Bandwidth Adaptation in Hierarchical Scheduling Using Fuzzy Controllers

Nima Moghaddami Khalilzad, Moris Behnam, Giacomo Spampinato and Thomas Nolte
MRTC/Mälardalen University
P.O. Box 883, SE-721 23 Västerås, Sweden
nima.m.khalilzad@mdh.se

***Abstract*—In our previous work, we have introduced an adaptive hierarchical scheduling framework as a solution for composing dynamic real-time systems, i.e., systems where the CPU demand of their tasks are subjected to unknown and potentially drastic changes during run-time. The framework uses the PI controller which periodically adapts the system to the current load situation. The conventional PI controller despite simplicity and low CPU overhead, provides acceptable performance. However, increasing the pressure on the controller, e.g, with an application consisting of multiple tasks with drastically oscillating execution times, degrades the performance of the PI controller.**

Therefore, in this paper we modify the structure of our adaptive framework by replacing the PI controller with a fuzzy controller to achieve better performance. Furthermore, we conduct a simulation-based case study in which we compose dynamic tasks such as video decoder tasks with a set of static tasks into a single system, and we show that the new fuzzy controller outperforms our previous PI controller.

I. Introduction

The Hierarchical Scheduling Framework (HSF) is a component based technique for scheduling complex real-time systems [1], [2]. Using such a framework, each component is allocated a portion of the CPU and, in turn, it guarantees that with this portion all its internal tasks will be scheduled such that their corresponding timing constraints are respected. The CPU portions are often specified by the component period and budget (interface parameters). The interface parameters can be calculated either based on the Worst Case Execution Time (WCET) of the tasks such as the method presented in [3] and kept fixed during run-time, or be initiated using such a method and then be adapted during run-time based on the current workload [4]. The dynamic resource allocation techniques are especially efficient when the system components are composed of dynamic tasks in which their execution times are changing significantly during run-time. For example, when a component consists of control tasks or video decoder tasks, since the execution time of such tasks are dynamic during run-time, fixed recourse allocation techniques are not efficient and may result in underutilized systems and consequently the CPU resource will be wasted.

We have introduced the Adaptive Hierarchical Scheduling Framework (AHSF) [4] as a solution for composing dynamic components. In this hierarchical framework we assume a fixed period for each subsystem, however, each subsystem is equipped with a budget controller which adapts the subsystem budget based on two feedback loops. The feedback loops are controlling the number of deadline misses and the amount of idle time in the subsystem. In that work, we used the well known PI controller, designed based on an approximate system model.

In this paper we investigate a more advanced controller which does not require any pre knowledge about the system. We introduce the following contributions in this paper.

- i) We investigate using a model-free fuzzy controller instead of conventional PI controllers and define a new control variable based on the consumed budget after missing the deadlines instead of the number of deadline misses.
- ii) We study the stability of our controller using Lyapunov's direct method which gives boundaries on the budget controller's gain values.
- iii) We tune the designed fuzzy controller using a multi-criteria Genetic Algorithm (GA).
- iv) We evaluate the performance of the proposed controller by conducting a case study and we compare the result of using the proposed controller against the PI controller approach presented in [4].

The remainder of this paper is organized as follows. Related work is presented in Section II. Section III describes the structure of our AHSF. In Section IV we give insight into our fuzzy budget controller. The stability study is presented in Section V. We describe the controller tuning in Section VI. A simulation-based case study is presented in Section VII. The implementation complexity of the proposed approach is discussed in VIII. Finally, we conclude the paper in Section IX.

II. Related Work

The idea of closed-loop real-time scheduling emerged in late 90's [5] and since then there has been a growing attention in combining the real-time scheduling theory with the well-established control techniques. The deadline miss ratio is controlled in [6]. In [7], in addition to the deadline miss ratio, the CPU utilization is controlled as well. In a similar context, the CPU utilization is controlled by modifying task periods using a fuzzy controller in [8]. The idea is also applied to scheduling of control tasks where the quality-of-control is regulated using a feedback-feedforward method [9].

Resource reservation [10] and hierarchical scheduling [11], [12], [3], [13], [14], [15], [16] techniques have received increasingly more attention over the past two decades since they provide temporal isolation and consequently predictability in integrating different task models. Hierarchical scheduling is used in scheduling of soft real-time systems in [17], [18], [19]. All aforementioned methods assign fixed CPU portions to the subsystems and therefore it makes them inefficient when composing dynamic tasks.

Recently, there has been some work to enable the adaptability feature for resource reservation scheduling techniques by using feedback control techniques. In [20] Abeni et al. have introduced an adaptive Constant Bandwidth Server (CBS) as an extension to the CBS [21] in which the server budgets are adjusted during run-time. Their control variable is limited to existence of one task per server. Adaptive CPU resource management is presented in [22] where the hard CBS scheduling algorithm is used and the server budgets are adapted during run-time. Although in theory this approach can support existence of multiple tasks in a server, it is evaluated by using only one task per server. In our AHSF [4], we bring the feedback scheduling techniques in the context of resource reservation scheduling in which the servers (components) consist of multiple tasks and they are scheduled using a real-time scheduler (hierarchical scheduling). Besides, we use periodic servers [23] instead of CBSs in our framework, however, our work can be extended to work under other type of servers such as CBSs with minor modifications.

III. The Adaptive Hierarchical Scheduling Framework

We consider a two level Adaptive Hierarchical Scheduling Framework (AHSF) in which a system S consisting of N components, here denoted as subsystems $S_s \in S$, is executed on a single processor. In the AHSF, a global scheduler schedules subsystems, and a local scheduler in each subsystem is responsible for scheduling its corresponding internal tasks. We use the EDF scheduling algorithm in both global and local level in this paper, however the presented approach can be extended easily to include other scheduling algorithm, i.e., fixed priority scheduling. Figure 1 shows the architecture of our two-level AHSF. We use one fuzzy budget controller per subsystem to adapt its budget according to the CPU resource demand of its

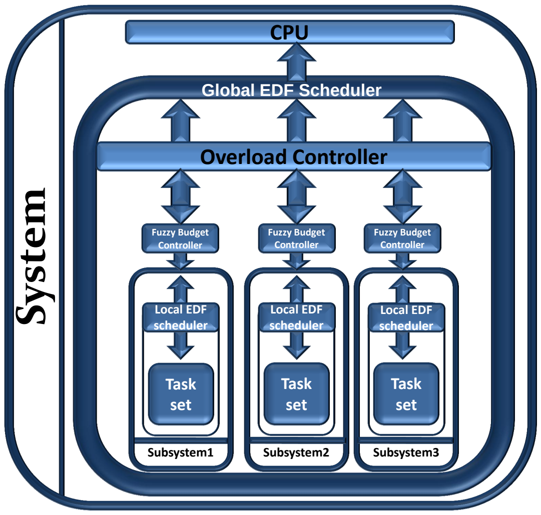

Fig. 1. The Adaptive Hierarchical Scheduling Framework.

tasks. In addition, there is an overload controller which deals with overload situations i.e., when the total resource request of the subsystems are more than the available CPU resources.

A. Subsystem Model

Each subsystem S_s is represented by its temporal interface parameters (T_s, B_s, D_s, ζ_s) where T_s, B_s, D_s and ζ_s are subsystem period, budget, relative deadline and criticality respectively. The relative deadline of a subsystem is assumed to be equal to its corresponding subsystem period ($D_s = T_s$). Each subsystem S_s consists of a set of n_s tasks τ_s and a local scheduler. The criticality of a subsystem ζ_s, which shows how critical a subsystem is in comparison to other subsystems, is used only in overload situations. We assume that subsystems are sorted according to their criticality, in the order of decreasing criticality, and $\zeta_s = s$, i.e., S_1 has the highest criticality in the system while S_N has the lowest criticality.

We use periodic servers (subsystems) which works as follows. The required CPU portion is always allocated to the subsystems every predefined period, and in the case that there is no active task in the subsystem, it will idle its budget.

B. Task Model

We assume the periodic soft real-time task model $\tau_{i,s}(T_{i,s}, C_{i,s}, D_{i,s})$, where $T_{i,s}$, $C_{i,s}$ and $D_{i,s}$ are period, execution time and relative deadline of task i in subsystem S_s respectively. The relative deadline of a task is assumed to be equal to its corresponding task period ($D_{i,s} = T_{i,s}$). When a task misses its deadline it can continue its execution to the end.

C. The Budget Controller

We use two feedback loops in the structure of our budget controller. For the first loop controller computations, the amount of subsystem budget that is used by tasks after missing their deadlines to finish their executions, is monitored. Therefore, the error in this loop is defined as follows:

$$e_m(t) = \sum_{\tau_{i,s} \in \tau_s} \frac{\beta_{i,s}(t)}{T_s} \tag{1}$$

where $\beta_{i,s}(t)$ is the controlled variable of the feedback loop which is the amount of subsystem budget of S_s used by task i after missing its deadline at sampling time t. We call this feedback loop the "m-loop" in the rest of the paper. Note that in [4] we measure the number of deadline misses in the "m-loop", however, we believe that our new control variable (Equation 1) provides the controller with more precise information about the state of the system, consequently the controller can take more effective actions in controlling the environment. It goes without saying that Equation 1 can only be used if tasks are allowed to continue executing after missing their deadlines.

In the second loop, the amount of idle time (unused budget) in each subsystem is monitored. Therefore, the error in this loop is defined as follows:

$$e_u(t) = \frac{\alpha_s(t)}{T_s} \tag{2}$$

where $\alpha_s(t)$ is the controlled variable of the second feedback loop which is the amount of idle time in subsystem S_s measured at sampling time t. Similar to the m-loop, we call the second feedback loop the "u-loop" in the rest of the paper. Figure 2 illustrates the defined controlled variables in subsystem S_1 of an example system. There are two tasks in S_1 where $\tau_{2,1}$ misses its deadline at t_0, however, $\tau_{1,1}$ finishes its execution before its deadline. If we consider t_0, t_1 and t_2 as the sampling times, the value of the controlled variables in Figure 2 at the sampling times are as follows:

$$\alpha_1(t_0) = \alpha_1(t_1) = \alpha_1(t_2) = q_1,$$
$$\beta_{1,1}(t_0) = \beta_{1,1}(t_1) = \beta_{1,1}(t_2) = 0,$$
$$\beta_{2,1}(t_0) = 0, \beta_{2,1}(t_1) = x_1, \beta_{2,1}(t_2) = x_1 + x_2.$$

The controllers of both loops are executed periodically and both error values are reset to zero at each control period. The controller periods are assumed to be proportional to the subsystem periods. In addition to the error value, the controller should be provided with the error difference value which is calculated as follows:

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

where $e(t)$ is the error value at sampling time t. The error is either e_m or e_u depending on the control loop, therefore,

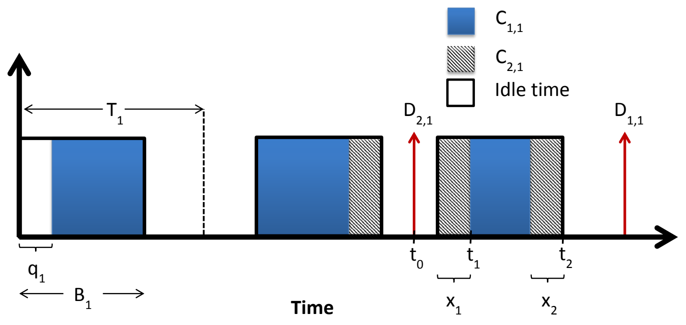

Fig. 2. The controlled variables in S_1.

there is an error difference variable per control loop: $\Delta e_m(t)$ corresponding to the m-loop and $\Delta e_u(t)$ corresponding to the u-loop. We provide the fuzzy controller with $e(t)$ and $\Delta e(t)$, and the controller computes a CPU portion $\Delta w(t)$ which affects the subsystem budget, i.e., it might increase the subsystem budget if there are deadline misses or decrease the subsystem budget if there is much unused budget. Hence, the new subsystem budget is calculated using the controller output and subsystem period:

$$B(t) = B(t-1) + T_s \Delta w(t). \tag{4}$$

The fuzzy budget controller is indeed an integral controller which adds the controller output to the current budget. The controller output is calculated as follows:

$$\Delta w(t) = K_f e(t) \tag{5}$$

where K_f is the proportional gain which is extracted from the fuzzy rule-base given $e(t)$ and $\Delta e(t)$. The rule-base and the fuzzy logic control is explained in detail in Section IV. The block diagram of the fuzzy budget controller is illustrated in Figure 3.

D. Integration of Feedback Loops

As mentioned earlier in this section, we use two feedback loops in our framework. Each loop calculates a budget change value $\Delta w(t)$, however, a mechanism should be provided for integrating these two values. We design a fuzzy multiplexer which combines the output of the control loops. The multiplexer simply looks at the systems state, if the m-loop error is large, the output of the m-loop $\Delta w_m(t)$ will have the main impact on the final output, otherwise the final output is mainly based on the output of the u-loop $\Delta w_u(t)$. The block diagram of the fuzzy multiplexer is shown in Figure 3. There are two fuzzy sets in the structure of the multiplexer: large and zero. Basically, the fuzzification and defuzzification steps in the integration phase are very similar to the steps presented in Section IV for the budget controller, hence to avoid redundancy we do not explain them in this section. However, it is important to highlight that we use different fuzzy sets and a

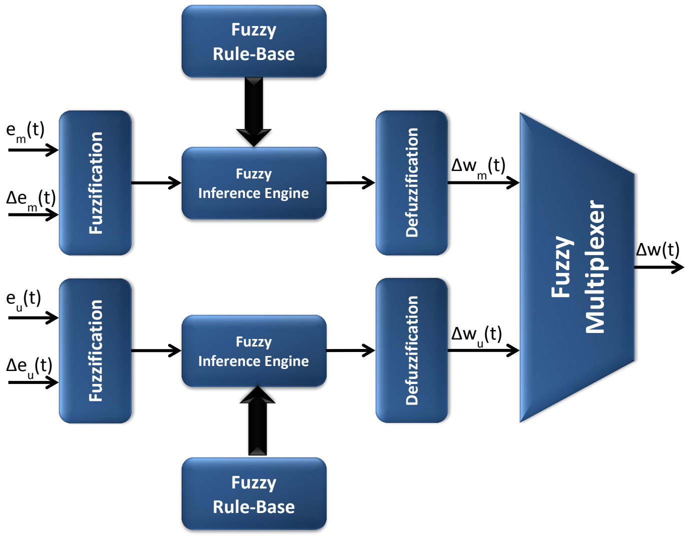

Fig. 3. Block diagram of the fuzzy budget controller. $\Delta w_m(t)$: m-loop output, $\Delta w_u(t)$: u-loop output.

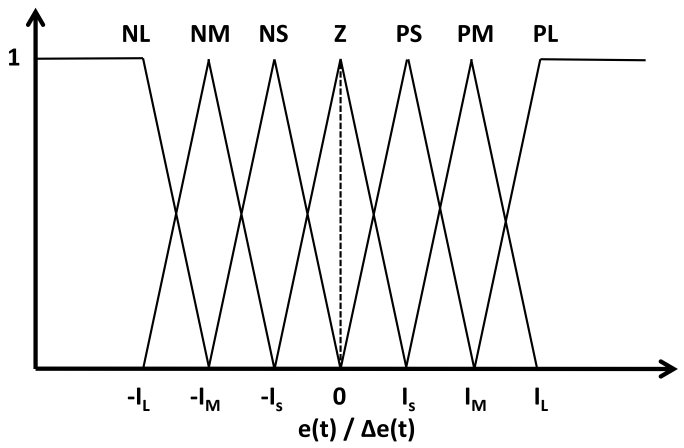

Fig. 4. Membership function (NL:Negative Large, NM:Negative Medium, NS:Negative Small, Z:Zero, PS:Positive Small, PM: Positive Medium, PL:Positive Large, I_S: Ceiling of Small, I_M: Ceiling of Medium, I_L: Ceiling of Large).

	$e_m(t)$	
$e_u(t)$	Zero	Large
Zero	$\Delta w_u(t)$	$\Delta w_m(t)$
Large	$\Delta w_u(t)$	$\Delta w_m(t)$

TABLE I
FUZZY MULTIPLEXER RULE-BASE.

different rule-base than the budget controller. The rule-base presented in Table I is used in the fuzzy multiplexer.

E. The Overload Controller

The overload situation can happen when the total system utilization is more than 100% and since the EDF scheduling algorithm is assumed, it is detected by performing the following test:

$$\sum_{\forall S_s \in S} \frac{B_s}{T_s} > 1. \tag{6}$$

If the controller detects the overload situation, it redistributes the CPU resource among subsystems according to their criticality values ζ_s. It starts from the highest criticality subsystem S_1 and provides it with required budget. Thereafter, it moves to a lower criticality subsystem. The lower criticality subsystem can at most receive a budget value which corresponds to the CPU resource that is left after allocation to the highest criticality subsystems. This process continues until the lowest criticality subsystem receives CPU resources, which happens after all other subsystems have been assigned a new budget. In other words, when the controller finds out that there are not enough resources for all the subsystems, it tries to satisfy the high criticality subsystems by sacrificing the lower criticality subsystems. Note that in this approach, the low criticality subsystems might receive very small CPU portions or be completely shut down which is unavoidable due to the limited CPU resources.

Without having an overload controller, in the overload mode high priority subsystems will receive more resources than the low priority ones. Since we use the EDF scheduler at the global level, the shorter period subsystem are more likely to have higher priorities. However, the shorter period subsystems are often not the more important ones in the system since the periods are usually describing the temporal requirement and not the importance of the subsystems.

IV. Fuzzy Logic Control

In this section, we explain how the control input $e(t)$ and $\Delta e(t)$ are mapped to the control output $\Delta w(t)$ using Fuzzy Logic Control (FLC) [24]. As illustrated in Figure 3, the first step in the FLC is fuzzification in which we map the crisp input error to a linguistic value. We use the membership function presented in Figure 4 for the fuzzification purpose. Given an input value and using the membership function we get a set of truth values indicating how much the input belongs to each fuzzy set. According to the definition of the controlled variables the error value is always positive, therefore only the positive side of the membership function is needed for $e(t)$, however, $\Delta e(t)$ can be either negative or positive. We use the same membership function for both $e(t)$ and $\Delta e(t)$. In the next step we apply fuzzy rules to our fuzzy inputs to get a fuzzy control output. We use different rule-bases for each feedback loop which are presented in Table II.

We use the minimum operator as "fuzzy and" to calculate the truth value of each fuzzy rule for the inference purpose. The final step in FLC is defuzzification in which a linguistic

	e(t)			
$\Delta e(t)$	Z	PS	PM	PL
NL	NM / PL	NM / PM	NS / PS	Z / Z
NM	NS/PM	NS / PS	Z / Z	PS / NS
NS	NS / PS	Z / Z	PS / NS	PM / NM
Z	NZ / PM	PS / Z	PM / NM	PL / NL
PS	PS / PM	PM / Z	PL / NL	PL / NL
PM	PM / PS	PL / Z	PL / NL	PL / NL
PL	PL / Z	PL / Z	PL / NL	PL / NL

TABLE II
FUZZY CONTROLLER RULE-BASE (M-LOOP / U-LOOP).

control action is mapped to a crisp value. Let $o(k)$ and $\mu(k)$ represent the rule consequent and the truth value of k'th fuzzy rule respectively. Finally, according to the Sugeno's defuzzification model [25] the proportional gain is the weighted average of all rule outputs:

$$K_f = \frac{\sum o(k).\mu(k)}{\sum \mu(k)}. \tag{7}$$

The rule consequent can be either zero gain K_z, small gain K_s, medium gain K_m or large gain K_l. We assume the zero gain is always 0. For example, assume $e(t) = \Delta e(t) = \frac{I_s}{2}$ (I_s is the ceiling of small fuzzy set). Then, for both $e(t)$ and $\Delta e(t)$, $\mu_Z = \mu_{PS} = 0.5$ where μ_Z and μ_{PS} are the truth value of the "Zero" and "Positive Small" fuzzy sets. Therefore the output of m-loop is:

$$K_f = \frac{0 \times 0.5 + K_s \times 0.5 + K_s \times 0.5 + K_m \times 0.5}{0.5 + 0.5 + 0.5 + 0.5}, \tag{8}$$

and the output of u-loop is:

$$K_f = \frac{K_m \times 0.5 + 0 \times 0.5 + K_m \times 0.5 + 0 \times 0.5}{0.5 + 0.5 + 0.5 + 0.5}. \tag{9}$$

V. STABILITY STUDY

In controlled systems, stability is one of the important properties which should be studied after designing the controller. We use the direct method of the Lyapunov's stability analysis [26] to study the stability of our system. Assume that $y(t)$ is a function representing the distance of the current budget $B(t)$ from the equilibrium budget B_{eq} at sampling time t. The equilibrium budget is the budget that the subsystem neither experiences idle time nor its tasks miss their deadlines:

$$y(t) = B(t) - B_{eq}. \tag{10}$$

We define the Lyapunov function as follows:

$$V(y(t)) = y(t)^2. \tag{11}$$

Now, we should prove that the following conditions are satisfied.

1) $V(y(t)) = 0$, if the system is in its equilibrium state.
2) $V(y(t)) > 0$, if the system is in other states than its equilibrium.
3) $V(y(t+1)) - V(y(t)) = y^2(t+1) - y^2(t) < 0$

From the definition of our Lyapunov function, condition 1 and 2 are obviously satisfied. Now we need to study the third condition. From the definition of $y(t)$ we have:

$$y^2(t+1) - y^2(t) = (B(t+1) - B_{eq})^2 - (B(t) - B_{eq})^2. \tag{12}$$

Expanding 12 and replacing $B(t+1)$ using Equation 4 and 5 we get:

$$y^2(t+1) - y^2(t) = K_f^2 T_s^2 e(t)^2 + 2K_f T_s e(t)(B(t) - B_{eq}). \tag{13}$$

Using Equation 11 we get:

$$y^2(t+1) - y^2(t) = K_f^2 T_s^2 e(t)^2 + 2K_f T_s e(t) y(t). \tag{14}$$

Therefore, according to the third condition the following inequality should be valid:

$$K_f^2 T_s^2 e(t)^2 + 2K_f T_s e(t) y(t) < 0 \tag{15}$$

which yields to the following two results:

1) $sign(K_f) = -sign(y(t))$ because all other variables are positive.
2) $|K_f e(t)| < |2\frac{y(t)}{T_s}|$.

Result 1 is used in design of the rule-base meaning that when the distance from the equlibrium budget is positive and consequently we have some idle time in the subsytem, the gain value K_f is negative, and if y is negative and there are some deadline misses in the subsytem K_f is positive.

Figure 5 shows a simple scenario that two tasks exist in a subsystem. In this case both $\tau_{1,1}$ and $\tau_{2,1}$ are missing their deadlines, and at any sampling time t: $y(t) = x_1 + x_2 + x_3$ meaning that if we add $y(t)$ to the current budget it is guaranteed that the tasks can finish their execution times before their deadlines. However, assuming that the controller period is proportional to the subsystem period (see Section VII), the controller can sample the subsystem at either t_0, t_1 or t_2 depending on the controller period. The control variables $\beta_{1,1}$ and $\beta_{2,1}$ at these sampling times are as follows.

- At t_0: $\beta_{1,1} = \beta_{2,1} = 0$ and $\sum_{\tau_{i,1} \in \tau_1} \beta_{i,1}(t_0) < y(t_0)$.
- At t_1: $\beta_{1,1} = x_1, \beta_{2,1} = 0$ and $\sum_{\tau_{i,1} \in \tau_1} \beta_{i,1}(t_1) < y(t_1)$.
- At t_2: $\beta_{1,1} = x_1, \beta_{2,1} = x_2 + x_3$ and $\sum_{\tau_{i,1} \in \tau_1} \beta_{i,1}(t_2) = y(t_2)$.

Hence, we conclude that:

$$\sum_{\tau_{i,s} \in \tau_s} \beta_{i,s}(t) \leq y(t) \tag{16}$$

at any sampling time t. A similar reasoning can be done for the u-loop and its controlled variable $\alpha_s(t)$. Therefore, according to Equation 16 and the error definitions we derive:

$$e(t) \leq \frac{y(t)}{T_s}. \tag{17}$$

From result 2 and Equation 17 we derive that in order for the system to be stable, $|K_f|$ should be less than 2. The upper

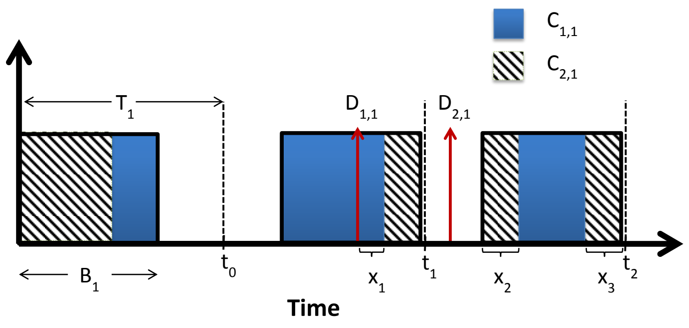

Fig. 5. The controlled variables in subsystem S_1 of a sample system.

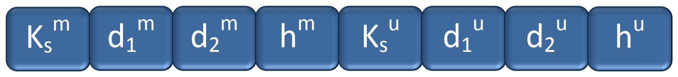

Fig. 6. Structure of the chromosomes used in the GA.

bound for $|K_f|$ can be considered higher than 2 depending on the controller frequency. The higher the controller frequency the greater the gain boundary, although from the stability point of view, it is always safe for the system to fulfill $|K_f| < 2$ condition.

The stability analysis gives some guidelines on how to design the rule-base and how to configure the controller. Although we do not get exact boundaries on the gain value K_f, the stability study provides us with approximate boundaries which we use in tuning of the controller. In Section VI we use evolutionary search to find an optimum configuration for our controller. Indeed, confining K_f, using the stability study in this section, limits the search space that we need to explore and speed-ups the convergence of the search.

VI. Tuning the Controller Using Evolutionary Search

There are many parameters in our designed fuzzy controller that need to be tuned. For instance the fuzzy set intervals (both in the budget controller and in the multiplexer) are crucial parameters that should be carefully chosen. In addition, the gain values that are used in the output of the fuzzy rule-base need tuning. We use the GA to find the optimum parameters that maximize the performance of the controller.

There is a set of parameters associated with each control loop. For each loop the interval of small, medium and large fuzzy sets in addition to the three gain values: small gain K_s, medium gain K_m and large gain K_l should be tuned. To define the set intervals we only need to consider three values (see Figure 4): the ceiling of "small" set I_s, the ceiling of "medium" set I_m and the ceiling of "large" set I_l. Note that we have different sets for each of the loops. We show u-loop parameters using the super script u, and if the parameter belongs to m-loop we show it using the super script m. For example K_s^u is the small gain value corresponding to u-loop while K_s^m is the small gain value in m-loop.

The first step in using the GA is to design the structure of the so called chromosomes. We assume that each chromosome contains the information of all the parameters. However, we store the information indirectly to bias the GA. We assume that the fuzzy sets are harmonic meaning that:

$$I_l - I_m = I_m - I_s = I_s - 0 = h$$

where h is the base interval size. This assumption limits the search space and helps the GA to converge faster, however, it might prevent it to find the absolute optimum solution. Recall from Section III-D that the fuzzy multiplexer has two fuzzy sets. The ceiling of "large" set is assumed to be $1.5 \times h$ while the ceiling of "zero" set is 0. In addition, assuming that $K_l > K_m > K_s$, we can write:

- $K_m = K_s + d_1$
- $K_l = K_m + d_2$

where d_1 and d_2 are the difference of the medium gain with the small gain and the difference of the large gain with the medium gain respectively. Figure 6 illustrates the structure of the chromosomes showing that they contain all the tunable parameters.

Designing the mutation and crossover operators are the next stage in using the GA. We use the one point crossover operator on the randomly selected parents from the breeding pool. The mutation point is selected randomly as well. Thereafter, we add a random value between -0.1 and 0.1 to the selected variable. However, there are some boundaries on the variables, for instance none of the fields can be less than zero, therefore, if the result of the mutation is out of the valid region we immediately conduct another mutation.

The fitness function should be designed such that it directs the generations towards more optimum generations. We have two criteria in evaluating each chromosome. The lower the number of deadline missed tasks, the higher the efficiency. In addition, the amount of the idle time in the subsystems should be as low as possible. Therefore, we use a multi-objective GA approach called Vector Evaluated GA (VEGA) [27]. In this approach there are multiple fitness variables associated with each chromosome based on different criteria. When we want to select a parent, first we randomly choose the effective criterion, meaning that the chromosomes that are fit with respect to the selected criterion have a higher chance in being selected as a parent. Thereafter, we select the first parent and repeat the same procedure to select the second parent. After conducting extensive simulations we came to the conclusion that the solution converges faster when using three objectives which are based on i) number of deadline misses ii) amount of idle time iii) combination of i and ii.

Algorithm 1 Tuning the control parameters using VEGA

```
for $i = 0$ to $i = populationSize$ do
    population(i) = random();
end for
for $j = 0$ to $i = maxGeneration$ do
    for $i = 0$ to $i = populationSize$ do
        simulation(population(i));
        fitnessIdle(i) =calculateFitness(idle);
        fitnessDl(i) =calculateFitness(dl);
        fitnessTotal(i) =calculateFitness(total);
    end for
    pool = population;
    for $k = 0$ to $k = populationSize/2$ do
        objective = randomInt(1,3);
        parent1 = selectParent(objective);
        objective = randomInt(1,3);
        parent2 = selectParent(objective);
        crossoverPoint = randomInt(1,8);
        children = crossover(parent1, parent2, crossoverPoint);
        population(2*i-1) = mutate(children(1));
        population(2*i) = mutate(children(2));
    end for
end for
```

Algorithm 1 shows the multi-objective GA used for tuning the parameters. It starts by generating random chromosomes and runs the simulation using their parameters. Thereafter it finds the fitness value of the chromosomes based on the three objectives. Afterward, the next generation is created based on the previous generation given that fitter chromosomes have more chance in being selected as a parent. The functions and variables involved in the algorithm are:

- *"populationSize"* is the size of population.
- *"population"* is an array of chromosomes.
- *"random()"* is a function that returns a random variable between zero and one.
- *"maxGeneration"* is the number of generations that the GA tries to perform the optimization.
- *"simulation(population(i))"* given task sets, execution time change patterns and the variables in the chromosome of population i performs the simulation.
- *"fitnessIdle(i)"*, *"fitnessDl(i)"* and "*fitnessTotal(i)*" store the fitness value of the population i based on the idle time, deadline miss and combination of both objectives.
- "*calculateFitness()*" function calculates the fitness value based on the input objective.
- "*idle*", "*dl*" and "*total*" are the thee objectives that we use in the GA.
- *"pool"* stores the current generation which is used to generate the next generation.
- "*randomInt(a, b)*" returns an integer random variable between a and b.
- *"objective"* stores the randomly selected objective.
- *"parent1"* and *"parent2"* are the selected parents for the crossover.
- *"selectParent(objective)"* randomly selects an individual from *"pool"* given the input objective meaning than the individuals that are fit with respect to the input objective have a higher chance to be selected.
- "*crossover()*" does the crossover operation on its input chromosomes and given the crossover point. It returns two children.
- *"crossoverPoint"* stores the gene number that the crossover should be performed on it.
- "*chidden*" is an array which stores the output of the crossover operators.
- "*mutate()*" chooses a random mutation point and performs the mutation.

VII. Evaluation

In this section we design a case study using real task execution times measured from running a video decoder task on a sequence of TV frames. We use the same data as the authors of [28], which they used for evaluating their work. We have used the TrueTime [29] simulation tool for our evaluation purposes. The TrueTime kernel has been modified such that our AHSF has been implemented.

In this study, we assume a system consisting of three subsystems where each of them is composed of three tasks. Subsystem S_1 is composed of a decoder task with two other tasks having fixed execution times during run-time. Subsystem S_2 contains two fixed execution time tasks and a dynamic task. The dynamic task operates in two modes: low and high, where its execution time is doubled when it is in the high mode. We assume that this task changes its mode each 2 seconds. The reason that we add this task to our sample system is to increase the pressure on the budget controllers, hence their difference with respect to the control performance can be revealed and compared easily. Subsystem S_3 has the same type of tasks as S_1. The only difference between them is that the decoder task in S_1 decodes the frames in a higher quality level than the decoder task in S_3. The execution times of the decoder tasks decoding various frames are shown in Figure 7. The figure illustrates that the execution time is fluctuating depending on the content of the frames. Table III shows the task specifications in detail. The executions times reported for the dynamic tasks are the mean execution times.

Three types of budget allocation techniques are studied for scheduling the described system.

- First of all we allocate a fixed budget for each subsystem using the analytical approach presented in [3]. The exe-

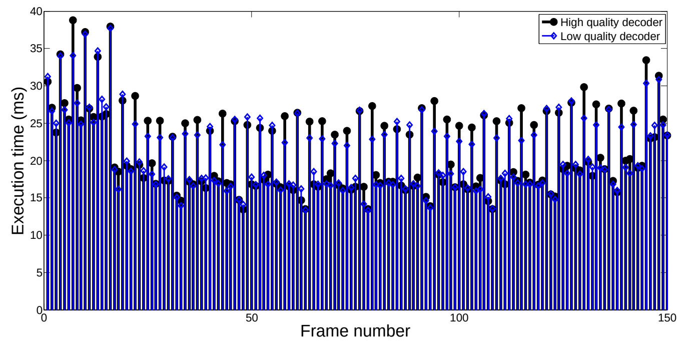

Fig. 7. Execution times of the decoder tasks in two quality levels decoding the first 150 frames of the TV stream.

$\tau_{i,s}/S_s$	$T_{i,s}/T_s$	$C_{i,s}/C_s$
$\tau_{1,1}$	40	2.4
$\tau_{2,1}$	30	5
$\tau_{3,1}$	30	4
S_1	10	2.5
$\tau_{1,2}$	60	8
$\tau_{2,2}$	50	5
$\tau_{3,2}$	90	4
S_2	15	5
$\tau_{1,3}$	40	2.3
$\tau_{2,3}$	70	7
$\tau_{3,3}$	80	6
S_3	20	8.5

TABLE III
SPECIFICATIONS OF TASKS AND SUBSYSTEMS USED IN THE CASE STUDY.

cution time of the dynamic tasks are assumed to be equal to the mean value of their execution times in the budget calculation analysis.

- Secondly, we use the PI controller [4] for dynamically allocating the budgets.
- Finally, we use the fuzzy controller introduced in this paper which allocates the budgets during run-time.

The control frequency is an important parameter which should be taken into account when designing an adaptive scheduler. Although frequently sampling and manipulating the environment might give a good control performance, due to the control overhead on the CPU, it is desirable to invoke the controller in a lower frequency. A reasonable approach for setting the control period is to set it proportional to its subsystem period. Therefore, in the PI controller, the controller period of each subsystem is set to be equal to the corresponding subsystem period times two. However, since the fuzzy controller has more overhead than the PI controller, we assign longer control periods which are equal to the subsystem periods times six.

The fuzzy controller is tuned using the first 10 seconds of the simulation with the help of the GA presented in Section VI. We started with 150 individuals and stopped the GA after 50 generations. Then we picked the best individual from

Performance metric	Technique	S_1	S_2	S_3
# DL misses	Fixed	7566	3027	2
	PI	44	1004	77
	Fuzzy	76	335	64
DL miss ratio	Fixed	29.68%	24.08%	0.03%
	PI	0.24%	9.51%	0.97%
	Fuzzy	0.41%	3.39%	0.81%
idle time	Fixed	26	2661	44175
	PI	2149	6003	2393
	Fuzzy	2352	4304	3037

TABLE IV
COMPARISON OF THE PERFORMANCE OF THE THREE BUDGET ALLOCATION APPROACHES. DL: DEADLINE. IDLE TIME IS IN MILLISECOND.

the 50'th generation which was fit with respect to the third objective (see Section VI). Afterwards, we ran the simulations for 200 seconds and compared the performance of the budget allocation techniques with each other.

Figure 8 illustrates the budget value of the three subsystems during run-time, which are allocated using the three techniques. The dynamic tasks are also shown in the figure. The y-axes of the figure shows the value of subsystem budgets and the value of execution times of the dynamic tasks. Since the execution times of the dynamic tasks are changing, the subsystem budgets (in the case that they are adaptive) are changing as well. Note that S_1 has the highest criticality in the system and $\zeta_1 > \zeta_2 > \zeta_3$. The figure clearly shows that assigning fixed budgets using the analysis is very pessimistic and it results in resources being wasted, although we used the mean value of the dynamic tasks in the analysis. Therefore, if we use the maximum execution times in the analysis, it will give even more pessimistic budgets.

Table IV summarizes the performance metrics that we are interested in after running the simulation using the three techniques. The table shows that a fixed budget allocation is not efficient at all since the system is overloaded, and between the PI controller and the fuzzy controller, despite the fact that the fuzzy controller has a longer control period, the fuzzy controller is more successful in reducing task deadline misses. The main difference between the performance of the two controllers is in deadline miss ratio of S_2, where the fuzzy controller managed to reduce the deadline miss ratio with an additional approximately 6% compared to the PI controller.

In the second simulation we modify task $\tau_{1,2}$ in the previous sample system such that the execution time of the dynamic task is tripled in the high mode which imposes even more pressure on the budget controllers. However, this time we do not tune our fuzzy controller and we use the previously tuned controller to see how well it works in a scenario similar to the one that it is tuned to work in. The deadline miss ratio for the three subsystems is presented in Table V. The table

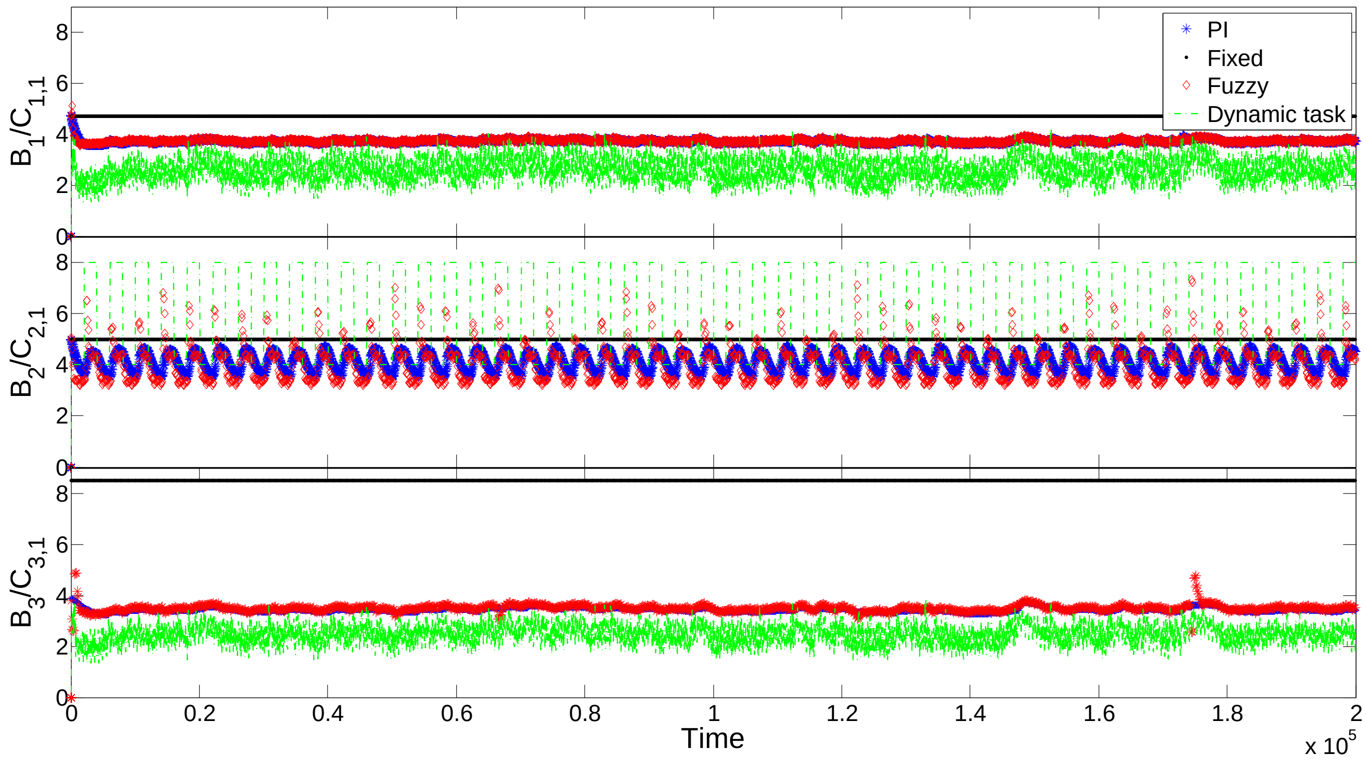

Fig. 8. Budget adaptation in the case study.

Controller	S_1	S_2	S_3
PI	0.24%	17.70%	1.33%
Fuzzy	0.41%	3.10%	7.58%

TABLE V
DEADLINE MISS RATIO IN THE SECOND SIMULATION.

shows that the fuzzy controller is able to reduce the deadline miss ratio with an additional approximately 14% compared to the performance of the PI controller in S_2, however the PI controller is 6% better in S_3. Since the system is overloaded one subsystem should be sacrificed, and since the PI controller is slower in adaptation, it sacrifices the higher criticality subsystem S_2 and serves the lower criticality one S_3. Therefore in total the fuzzy controller successfully schedules 8% more tasks, and taking the criticalities into account, the value of avoiding deadline misses in S_2 is higher than S_3.

A potential drawback with the fuzzy budget controller could be its tuning. Since the tuning process is application specific, a tuned controller for a specific application could be less efficient for other applications. However, the simulation results suggest that the fuzzy controller works fine in relatively similar dynamic scenarios to the scenario that it is tuned for. In general, the closer the tuning scenario is to the test scenario, the better the performance.

VIII. Implementation Complexity

In this section we explain the implementation complexity of the different stages in the fuzzy budget controller.

The input values $e(t)$ and $\Delta e(t)$ can at most belong to two neighbor fuzzy sets. The corresponding fuzzy sets can be found by a couple of "if" statements. Thereafter, their membership value should be calculated. Given the set boundaries calculating μ requires a sum operation together with a multiplication. Afterwards, the "fuzzy and" operator should be performed on the two membership value. The "fuzzy and" operator consist of an "if" statement. At most four "fuzzy and" operations should be performed. Finally Equation 7 should be executed for the defuzzification purpose. These stages are done for both control loops.

The fuzzy multiplexer has the same stages as the budget controller, however, since the number of fuzzy sets are fewer, finding the corresponding fuzzy set for the input value requires less computation.

The overload controller, which only gets activated in the overload situation, consist of sum and assignment operations. The controller loops through the subsystems, assigns a new budget to them if necessary and updates the available CPU resource. The number of iterations in the loop is equal to the number of subsystems in the system.

As a conclusion, given that the fuzzy controller requires running in lower frequency than the PI controller, it does not add a significant overhead when it is implement.

IX. Conclusion

In this paper, we studied the use of a more advanced controller (fuzzy controller) than the conventional PI controller in our adaptive hierarchical scheduling framework for controlling the subsystem budgets during run-time. Thereafter, we showed how the fuzzy budget controller is tuned using a multi-objective genetic algorithm. To study the performance of the new budget controller we conducted a case study using video decoder tasks where the fuzzy controller outperformed the PI controller.

We intend to extend our work to the context of multi-core systems where an adaptive hierarchical framework runs on a multi-core CPU. Furthermore, since the control overhead is one of the main issues in our adaptive framework, we want to deeply study this issue by implementing the controllers in the Linux kernel.

References

[1] Z. Deng and J. W.-S. Liu, "Scheduling real-time applications in an open environment," in *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS '97)*, December 1997, pp. 308 – 319.

[2] G. Lipari and S. Baruah, "A hierarchical extension to the constant bandwidth server framework," in *Proceedings of the 7th IEEE Real-Time Technology and Applications Symposium (RTAS '01)*, May 2001, pp. 26 – 35.

[3] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS '03)*, December 2003, pp. 2 – 13.

[4] N. M. Khalilzad, T. Nolte, M. Behnam, and M. Asberg, "Towards adaptive hierarchical scheduling of real-time systems," in *Proceedings of the 16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA '11)*, September 2011, pp. 1 – 8.

[5] J. Stankovic, C. Lu, S. Son, and G. Tao, "The case for feedback control real-time scheduling," in *Proceedings of the 11th Euromicro Conference on Real-Time Systems (ECRTS '99)*, June 1999, pp. 11 – 20.

[6] C. Lu, J. Stankovic, G. Tao, and S. Son, "Design and evaluation of a feedback control EDF scheduling algorithm," in *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS '99)*, December 1999, pp. 56 – 67.

[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. 1/2, pp. 85 – 126.

[8] C. Basaran, M. H. Suzer, K.-D. Kang, and X. Liu, "Robust fuzzy CPU utilization control for dynamic workloads," *Journal of Systems and Software*, vol. 83, no. 7, pp. 1192 – 1204, July 2010.

[9] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzen, "Feedbackfeed-forward scheduling of control tasks," *Real-Time Systems*, vol. 23, no. 1/2, pp. 25 – 53, July 2002.

[10] C. Mercer, S. Savage, and H. Tokuda, "Processor capacity reserves: operating system support for multimedia applications," in *Proceedings of the International Conference on Multimedia Computing and Systems*, May 1994, pp. 90 – 99.

[11] A. Mok, X. Feng, and D. Chen, "Resource partition for real-time systems," in *Proceedings of the 7th Real-Time Technology and Applications Symposium (RTAS '01)*, May 2001, pp. 75 – 84.

[12] F. Zhang and A. Burns, "Analysis of hierarchical EDF pre-emptive scheduling," in *Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS '07)*, December 2007, pp. 423 – 434.

[13] T.-W. Kuo and C.-H. Li, "A fixed-priority-driven open environment for real-time applications," in *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS '99)*, December 1999, pp. 256 – 267.

[14] L. Almeida and P. Pedreiras, "Scheduling within temporal partitions: response-time analysis and server design," in *Proceedings of the 4th ACM International Conference on Embedded Software (EMSOFT '04)*, September 2004, pp. 95 – 103.

[15] G. Lipari and S. K. Baruah, "Efficient scheduling of real-time multi-task applications in dynamic systems," in *Proceedings of the 6th IEEE Real Time Technology and Applications Symposium (RTAS '00)*, May 2000, pp. 166–.

[16] G. Lipari, J. Carpenter, and S. Baruah, "A framework for achieving inter-application isolation in multiprogrammed, hard real-time environments," in *Proceedings of the 21st IEEE Real-time Systems Symposium (RTSS '00)*, November 2000, pp. 217 – 226.

[17] J. Regehr and J. Stankovic, "HLS: a framework for composing soft real-time schedulers," in *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS '01).*, December 2001, pp. 3 – 14.

[18] P. Goyal, X. Guo, and H. M. Vin, "A hierarchical CPU scheduler for multimedia operating systems," in *Proceedings of the 2nd USENIX Symposium on OS Design and Implementation (OSDI '96)*, 1996.

[19] H. Leontyev and J. H. Anderson, "A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees," in *Proceedings of the 20th Euromicro Conference on Real-Time Systems (ECRTS '08)*, July 2008, pp. 191 – 200.

[20] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole, "Analysis of a reservation-based feedback scheduler," in *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS '02)*, December 2002, pp. 71 – 80.

[21] L. Abeni and G. Buttazzo, "Resource reservation in dynamic real-time systems," *Real-Time Systems*, vol. 27, no. 2, pp. 123 – 167, July 2004.

[22] V. Romero Segovia, "Adaptive CPU resource management for multicore platforms," Licentiate Thesis, September 2011.

[23] L. Sha, J.P.Lehoczky, and R. Rajkumar, "Solutions for some practical problems in prioritized preemptive scheduling," in *Proceedings of the Real-Time Systems Symposium (RTSS '86)*, July 1986, pp. 181 – 191.

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

[25] M. Sugeno, *Industrial applications of fuzzy control*. Elsevier Science Pub. Co., 1985.

[26] M. G. I. J. Nagrath, *Control systems engineering*. Anshan, 2008.

[27] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Reliability Engineering and System Safety*, vol. 91, no. 9, pp. 992 – 1007, 2006.

[28] C. C. Wust, L. Steffens, W. F. J. Verhaegh, R. J. Bril, and C. Hentschel, "QoS control strategies for high-quality video processing," *Real-Time Systems*, vol. 30, no. 1-2, pp. 3 – 12, May 2005.

[29] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Arzen, "How does control timing affect performance? analysis and simulation of timing using jitterbug and truetime," *Control Systems, IEEE*, vol. 23, no. 3, pp. 16 – 30, June 2003.