this problem [9, 23].

Our QoS adaptation scheme via task period adaptation is similar to [18]. Abeni et al. [24] takes an alternative approach where the task budget rather than the task period is adapted under overload. Our approach can be integrated with the adaptive reservation scheme [24] too. This is reserved for future work.

Constrained predictive control techniques are applied to control the CPU utilization in a multiprocessor environment [3, 11]. Self-tuning regulators based on adaptive control theory [12] estimate the system model for automatic tuning of the controllers to manage the performance of e-commerce servers [25]. It is shown that a self-tuning regulator can converge to the target performance, if a set of conditions are met [25]. Adaptive control is also applied to differentiated web caching services [26]. However, model predictive control and adaptive control approaches are subject to online modeling errors. Therefore, they can only handle moderate nonlinearity. In contrast, fuzzy logic control is very effective to manage the performance of nonlinear, complex systems due to the model-free nature [1].

Fuzzy control theory has been applied to maximize the profit in an e-mail server [27]. eQoS [22] applies fuzzy control theory to differentiate services in a web server. However, they do not consider real-time constraints.

Little prior work has been done to apply fuzzy control theory to real-time performance management. Li et al. [15] apply fuzzy control to visual tracking; however, they do not consider the utilization control problem. Further, they do not analyze the stability of the fuzzy closed-loop system. Suzer et al. [14] have developed a fuzzy utilization controller. However, this paper presents a more advanced fuzzy rule-base to reduce potential overshoots and undershoots. Further, [14] does not provide stability analysis and evaluates performance via simulation.

7. Conclusion

In a number of real-time applications such as target tracking and traffic control, it is challenging to support the desired real-time performance. To closely support the specified utilization set-point in the presence of dynamic workloads and system behaviors, we design a fuzzy closed-loop system, while mathematically proving the stability of the fuzzy closed-loop system. Also, extensive experiments are performed to thoroughly evaluate the fuzzy, PI [2], and model predictive [3] controllers in a real-time kernel. Among the tested approaches, our fuzzy logic controller shows the smallest overshoots, undershoots, and reference tracking error as well as the shortest settling time to the set-point across all the tested workloads. To the best of our knowledge, no prior work has designed a fuzzy control system for real-time performance management with formal stability analysis, while comparing it to the PI and

model predictive controllers. In the future, we will develop more advanced fuzzy control techniques for real-time performance management.

References

[1] K. M. Passino and S. Yurkovich, *Fuzzy Control*. Addison-Wesley, 1998.

[2] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son, "Feedback Control Real-Time Scheduling: Framework, Modeling and Algorithms," *Real-Time Systems, Special Issue on Control-Theoretical Approaches to Real-Time Computing*, vol. 23, May 2002.

[3] C. Lu, X. Wang, and X. Koutsoukos, "Feedback Utilization Control in Distributed Real-Time Systems with End-to-End Tasks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, pp. 550–561, June 2005.

[4] J. Loyall, R. Schantz, D. Corman, and J. P. andS. Fernandez, "A Distributed Real-Time Embedded Application for Surveillance, Detection, and Tracking of Time Critical Targets," in *The 11th IEEE Real-Time Embedded Technology and Applications Symposium*, 2005.

[5] R. Carlson, "Sandia SCADA Program High-Security SCADA LDRD Final Report," tech. rep., SANDIA, 2002. SANDIA Report SAND2002-0729.

[6] J. W. S. Liu, *Real-Time Systems*. Prentice Hall, 2000.

[7] C. L. Phillips and H. T. Nagle, *Digital Control System Analysis and Design (3rd edition)*. Prentice Hall, 1995.

[8] M. Amirijoo, J. Hansson, S. H. Son, and S. Gunnarsson, "Experimental Evaluation of Linear Time-Invariant Models for Feedback Performance Control in Real-Time System," in *Real-Time Systems*, vol. 35, 2007.

[9] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. A John Wiley and Sons, Inc., Publication, 2004.

[10] J. Maciejowski, *Predictive Control with Constraints*. Prentice Hall, 2002.

[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, pp. 996–1009, July 2007.

[12] K. J. Åström and B. Wittenmark, *Adaptive Control*. Addison-Wesley, 2nd ed., 1994.

[13] "RTAI - the RealTime Application Interface for Linux from DIAPM." http://www.rtai.org.

[14] M. H. Suzer and K. D. Kang, "Adaptive Fuzzy Control for Utilization Management," in *IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing*, 2008.

[15] B. Li and K. Nahrstedt, "A Control-Based Middleware Framework for Quality of Service Adaptations," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 9, 1999.

[16] X. Wang, C. Lu, and X. Koutsoukos, "Enhancing the Robustness of Distributed Real-Time Middleware via End-to-End Utilization Control," in *IEEE Real-Time Systems Symposium*, 2005.

[17] X. Wang, C. Lu, and C. Gill, "FCS/nORB: A Feedback Control Real-Time Scheduling Service for Embedded ORB Middleware," *Microprocessors and Microsystems*, vol. 32, pp. 413–424, Nov. 2008.

[18] G. Buttazzo, G. Lipari, and L. Abeni, "Elastic Task Model For Adaptive Rate Control," in *IEEE Real-Time Systems Symposium*, 1998.

[19] R. K. Mudi and N. R. Pal, "A Self-Tuning Fuzzy PI Controller," *Fuzzy Sets and Systems*, vol. 115, pp. 327–338, October 2000.

[20] L. A. Zadeh, "Fuzzy Sets," *Information and Control*, vol. 8, no. 3, pp. 338–353, 1965.

[21] G. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice Hall, 1995.

[22] J. Wei and C.-Z. Xu, "eQoS: Provisioning of Client-Perceived End-to-End QoS Guarantees in Web Servers," *IEEE Transactions on Computers*, vol. 55, pp. 1543–1556, December 2006.

[23] T. F. Abdelzaher, J. A. Stankovic, C. Lu, R. Zhang, and Y. Lu, "Feedback Performance Control in Software Services," *IEEE Control Systems Magazine*, vol. 23, no. 3, 2003.

[24] L. Abeni, T. Cucinotta, G. Lipari, L. Marzario, and L. Palopoli, "QoS Management Through Adaptive Reservations," *Real-Time Systems*, vol. 29, no. 2-3, 2005.

[25] C. Karamanolis, M. Karlsson, and X. Zhu, "Designing Controllable Computer Systems," in *USENIX Workshop on Hot Topics in Operating Systems*, 2005.

[26] Y. Lu, A. Saxena, and T. F. Abdelzaher, "Differentiated Caching Services; A Control-Theoretical Approach," in *the 21st International Conference on Distributed Computing Systems*, 2002.

[27] Y. Diao, J. L. Hellerstein, and S. Parekh, "Using Fuzzy Control to Maximize Profits in Service Level Management," *IBM Systems Journal*, vol. 41, no. 3, 2002.