Real-Time Syst
DOI 10.1007/s11241-015-9224-1

Hard real-time guarantees in feedback-based resource reservations

Alessandro Vittorio Papadopoulos[1] · **Martina Maggio**[1] · **Alberto Leva**[2] · **Enrico Bini**[3]

Abstract Resource reservation is a technique that allows isolating applications from interfering among each other. In the most classic setting, this method requires the periodic allocation of a given budget of resource over time. However, in reality, the actual budget allocation may deviate from its ideal value. Examples of causes of this deviation are: the presence of a system tick, the usage of shared resources, the self-blocking on I/O operations, etc. Since control techniques are an effective mean to deal with uncertainties and disturbances, unknown at design time but bounded, in this paper we propose to use feedback to achieve the target budget allocation, which may have deviated due to on-line events. The proposed scheme, called Self-Adaptive Server (SAS), is described and analyzed. We prove that the controller gain, which maximizes the resource delivered to the application, is $\frac{3-\sqrt{5}}{2}$. We also implemented the scheduler on a lightweight operating system for a microcontroller. Thanks to the extremely simple implementation, SAS servers are well suited for low-overhead resource isolation mechanisms with proved real-time guarantees.

✉ Enrico Bini
e.bini@sssup.it

Alessandro Vittorio Papadopoulos
alessandro.papadopoulos@control.lth.se

Martina Maggio
martina.maggio@control.lth.se

Alberto Leva
alberto.leva@polimi.it

[1] Lund University, Ole Romers Väg 1, Lund, Sweden

[2] Politecnico di Milano, Piazza Leonardo da Vinci 32, 20100 Milano, Italy

[3] Scuola Superiore Sant'Anna, Via G. Moruzzi 1, 56124 Pisa, Italy

Published online: 16 April 2015

Keywords Compositional analysis · Resource reservation · Feedback control · Hard real-time systems · Supply bound function · Minimizing ℓ_∞ gain

1 Introduction

In multi-tasking environments, it is necessary to share the computing capacity among the demanding applications. In operating systems (OSes) such an arbitration is accomplished by the scheduler.

Since the initial work by Corbató et al. (1962), plenty of schedulers were proposed to solve this problem, each one with different characteristics in terms of fraction of allocated resource, management of shared resources, delay of the resource allocation, overhead, etc.

One of the most popular technique is resource reservation (Mercer et al. 1993), in which an application is assigned a *server* that is in charge to reserve a share of the CPU time. Applications running within a server have the illusion of running over a slower fully dedicated processor. Several schedulers are designed according to these design principles: P-fair (Baruah et al. 1996) and the Constant Bandwidth Server (CBS) (Abeni and Buttazzo 1998) are just two notable examples.

Servers can indeed provide the highly desired feature of *isolation*: if an application misbehaves by entering, for example, in livelock, then the misbehavior is confined to the application itself. Resource reservation servers, then, prevent possible misbehaviors of applications to affect the correctly functioning ones.

The basic principle of servers is the periodic allocation of a budget of CPU time. For example, a server with budget 10 ms and period 100 ms provides 10 % of the CPU time to the application. Servers of this type can today be easily be implemented through the `SCHED_DEADLINE` Linux scheduling class (Faggioli et al. 2009), which is now part of the kernel from version 3.14.

To enforce the isolation between applications and not to exceed the available computing capacity, servers must release the CPU when the assigned budget is consumed. However, there are a number of circumstances that may prevent the release of the CPU at the precise budget exhaustion time. Examples of these situations are:

- the presence of a system tick, which enforces all scheduling events to be aligned at a multiple of such a tick;
- the usage of shared resources: if the server budget expires when the application is holding a lock, then either the server is allowed to temporarily exceed the budget until the lock is released (Behnam et al. 2010), or it is suspended earlier if the remaining budget is not enough to complete the critical section (Bertogna et al. 2009);
- if the application is waiting for some I/O event, then the server may release the CPU even if there is still budget available.

To cope with the variability of the allocated budget at run-time, we propose to use a feedback action. Regardless of the reasons of the variation from the target budget, which are not the focus of this paper, feedback is capable to follow the target allocation with guarantees of stability and adaptation rate.

The use of feedback in server mechanisms is not new. To properly compare our contribution with the existing literature, below we review the most relevant results in the area.

In Sect. 2 we discuss the necessary background functional to the rest of the presentation. In Sect. 3 we present the proposed Self-Adaptive Server (SAS), while in Sect. 4 we derive the formal real-time guarantees for it. Section 5 presents an optimality design principle for SAS. Section 6 discusses an implementation of SAS, while Sect. 7 concludes the paper.

1.1 Related work

The application of feedback to the resource management is not new. The main motivation of feedback-based resource management is the necessity to cope with time-varying and unpredictable loads. In presence of load variations, controllers may adjust the amount of allocated resource. Also, the application may vary the QoS to adjust the resource demand. Feedback was successfully applied to control the memory allocation (Storm et al. 2006) and delays in web servers (Ku et al. 2001). The application of feedback to scheduling problems, which is more closely related to the presented research, was also investigated by Stankovic et al. (1999), Cervin and Eker (2000), and Abeni et al. (2002), just to mention a few. All these works applies to soft real-time tasks only, in which deadlines may be occasionally missed. Our work, instead, assumes that the load generated by the applications is known and has hard deadlines. The feedback is used to compensate for run-time events, which may produce a deviation from the target resource allocation.

When applications must be guaranteed to meet hard deadlines, a common model for the time supplied by a server is the so-called supply function $\mathsf{sbf}(t)$, which represents the minimum amount of time supplied by the server to the application in any interval of length t (details on the supply function will be recalled in Sect. 2). After the supply function $\mathsf{sbf}(t)$ is computed, the application deadlines are guaranteed, through a set of inequalities, which ensure that the $\mathsf{sbf}(t)$ is never less than the time needed by the application to be scheduled. This technique was extensively applied in different research areas and sometime under different names. A sample of the earlier works in this area is listed next. Mok et al. (2001) used the supply function to guarantee fixed/dynamic priority tasks. Cruz (1991) introduced the network calculus, in which the same supply/demand analysis was applied to network elements. Inspired by the network calculus, Thiele et al. (2000) proposed the real-time calculus to analyze real-time constraints.

The SAS is a simplification of the “I+PI” scheme recently proposed by Maggio et al. (2014). The I+PI scheme applied control theory to the design of a scheduler based on an equation-based model, that captured the dynamic of the CPU allocation. The main contribution of the scheme was to relegate to disturbances to be counteracted any action that was not under the scheduler control, for example yields on the application side and critical sections. The scheme was proposed, implemented and validated with some case studies. However, there was no guarantee on the scheduler behavior in the general case, i.e., no hard guarantee on the supply function of this scheme was given for the I+PI scheme.

The original contributions of this paper are:

- the exact characterization of SAS servers in terms of supply function;
- the optimal design of the controller within the SAS server, which maximizes the computation delivered to the application.

2 Background on supply functions

In the context of interest of this paper, applications are isolated from interfering with each other by running within a *server*, which provides the illusion of a dedicated less performing processor. Since the analysis of applications within servers is based on the derivation of the *supply bound function* of a server (Mok et al. 2001), we allow any application model translating the computational demand into a function over time. Suitable application models range from a simple set of periodic tasks scheduled by Fixed Priority (FP) or Earliest Deadline First (EDF), to the extended digraph task model with k global constraints (k-EDRT) (Stigge et al. 2011), the event-stream task model (Gresser 1993; Richter and Ernst 2002), or arrival curves in network/real-time calculus (Cruz 1991; Thiele et al. 2000). Hence, we do not focus here on the model of application demand, but rather on the resource provisioning on server side.

The *supply bound function* $\mathsf{sbf}(t)$ of a server (Mok et al. 2001) is defined, in words, as the minimum amount of resource provided in any interval of length t. More formally, let $s(t)$ be the schedule function of the server that is

$$s(t) = \begin{cases} 1 & \text{the server allocates resource at} t \\ 0 & \text{the server does not allocate resource at} t, \end{cases} \tag{1}$$

then, the $\mathsf{sbf}(t)$ is such that it is

$$\forall t_0, t, \quad \mathsf{sbf}(t) \leq \int_{t_0}^{t_0+t} s(\tau)\, d\tau. \tag{2}$$

Hence, in any interval of length t we are certain that there is at least $\mathsf{sbf}(t)$ resource available to the application. Notice, that a valid function is $\mathsf{sbf}(t) = 0$ for all t. However, such a pathological case is not useful since it does not allow to schedule any application. To guarantee applications with real-time constraints, the challenge is to compute the *largest* $\mathsf{sbf}(t)$, satisfying (2).

A general model of the resource supply over time, which is convenient for the purpose of this paper and to generalize various approaches, is given by a sequence of supply intervals, interleaved with a sequence of idle intervals. The lengths of the supply intervals are represented by the sequence $\{S(k)\}_{k=1,2,\ldots}$, while the lengths of idle intervals is represented by the sequence $\{Z(k)\}_{k=1,2,\ldots}$. Without loss of generality, we set the time $t = 0$ at the instant when the first resource supply $S(1)$ starts. Figure 1 illustrates the resource provisioning over time according to the above introduced

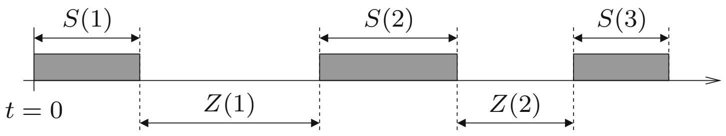

Fig. 1 Budget provisioning in servers

notation. Also notice that any overhead for switching from one server to another can be added to the idle intervals, without requiring any additional notation (measures of this overhead are reported at the end of the paper in Sect. 6).

For pair of sequences $S(k)$ and $Z(k)$ (resulting from any server logic), we can compute the schedule function $s(t)$ of (1) in a straightforward way, which is

$$s(t) = \begin{cases} 1 & 0 \le t - \sum_{k=1}^{n} (S(k) + Z(k)) < S(n+1) \\ 0 & 0 \le t - \sum_{k=1}^{n} (S(k) + Z(k)) - S(n+1) < Z(n+1) \end{cases} \tag{3}$$

for any $n \in \mathbb{N}$.

The derivation of a valid supply function $\mathsf{sbf}(t)$ satisfying (2) is made by standard techniques (Mok et al. 2001). Next Lemma, then, is a simple repetition of well known results adapted to the newly introduced notation of $S(k)$ and $Z(k)$.

Lemma 1 *A server characterized by a sequence of supply intervals of length* $\{S(k)\}_{k=1,2,\dots}$ *interleaved by a sequence of idle intervals of length* $\{Z(k)\}_{k=1,2,\dots}$ *has the following supply bound function*

$$\mathsf{sbf}(t) = \min\{t - \sigma_Z(n), \sigma_S(n)\}, \quad t \in I_n, n \in \mathbb{N} \tag{4}$$

with the sequence of intervals $\{I_n\}_{n \in \mathbb{N}}$ *defined as*

$$I_n = \begin{cases} \big[0, \sigma_Z(1)\big) & n = 0 \\ \big[\sigma_Z(n) + \sigma_S(n-1), \sigma_Z(n+1) + \sigma_S(n)\big) & n \ge 1 \end{cases} \tag{5}$$

and with

$$\sigma_S(n) = \inf_{n_0} \sum_{k=n_0}^{n_0+n-1} S(k),$$
$$\sigma_Z(n) = \sup_{n_0} \sum_{k=n_0}^{n_0+n-1} Z(k), \tag{6}$$

properly extended at $n = 0$ *with* $\sigma_S(0) = \sigma_Z(0) = 0$.

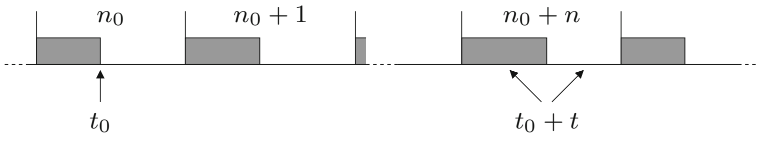

Fig. 2 Illustrating the proof of Theorem 1

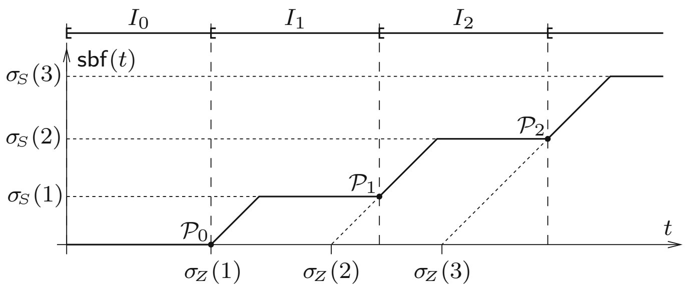

Fig. 3 An example of supply bound function

Proof With the definitions of (5), (6) in mind, we first observe that $\forall t \in I_0$ we have $\mathsf{sbf}(t) = 0$ since $\sigma_Z(1)$ is the length of the longest interval with no resource. If $t > \sigma_Z(1)$, then some resource must be available in any interval $[t_0, t_0 + t]$.

For any t, we first observe that among the intervals $[t_0, t_0 + t]$ of length t with minimal amount of resource, there is always one interval with t_0 coinciding with the end of service time $S(n_0)$, at some round n_0. Let then $n_0 + n$, with $n \geq 1$, be the index of the round in which the instant $t_0 + t$ falls. We distinguish between two cases: some resource is scheduled at $t_0 + t$ or not (see also Fig. 2 for an illustration of the two cases).

If the resource is scheduled at $t_0 + t$, then the resource available in $[t_0, t_0 + t]$ amounts to $t - \sum_{k=n_0}^{n_0+n-1} Z(k)$, whose minimal value is $t - \sigma_Z(n)$. If no resource is scheduled at t_0+t, then the resource available in $[t_0, t_0+t]$ amounts to $\sum_{k=n_0+1}^{n_0+n} S(k)$, whose minimal value is $\sigma_S(n)$. The minimum among the two cases is then given by (4), which concludes the proof. □

Notice that the expression of $\mathsf{sbf}(t)$ in (4) generalizes many other resource models. For example, by setting the minimum sum of n consecutive budgets as $\sigma_S(n) = nQ$ and the maximum sum of n consecutive idle intervals as $\sigma_Z(n) = n(P - Q) + D - Q$, the resulting $\mathsf{sbf}(t)$ of (4) is equivalent to the supply function of the EDP resource model (Easwaran et al. 2007) with budget Q, period P, and deadline D.

For convenience, an illustration of the $\mathsf{sbf}(t)$ of (4) is given in Fig. 3. In the figure, we also draw on top the extent of the intervals I_n, as well as the points $\mathcal{P}_n$ at the "bottom-right corners" of the curve $\{(t, \mathsf{sbf}(t)) : t \geq 0\}$.

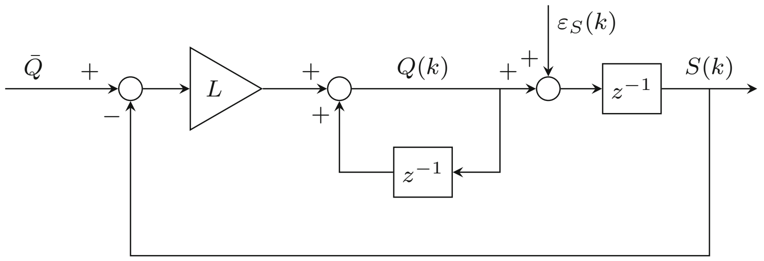

Fig. 4 Block diagram of the controller of SAS servers

3 The self-adaptive server

In classic periodic servers, a budget $\bar{Q}$ is allocated every period $\bar{P}$. However, a number of causes may disturb such an ideal allocation. A non-exhaustive list of causes of this disturbances are:

- the usage of shared resources, which may prevent the release of the processor at the budget expiration instant, depending on the protocol adopted to manage shared resources (Behnam et al. 2010; Bertogna et al. 2009);
- the self-suspension of the application occurring earlier than the budget expiration;
- the necessity for the application to synchronize with some I/O event, such as the collection of a sensor measurement;
- the presence of a system tick, which force scheduling events to occur only at some predetermined instants.

We propose to compensate for these variations with a feedback mechanism. We propose the SAS, which is a variation of the "I+PI" server proposed by Maggio et al. (2014). A SAS server aims to provide a budget $\bar{Q}$, every period $\bar{P}$. However, the actual amount of service time $S(k)$ that the server provides at the k-th round may be different than the set value $Q(k)$, because of the above listed reasons. At round k, the deviation between the desired budget allocation $Q(k)$ and the actual allocated time $S(k)$, is denoted by $\varepsilon_S(k) = S(k) - Q(k)$. We often call $\varepsilon_S(k)$ *disturbance*. Also, at every round SAS servers do allocate resource non-preemptively. From this property, it follows that $S(k)$ is both the service time and the length of the supply interval during k-th round.

Figure 4 shows the control logic of a SAS server. The same logic is expressed by the equations below.

$$S(0) = Q(0) = \bar{Q}, \tag{7}$$

$$S(k+1) = Q(k) + \varepsilon_S(k), \tag{8}$$

$$Q(k+1) = Q(k) + L(\bar{Q} - S(k)). \tag{9}$$

We choose such a simple control logic, not to consume too much computation time in executing the control logic itself. In absence of disturbances ($\varepsilon_S(k) = 0$), the allocated budget $S(k)$ is constantly equal to the target value $\bar{Q}$, as desired. The

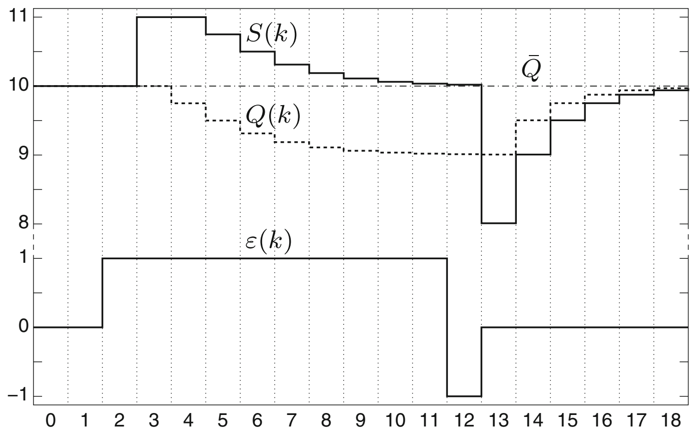

Fig. 5 Example of dynamics for the allocated budget

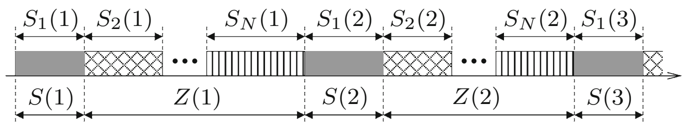

Fig. 6 N servers in loop

controller gain L adjusts the budget in response to deviations from the target value $\bar{Q}$. When $L \in (0, 1)$, then the controlled system is stable (Leva and Maggio 2010) (see also Sect. 5). For any stabilizing L, the controller is also capable to reject constant disturbances (the step response from $\varepsilon_S(k)$ to $S(k)$ tends to zero Leva and Maggio 2010). In fact, if $\varepsilon_S(k) = \bar{\varepsilon}_S$ and $L \in (0, 1)$ then, after a transient, $Q(k) \to \bar{Q} - \bar{\varepsilon}_S$ so that $S(k) \to \bar{Q}$. An example of resource provisioning is shown in Fig. 5. In this figure, the target budget per round is $\bar{Q} = 10$ and the controller gain is $L = 1/4$. In presence of a constant disturbance ($\varepsilon(k) = 1$ up to $k = 11$), the server is still capable of allocating the desired target budget, by assigning $Q(k) \to 9$ which implies $S(k) \to 9+1 = \bar{Q}$. If, at some instant ($k = 12$ in the figure), the disturbance switches from 1 to -1 and then sets to 0, then the allocated budget can suddenly decrease to 8 and then it will reach again the target budget.

Following the model introduced in Sect. 2, consecutive budgets $S(k)$ and $S(k+1)$ are separated by a time-varying sequence of idle intervals of length $Z(k)$, as also shown in Fig. 1. Depending on the logic of the server the sequence of idle intervals may obey to different rules.

If, for example, the server enforces the start times of the supply intervals to be separated by exactly $\bar{P}$, then it is simply $Z(k) = \bar{P} - S(k)$. However, for the same reasons that prevent an exact budget allocation and due to the interference with other servers, such a condition is difficult to guarantee.

Another condition of interest is when N SAS servers coexist and are scheduled in loop, as shown in Fig. 6. Let then the i-th server be characterized by a target budget $\bar{Q}_i$, a computed budget $Q_i(k)$, an allocated budget $S_i(k)$, and a disturbance $\varepsilon_i(k)$. Then the dynamics of the i-th server is analogous to the one of (7)–(9), that is

$$\begin{cases} S_i(0) &= Q_i(0) = \bar{Q}_i, \\ S_i(k+1) &= Q_i(k) + \varepsilon_i(k), \\ Q_i(k+1) &= Q_i(k) + L(\bar{Q}_i - S_i(k)). \end{cases} \tag{10}$$

We assume, without loss of generality, that the first server is the one under analysis, while the servers from the second to the N-th are the "adversaries", meaning that the time allocated to them is the idle time for the first server, then the intervals $Z(k)$ of idle time are exactly the sum of the $N-1$ server budgets, from $S_2(k)$ to $S_N(k)$. The dynamics of the sequence of idle intervals $Z(k)$ can then be obtained by summing all equations in (10) for i from 2 to N, that is

$$Z(0) = X(0) = \bar{P} - \bar{Q}, \tag{11}$$

$$Z(k+1) = X(k) + \varepsilon_Z(k), \tag{12}$$

$$X(k+1) = X(k) + L((\bar{P} - \bar{Q}) - Z(k)), \tag{13}$$

in which we set

$$\begin{cases} \bar{P} - \bar{Q} = \sum_{i=2}^{N} \bar{Q}_i, & \varepsilon_Z(k) = \sum_{i=2}^{N} \varepsilon_i(k), \\ X(k) \quad = \sum_{i=2}^{N} Q_i(k), & Z(k) = \sum_{i=2}^{N} S_i(k). \end{cases} \tag{14}$$

Hence, the control logic of the idle intervals is analogous to the one of (7)–(9) with disturbance $\varepsilon_Z(k)$, the same controller gain L, and target $\bar{P} - \bar{Q}$. From now on, we assume then that the idle time follows the dynamics of (14). Nonetheless, much of the theory developed next can be borrowed also in the case of different logic of the idle intervals.

Notice that the hypothesis of serving N servers in loop, as illustrated in Fig. 6, implies that all the N servers share a common period, which limits to some extent the applicability of the model. In this regard, we observe that by a properly selecting the lengths $S(k)$ and $Z(k)$ of the supply and idle intervals, any resource schedule can be replicated, even with servers with different periods. If, for example, three servers have to be scheduled, with budget/period pair respectively equal to $(1, 3)$, $(2, 4)$, and $(1, 6)$, then their resulting EDF schedule is the one depicted in Fig. 7. The very same resource schedule can be achieved by setting the sequence $S(k)$ of target budgets equal to $\{1, 1, 1, 2, 1, 1, 2, 1, 1, 2, \dots\}$ and the sequence $Z(k)$ of lengths of idle intervals equal to $\{3, 2, 3, 3, 2, 3, \dots\}$. Notice, however, that this paper is not aiming at translating the server parameters into patterns of resource schedule, nor, more in general, at determining the budget allocation rule. Rather, we aim at making *any budget allocation policy* robust against disturbances.

We conclude this introduction to SAS servers with some natural questions, which needs to be investigated: Is the SAS server governed by (7)–(9) capable to provide the target bandwidth $\bar{Q}/\bar{P}$? With bounded disturbances, what is the maximum deviation

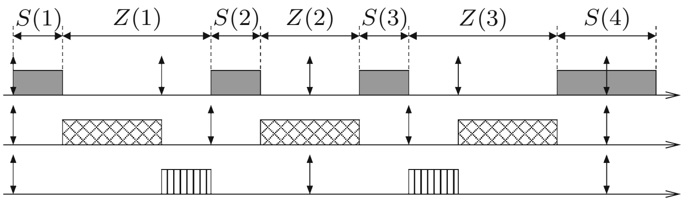

Fig. 7 Example of resource schedule of EDF servers

between the sequence of service/idle times $S(k)$ and $Z(k)$, from their ideal values $\bar{Q}$ and $\bar{P} - \bar{Q}$, respectively? What is the best choice for the controller gain L? In the following, we will start the investigation on the minimum amount of service time guaranteed by SAS.

4 Supply function of SAS servers

The definition of $\mathsf{sbf}(t)$ of Lemma 1 does depend on the values of $\sigma_S(n)$ and $\sigma_Z(n)$ of (6). However, it is unclear for the moment how to compute, or at least to estimate, the expressions of (6) for SAS servers.

First of all, we must necessarily assume that disturbances $\varepsilon_S(k)$ and $\varepsilon_Z(k)$ are bounded, as follows

$$\forall k \in \mathbb{N}, \qquad |\varepsilon_S(k)| \le \bar{\varepsilon}_S, \quad |\varepsilon_Z(k)| \le \bar{\varepsilon}_Z. \tag{15}$$

If disturbances are not bounded, it is not possible to guarantee any service time, with any resource allocation policy. We remark that a bound on the disturbances can often be provided. For example, in systems with system tick, in which the resource can be allocated and released only at multiple of system tick, the bound is simply represented by the size of the system tick. In protocols that manage shared resources by preventing the preemption until the shared resource is unlocked, the disturbance is bounded by the longest resource holding time (Bertogna et al. 2009).

If disturbances are bounded as in (15), then the quantities $\sigma_S(n)$ and $\sigma_Z(n)$, necessary to define the supply function from (4), are:

$$\begin{aligned} \sigma_S(n) &= \inf_{|\varepsilon_S(k)| \le \bar{\varepsilon}_S, n_0} \sum_{k=n_0}^{n_0+n-1} S(k), \\ \sigma_Z(n) &= \sup_{|\varepsilon_Z(k)| \le \bar{\varepsilon}_Z, n_0} \sum_{k=n_0}^{n_0+n-1} Z(k), \end{aligned} \tag{16}$$

The controller governing both the supply and idle intervals, described in Eqs. (7)–(9) and (11)–(13) respectively, is a linear time-invariant (LTI) system. Then, the computation of $\sigma_S(n)$ and $\sigma_Z(n)$ of (16) is related to the maximum/minimum variation of the output of a LTI system, in presence of bounded input. The analysis of this property is known in control theory as "bounded-input bounded-output (BIBO) stability". Below, we enunciate the following standard result on system theory, which is useful for our purpose.

Lemma 2 *Let $u(k) \in \mathbb{R}$, $h(k) \in \mathbb{R}$, and $y(k) = u(k) * h(k)$ be the input, impulse response, and output of a LTI discrete-time causal system (with the symbol "$*$" denoting the signal convolution). Then,*

$$\sup_{\forall j : |u(j)| \leq 1} |y(k)| = \sum_{j=0}^{\infty} |h(j)|. \tag{17}$$

Proof The interested reader can find the proof of the continuous-time equivalent problem in Sect. II.6 of the book "Feedback Systems: Input-Output Properties", by Desoer and Vidyasagar (1975). □

In signal norm terminology, we say that the l_∞ gain of a system is the l_1 norm of its impulse response.

Such a lemma, which relates the bound on the output to the bound on the input, is exploited in the next theorem to find the values of $\sigma_S(n)$ and $\sigma_Z(n)$.

Theorem 1 *Let $g(k)$ be the step response of the SAS server of Eqs. (7)–(9), with controller gain L, and let the disturbances $\varepsilon_S(k)$ and $\varepsilon_Z(k)$ be bounded by $\bar{\varepsilon}_S$ and $\bar{\varepsilon}_Z$, respectively, as in (15). Then:*

$$\sigma_S(n) = n\bar{Q} - \bar{\varepsilon}_S \mathcal{N}(n, L) \tag{18}$$

$$\sigma_Z(n) = n(\bar{P} - \bar{Q}) + \bar{\varepsilon}_Z \mathcal{N}(n, L), \tag{19}$$

with

$$\mathcal{N}(n, L) = \sum_{k=0}^{\infty} |g(k) - g(k-n)|. \tag{20}$$

Proof We first prove (18). Since the controller is linear, by the superposition principle the output $S(k)$ is equal to the sum of the output when $\varepsilon_S(k) = 0$, plus the output when $\bar{Q} = 0$, that is

$$S(k) = \underbrace{\bar{Q}}_{\text{when}\,\varepsilon_S(k)=0} + \underbrace{\varepsilon_S(k) * h(k)}_{\text{when}\,\bar{Q}=0}$$

where the symbol "$*$" denoted the signal convolution and $h(k)$ the response to an impulse on the input $\varepsilon_S(k)$.

Then,

$$\sum_{k=n_0}^{n_0+n-1} S(k) = n\bar{Q} + \sum_{k=n_0}^{n_0+n-1} \varepsilon_S(k) * h(k) \tag{21}$$

$$= n\bar{Q} + \sum_{k=n_0}^{n_0+n-1} \sum_{\ell=-\infty}^{\infty} \varepsilon_S(\ell) h(k-\ell) \tag{22}$$

$$= n\bar{Q} + \sum_{k=0}^{n-1} \sum_{\ell=-\infty}^{\infty} \varepsilon_S(\ell) h(k+n_0-\ell) \tag{23}$$

$$= n\bar{Q} + \sum_{k=0}^{n-1} \sum_{\ell=-\infty}^{\infty} \varepsilon_S(\ell+n_0) h(k-\ell) \tag{24}$$

$$= n\bar{Q} + \sum_{\ell=-\infty}^{\infty} \varepsilon_S(\ell+n_0) \sum_{k=0}^{n-1} h(k-\ell) \tag{25}$$

$$= n\bar{Q} + \varepsilon_S(k+n_0) * \sum_{k=0}^{n-1} h(k) \tag{26}$$

$$= n\bar{Q} + \varepsilon_S(k+n_0) * \left(\sum_{k=0}^{\infty} h(k) - \sum_{k=n}^{\infty} h(k) \right) \tag{27}$$

$$= n\bar{Q} + \varepsilon_S(k+n_0) * (g(k) - g(k-n)) \tag{28}$$

$$= n\bar{Q} + \bar{\varepsilon}_S \frac{\varepsilon_S(k+n_0)}{\bar{\varepsilon}_S} * (g(k) - g(k-n)), \tag{29}$$

where:

- from (21) to (26), we used the definition and the properties of the signal convolution;
- from (26) to (27), we wrote a finite sum as difference between two series; and
- from (27) to (28), by linearity of the system, we wrote the sum of impulse responses as response to sum of impulses, which is the step response $g(k)$.

Notice that the equations above prove the BIBO stability of the controlled system.

Now, by observing that:

1. the value of $\sigma_S(n)$ to be found is a lower bound of (29); and
2. the signal $\varepsilon_S(k+n_0)/\bar{\varepsilon}_S$ is bounded by 1;

then Lemma 2 asserts that

$$\left| \frac{\varepsilon_S(k+n_0)}{\bar{\varepsilon}_S} * (g(k) - g(k-n)) \right| \leq \mathcal{N}(n, L), \tag{30}$$

with $\mathcal{N}(n, L)$ defined as (20). Notice that the causality of the system implies that $g(k) = 0$ for all $k < 0$.

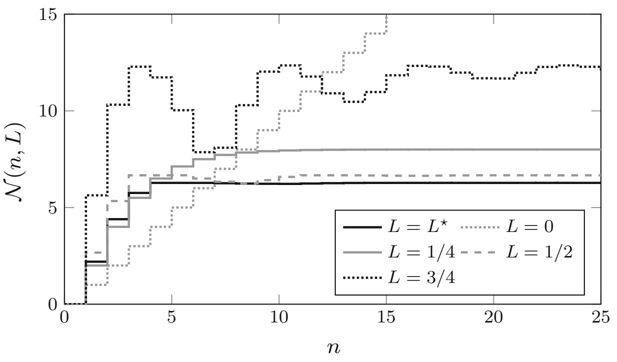

Fig. 8 Value of $\mathcal{N}(n, L)$, for some L

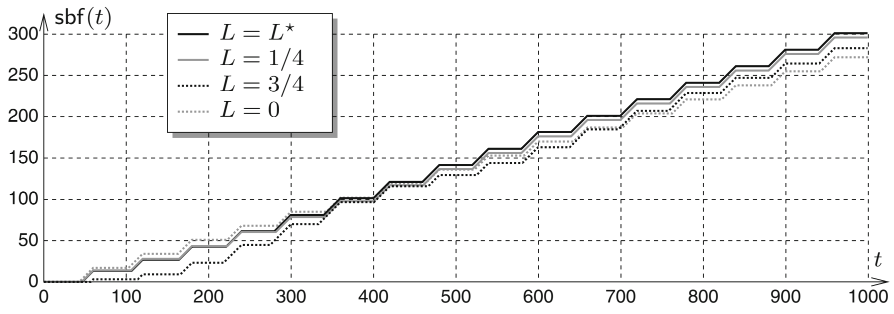

Fig. 9 Supply function of SAS servers when $P = 60$, $Q = 20$, and $\bar{\varepsilon}_S = \bar{\varepsilon}_Z = 3$

Hence, the value of (18) follows immediately, by observing that $\sigma_S(n)$ is a lower bound to (29).

The expression $\sigma_Z(n)$ of (19) follows by an analogous proof, reminding that:

- the reference value of $Z(k)$ is $\bar{P} - \bar{Q}$ (rather than $\bar{Q}$);
- the disturbance $\varepsilon_Z(k)$ is bounded by $\bar{\varepsilon}_Z$ (rather than $\bar{\varepsilon}_S$);
- the controller gain is L;
- $\sigma_Z(n)$ is the upper bound on the sum of n consecutive $Z(k)$ (rather than the lower bound on the sum of n consecutive $S(k)$).

This concludes the proof. $\square$

Theorem 1, finally, establishes the relationship between the controller gain L of SAS servers and the delivered supply function. Such a relationship is given through the quantity $\mathcal{N}(n, L)$ of (20). In Fig. 8, we plot $\mathcal{N}(n, L)$ for the values of $L \in \{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, L^\star\}$, with $L^\star = \frac{3-\sqrt{5}}{2}$, while in Fig. 9 the supply function $\mathsf{sbf}(t)$, as characterized in Lemma 1, is drawn for $L \in \{0, \frac{1}{4}, \frac{3}{4}, L^\star\}$ (the $\mathsf{sbf}(t)$ for $L = \frac{1}{2}$ is omitted as it overlaps significantly with the one for $L = L^\star$).

From Fig. 9, it can be observed that for small values of t, the supply function $\mathsf{sbf}(t)$ is larger when $L = 0$, that is not to compensate for disturbances. However, such a choice is not capable to asymptotically guarantee the target bandwidth of Q/P (see

Eq. 35 in Sect. 5). Instead, as it will be demonstrated next in Sect. 5, any controller with $L \in (0, 1)$ can guarantee the asymptotic bandwidth of Q/P. Among these values, the choice of $L = L^\star = \frac{3-\sqrt{5}}{2}$ achieves the lowest value for the limit $\lim_n \mathcal{N}(n, L)$, which in turn will produce the asymptotically largest possible supply bound function $\mathsf{sbf}(t)$.

5 Optimal design of SAS servers

The last section was concluded with a result that establishes a link between the supply bound function $\mathsf{sbf}(t)$ of a SAS server and controller gain L. In this section, we address the problem of selecting the controller gain L.

First, we find the condition which guarantees that all budgets are always non-negative. In fact, as it can be observed in (18), for large $\bar{\varepsilon}_S$ the supply function can indeed be negative meaning that, meaning that the necessary compensation may exceed the budget. Next Lemma establishes an upper bound on the maximum controllable disturbance.

Lemma 3 *If the disturbance $\bar{\varepsilon}_S$ and the budget Q are such that*

$$\frac{\bar{\varepsilon}_S}{Q} \le \frac{1}{\mathcal{N}(1, L)}, \tag{31}$$

then it is always $\sigma_S(n) \ge 0$.

Proof From (18), we have

$$\forall n,\ \sigma_S(n) = n\bar{Q} - \bar{\varepsilon}_S \mathcal{N}(n, L) \ge 0 \quad \Leftrightarrow \quad \frac{\bar{Q}}{\bar{\varepsilon}_S} \ge \sup_n \left\{ \frac{\mathcal{N}(n, L)}{n} \right\}. \tag{32}$$

If we denote by $h(k)$ and $g(k)$ the impulse and step response of a SAS server, respectively, from its definition of (20), $\mathcal{N}(n, L)$ can be upper bounded as follows,

$$\begin{aligned}\mathcal{N}(n, L) &= \sum_{k=0}^{\infty} |g(k) - g(k-n)| = \sum_{k=0}^{\infty} \left| \sum_{\ell=0}^{n-1} h(k-\ell) \right| \le \sum_{k=0}^{\infty} \sum_{\ell=0}^{n-1} |h(k-\ell)| \\ &= \sum_{\ell=0}^{n-1} \sum_{k=0}^{\infty} |h(k-\ell)| = \sum_{\ell=0}^{n-1} \sum_{k=0}^{\infty} |h(k)| = n \sum_{k=0}^{\infty} |h(k)| = n\,\mathcal{N}(1, L).\end{aligned}$$

Hence, it follows that $\sup_n \left\{ \frac{\mathcal{N}(n,L)}{n} \right\} = \mathcal{N}(1, L)$ and, by inverting (32), the Lemma is proved. □

Previous Lemma is a feasibility condition: if condition (31) does not hold, then the control policy of SAS servers may produce a negative server budget. In Fig. 10 we plot $1/\mathcal{N}(n,L)$ as function of L.

Hence if the maximum disturbance $\bar{\varepsilon}_S$ which needs to be compensated is larger than half of the target budget Q, then SAS servers are not suited.

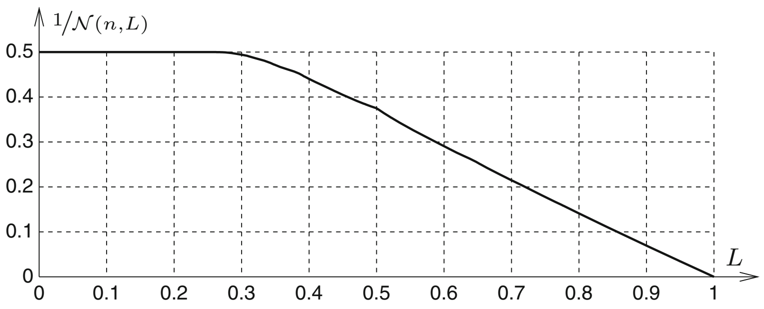

Fig. 10 Maximum disturbance handled by SAS servers

Now we seek for the controller gain, which can guarantee the largest amount of resource supply. We start by writing the asymptotic components of $\mathcal{N}(n, L)$, that is

$$\mathcal{N}(n, L) = c_1(L)n + c_0(L) + o(1). \tag{33}$$

In the case of SAS, the $c_1(L)$ component is equal to 0 for any value of $L \in (0, 1)$, due to the controller design that includes an integral action that is able to reject a constant (or step-shaped) disturbance (Åström and Hägglund 2005).

Now we can start to answer to some of the questions stated at the end of Sect. 3. For example, we can determine the bandwidth α provided by a SAS server. For this purpose, let us define the following points along the sbf curve, as

$$\mathcal{P}_n \in \{(t, \mathsf{sbf}(t)) : t = \sigma_Z(n+1) + \sigma_S(n), n \in \mathbb{N}\},$$

which are the points at the "bottom-right corner" of the sbf, also denoted by black dots in Fig. 3.

From the values of σ_S and σ_Z, of (18) and (19) respectively, and the expression of $\mathcal{N}(n, L)$ with its asymptotic components of (33), we have

$$\begin{aligned}\mathcal{P}_n &\equiv (\sigma_Z(n+1, \bar{\varepsilon}_Z) + \sigma_S(n), \sigma_S(n)) \\ &= (b_1 n + b_0 + o(1), a_1 n + a_0 + o(1))\end{aligned} \tag{34}$$

with the coefficients a_1, a_0, b_1, and b_0 conveniently defined as

$$\begin{aligned}a_1 &= \bar{Q} - \bar{\varepsilon}_S c_1(L) \\ a_0 &= -\bar{\varepsilon}_S c_0(L) \\ b_1 &= \bar{P} - \bar{\varepsilon}_S c_1(L) + \bar{\varepsilon}_Z c_1(L) \\ b_0 &= \bar{P} - \bar{Q} - \bar{\varepsilon}_S c_0(L) + \bar{\varepsilon}_Z (c_1(L) + c_0(L)).\end{aligned}$$

Thanks to this notation, we can compute the bandwidth α of a SAS server as the limit of the ratio of the coordinates of the points $\mathcal{P}_n$ of (34), that is

$$\alpha = \lim_{t\to\infty} \frac{\mathsf{sbf}(t)}{t} = \lim_n \frac{a_1 n + a_0 + o(1)}{b_1 n + b_0 + o(1)} = \frac{a_1}{b_1}$$
$$= \frac{\bar{Q} - \bar{\varepsilon}_S c_1(L)}{\bar{P} - \bar{\varepsilon}_S c_1(L) + \bar{\varepsilon}_Z c_1(L)}. \tag{35}$$

Equation (35) relates control characteristics, such as the linear coefficient $c_1(L)$ of the quantity $\mathcal{N}(n, L)$ of (20), to real-time characteristics of the server, such as its bandwidth α. In fact, if we choose $L \in (0, 1)$, then the following facts hold:

1. the system controlled with an SAS policy is asymptotically stable,
2. $\lim_k g(k)$ tends exponentially to zero,
3. the effect of a constant disturbance is rejected with no oscillations when $L \le 1/4$, with some oscillations otherwise,
4. $\mathcal{N}(n, L)$ tends to a constant, which implies
5. $c_1(L) = 0$, and then
6. the bandwidth α is exactly equal to the target value $\bar{Q}/\bar{P}$.

Among the controllers with $L \in (0, 1)$, which all provide the ideal bandwidth $\alpha = \bar{Q}/\bar{P}$, what is the best one to be chosen?

A natural design choice is to select the controller gain L, which makes the supply function $\mathsf{sbf}(t)$ as large as possible. Unfortunately, according to the actual uncertainties and disturbances that are present in the system, different values of L may give the best results at different instants t. Since, however, they are in principle unpredictable, a worst-case approach is the only viable solution for the design of L.

In real-time applications, it is common to lower bound the supply bound function $\mathsf{sbf}(t)$ by a linear function. It is then a natural choice to choose the controller gain L for which such a linear lower bound is as large as possible. Following the terminology and the notation by Mok et al. (2001), such a lower bound is written as $\alpha(t - \Delta)$, with the bandwidth α defined by the limit in (35) and Δ representing the delay of a so-called "bounded-delay partition". From the linear lower bound condition

$$\forall t \ge 0, \qquad \mathsf{sbf}(t) \ge \alpha(t - \Delta),$$

it follows that the smallest value of Δ satisfying this condition is

$$\Delta = \sup_{t\ge 0} \left\{ t - \frac{1}{\alpha}\mathsf{sbf}(t) \right\}.$$

In our SAS server, the value Δ can be computed by the coordinates of the points $\mathcal{P}_n$ of (34), along the supply function $\mathsf{sbf}(t)$. In fact, we have

$$\Delta = \lim_n \left(\underbrace{b_1 n + b_0}_{t} - \frac{b_1}{a_1} \underbrace{(a_1 n + a_0)}_{\mathsf{sbf}(t)} \right) = b_0 - \frac{b_1}{a_1} a_0,$$

which is, in case of a choice of $L \in (0, 1)$ and then $c_1(L) = 0$,

$$\Delta = \bar{P} - \bar{Q} + \left[\bar{\varepsilon}_Z + \left(\frac{\bar{P}}{\bar{Q}} - 1\right)\bar{\varepsilon}_S\right] c_0(L). \tag{36}$$

Let us now spend a few words to comment the value of Δ of (36). First, if there is no disturbance (that is $\bar{\varepsilon}_S = \bar{\varepsilon}_Z = 0$) then the longest delay is $\Delta = \bar{P} - \bar{Q}$, in accordance to well-known results on periodic resource model (Easwaran et al. 2007). Not surprisingly, Δ grows with $\bar{\varepsilon}_Z$ and $\bar{\varepsilon}_Z$: larger disturbances may have a greater impact on the worst-case resource provisioning.

The impact of the controller gain on Δ is given by the constant $c_0(L)$, which is the limit of $\mathcal{N}(n, L)$, as $n \to \infty$. A natural design target, aimed at maximizing the time available to real-time applications, would minimize Δ which is achieved by choosing L such that $c_0(L)$ is minimal. For this purpose, we state the next Lemma.

Lemma 4 *If* $L \in (0, 1)$*, then*

$$c_0(L) = \lim_n \mathcal{N}(n, L) = 2\sum_{k=0}^{\infty} |g(k)| \tag{37}$$

with $g(k)$ *being the response to a unitary step disturbance of the SAS server with controller gain* L.

Proof From the definition of $\mathcal{N}(n, L)$ of (20), we have

$$\begin{aligned}\mathcal{N}(n, L) &= \sum_{k=0}^{\infty} |g(k) - g(k-n)| \\ &= \sum_{k=0}^{n-1} |g(k) - g(k-n)| + \sum_{k=n}^{\infty} |g(k) - g(k-n)| \\ &= \sum_{k=0}^{n-1} |g(k)| + \sum_{k=0}^{\infty} |g(k) - g(k+n)|.\end{aligned} \tag{38}$$

As $n \to \infty$, the first term in (38) clearly tends to $\sum_{k=0}^{\infty} |g(k)|$. Since $\forall L \in (0, 1)$, $\lim_n g(n) = 0$, then the second term of (38) also tend to $\sum_{k=0}^{\infty} |g(k)|$. The Lemma is then proved. □

Lemma 4 relates the coefficient $c_0(L)$ to the l_1 norm of the step response $g(k)$. To find the characteristic polynomial and then the step response, we formulate the dynamics of $S(k)$ of (8)–(9) as follows

$$\begin{bmatrix} S(k+1) \\ Q(k+1) \end{bmatrix} = \overbrace{\begin{bmatrix} 0 & 1 \\ -L & 1 \end{bmatrix}}^{A} \begin{bmatrix} S(k) \\ Q(k) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \varepsilon_S(k).$$

The eigenvalues of matrix A dictate the system dynamics independently of the considered input, and can be easily found. We distinguish then the system analysis, depending on the eigenvalues placement.

If $L < \frac{1}{4}$, then the two eigenvalues are real-valued and distinct, and equal to:

$$\lambda_1 = \frac{1}{2} + \frac{\sqrt{1-4L}}{2}, \quad \lambda_2 = \frac{1}{2} - \frac{\sqrt{1-4L}}{2}.$$

In this case, the analytic expression of the step response can be obtained as

$$g(k) = \frac{\lambda_1^k - \lambda_2^k}{\sqrt{1-4L}},$$

with l_1-norm

$$\begin{aligned}\sum_{k=0}^{\infty} |g(k)| &= \sum_{k=0}^{\infty} \frac{|\lambda_1^k - \lambda_2^k|}{\sqrt{1-4L}} = \sum_{k=0}^{\infty} \frac{\lambda_1^k - \lambda_2^k}{\sqrt{1-4L}} \\ &= \frac{\frac{1}{1-\lambda_1} - \frac{1}{1-\lambda_2}}{\sqrt{1-4L}} = \frac{1}{L}.\end{aligned}$$

If $L = \frac{1}{4}$ the two eigenvalues are coincident at $\lambda_1 = \lambda_2 = \frac{1}{2}$, the step response is

$$g(k) = k2^{1-k}$$

with l_1-norm $\sum_{k=0}^{\infty} |g(k)| = 4$.

Finally, if $L > \frac{1}{4}$, then the two eigenvalues are complex conjugate that we write for convenience as

$$\lambda_1 = \rho e^{j\theta}, \quad \lambda_2 = \rho e^{-j\theta} \tag{39}$$

with

$$\rho = \sqrt{L}, \qquad \theta = \arctan\sqrt{4L-1}, \tag{40}$$

and the step response becomes

$$g(k) = \frac{2\rho^k \sin(k\theta)}{\tan\theta}. \tag{41}$$

Computing the l_1-norm of (41) requires a greater effort, since it is also negative, depending on the sign of $\sin(k\theta)$. We found an analytic expression of the l_1-norm of $g(k)$ only in the case of θ being an integer divisor of π. In this case, in fact, the periodicity of $\sin(k\theta)$ in $g(k)$ can be precisely determined. If we let $\theta = \frac{\pi}{p}$ for some positive integer p, then from (40) it means to restrict to controller gains of the form

$$L = \frac{\tan^2\frac{\pi}{p} + 1}{4}, \qquad p \geq 4. \tag{42}$$

In this case, we have

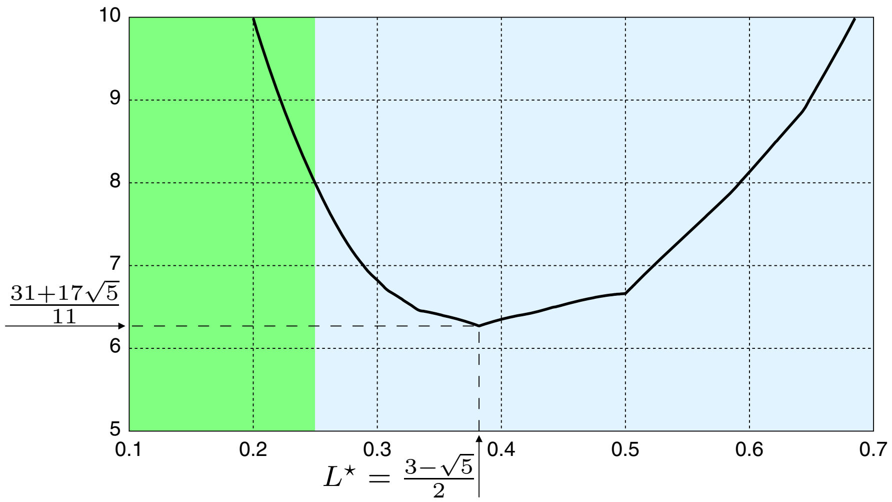

Fig. 11 Coefficient $c_0(L)$ as function of L

$$\begin{aligned}\sum_{k=0}^{\infty}|\rho^k\sin(k\theta)| &= \sum_{k=0}^{\infty}\sum_{\ell=0}^{p-1}\left|\rho^{\ell+kp}\sin((\ell+kp)\theta)\right|\\ &= \sum_{k=0}^{\infty}\rho^{kp}\sum_{\ell=0}^{p-1}\rho^{\ell}\sin(\ell\theta)\\ &= \sum_{k=0}^{\infty}\rho^{kp}\frac{\rho(1+\rho^p)\sin\theta}{1-2\rho\cos\theta+\rho^2}\\ &= \frac{1+\rho^p}{1-\rho^p}\frac{\rho\sin\theta}{\rho^2},\end{aligned}$$

where we also used the fact that $\rho\cos\theta$ is the real part of the poles of (39), which is $\frac{1}{2}$. Then, the l_1-norm of the step response is

$$\sum_{k=0}^{\infty}|g(k)| = \frac{1+\rho^p}{1-\rho^p}\frac{2\rho\cos\theta}{\rho^2} = \frac{1+\rho^p}{1-\rho^p}\frac{1}{\rho^2} = \frac{1}{L}\frac{1+L^{\frac{p}{2}}}{1-L^{\frac{p}{2}}}. \tag{43}$$

Although we are unable to compute the l_1-norm of $g(k)$ for all real values of $L > \frac{1}{4}$, we can indeed compute it numerically. In Fig. 11, we report the coefficient $c_0(L)$, which is two times the $\sum_{k=0}^{\infty}|g(k)|$ (Lemma 4), as a function of L. In the figure we distinguish the two areas in which the system has complex conjugate poles ($L > \frac{1}{4}$) and real-valued poles ($L \le \frac{1}{4}$). In the figure, we observe some corner points along the function $c_0(L)$. The most evident occurs at $L = 0.5$, at which we have $\theta = \frac{\pi}{4}$. This value of L belongs to those which can be found from (42), with $p = 4$ in this case. Hence, we can compute the l_1-norm of the step response $g(k)$ from (43) that is $\frac{10}{3}$,

Table 1 Example: parameters of the task set

i	C_i	T_i	$U_i = C_i/T_i$
1	15	150	0.1
2	50	400	0.125
3	60	1000	0.06

and finally $c_0(0.5) = 6 + \frac{2}{3}$, as shown in Fig. 11. The minimum, however, is taken at the "corner" corresponding to $p = 5$. By replacing $p = 5$ in (42), we can find the controller gain $L^\star$, which minimizes $c_0(L)$ and then delay Δ, that is

$$L^\star = \frac{\tan^2 \frac{\pi}{5} + 1}{4} = \frac{3 - \sqrt{5}}{2} \approx 0.38197 \tag{44}$$

and the corresponding minimal coefficient $c_0(L^\star)$

$$c_0(L^\star) = 2\frac{1}{L^\star}\frac{1 + (L^\star)^{\frac{5}{2}}}{1 - (L^\star)^{\frac{5}{2}}} = \frac{31 + 17\sqrt{5}}{11} \approx 6.2739. \tag{45}$$

In conclusion, we can assert that the controller gain which maximizes the linear lower bound of the $\mathsf{sbf}(t)$ is $L^\star = \frac{3-\sqrt{5}}{2}$. Notice that, although the linear lower bound is used to drive the design of the SAS server, the exact $\mathsf{sbf}(t)$ of (4) can be used to guarantee real-time tasks running within the SAS server. Hence, the guarantee test does not suffer from the typical approximation error of the linear lower bound.

5.1 Example of SAS server design

In this section we illustrate an example of design of SAS server. Let us assume to have a set of three tasks with execution time C_i and period T_i as indicated in Table 1. Tasks are scheduled by a fixed priority scheduler with Rate Monotonic priority assignment.

The design of a server which guarantees the schedulability of a task set and minimizes the server budget Q with a given period P has been already investigated in the literature (Lipari and Bini 2005; Dewan and Fisher 2010).

We assume that server scheduling decisions can only occur at a multiple of a system tick. The budget start and exhaustion times are computed according to the policy of SAS servers. However, due to the presence of the system tick, the actual start and completion times of the budget occur at the closest system tick. Hence, the difference between the ideal budget $Q(k)$ and the actually allocated one $S(k)$ (called disturbance in the terminology of this paper) is equal to the size of the system tick. A disturbance of the same size also applies to the sequence of idle intervals $Z(k)$. In Table 2 we show the minimum budget Q, found with the technique described in Lipari and Bini (2005) adapted to the supply function of (4), when the server period is set to $P = 60$. The budget is designed in presence of system tick equal to 0, 1, 2, and 3 (varying over rows) and controller gain $L \in \{0, 0.25, L^\star, 0.75\}$ (varying over columns). In round brackets, we also report the bandwidth increase of each case, compared to the condition with no disturbance (no system thick), that is $\varepsilon_S = \varepsilon_Z = 0$.

Table 2 Example: server budget Q (and bandwidth increase w.r.t. the case with no disturbance), when period $P = 60$

$\bar{\varepsilon}_S, \bar{\varepsilon}_Z$	$L = 0$	$L = 0.25$	$L = L^\star \approx 0.382$	$L = 0.75$
3	22.23 (15.6 %)	21.08 (9.6 %)	20.68 (7.5 %)	22.98 (19.5 %)
2	21.23 (10.4 %)	20.46 (6.4 %)	20.20 (5.0 %)	20.84 (8.4 %)
1	20.23 (5.2 %)	19.85 (3.2 %)	19.71 (2.5 %)	20.04 (4.2 %)
0	19.23 (0 %)			

In this example, it is confirmed that the controller gain L which provides the smallest server budget is $L^\star = \frac{3-\sqrt{5}}{2}$. The picture does not change significantly for different sets of tasks, as the linear lower bound of the supply function of a SAS server with $L = L^\star$ is demonstrated to be larger than in other cases.

5.2 Comments on the design of SAS servers

As demonstrated earlier, the selection of the controller gain $L = L^\star = \frac{3-\sqrt{5}}{2}$ maximizes the linear lower bound of the supply bound function $\mathsf{sbf}(t)$. Nonetheless, there may be other design guidelines, which may suggest the selection of different controller gains.

If, for example, it is desired not to have any oscillatory behavior, which appears as soon as the eigenvalues (λ_1, λ_2) have a non-zero imaginary part, then it is recommended to select $L = \frac{1}{4}$. As illustrated in Fig. 11, this gain provides the minimal $c_0(L)$ among the ones with no imaginary part. In addition, the multiplication by the gain $L = \frac{1}{4}$ can be implemented by right-shifting the binary fixed-point representation by two digits.

In addition, we remark that the optimal controller design described in Sect. 5 assumes that disturbances are allowed to be anything within the constraints (15). In reality, it may well happen that:

– disturbances are only positive, for example, in the case in which the server logic always prevents the release of the processor earlier than the expiration of the allocated budget $Q(k)$;
– disturbances may not occur at all rounds. For example, if disturbances are generated by budget overrun due to critical sections in applications, then the minimal time separation between two accesses to critical sections, may imply that disturbances cannot occur at all rounds.

In the future, we will then investigate the exploitation of extra information about the disturbances to improve the quality of the supply bound function $\mathsf{sbf}(t)$ and then, in turn, to improve schedulability conditions of the application running within the server.

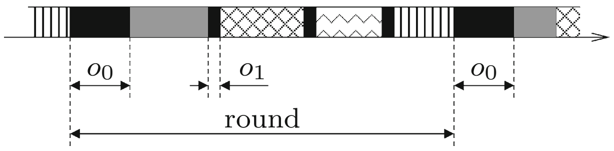

Fig. 12 Context switch overhead in SAS servers

6 Implementation

One of the main advantages of SAS servers is the simplicity of the control logic (which is just a multiplication by the controller gain). This enables a lightweight implementation. In fact, SAS servers were implemented in the Miosix kernel.[1]

Miosix is the kernel of an operating system designed to run on microcontrollers (currently supporting STM32 microcontrollers[2]), that supports the standard POSIX threads and it is distributed as free software. Among the kernel features, it exposes a separate scheduling API that allows to plug in different schedulers and select at compile-time the one to be used for the run. Currently, it implements the SAS scheduler described in the paper, together with EDF and a simple priority based scheduler. Thanks to the simplicity of the SAS algorithm, the code implementing the scheduling class is about 300 lines long, including comments and distribution license.[3] The code uses the fixed point arithmetic to compute the next budget to be assigned to the tasks (which is the operation in 9).

One of the most important claims behind the SAS server logic is that due to the simplicity of the controller, the benefits demonstrated in this paper in terms of predictability and capacity to adapt to deviations from the ideal, can be coupled with a low-overhead implementation. In fact, the control logic relies on few mathematical operations (sums and multiplications) and the time complexity of the performed calculations is constant for each thread.

The implementation introduces N SAS servers, one for each thread coexisting in the operating system. The threads are scheduled in loop and the budget computed for each of them is applied by the scheduler. Budgets are computed at the beginning of each round, therefore the scheduler operates two different types of context switches, as shown in Fig. 12. Before scheduling the first thread, the budget computation is performed, therefore the context switch time is o_0

$$o_0 = o_1 + o_{ctrl} \times N,$$

where o_1 is the actual time needed to perform the context switch operations (saving the state of the current thread and restoring the state of the thread that should resume its execution) and o_{ctrl} is the time taken to perform the amount of necessary mathematical operations (sums and multiplications) to compute the next budget to be assigned for

[1] Miosix is available at https://gitorious.org/miosix-kernel.

[2] http://www.st.com/web/en/catalog/mmc/FM141/SC1169.

[3] The specific C++ file implementing the scheduler functionality can be found at http://goo.gl/mo0KOI.

each of the N threads, in accordance to the control law of (9). All the subsequent $N-1$ context switches take o_1 to execute, since the budget has already been computed.

To experimentally evaluate the context switch overhead we performed some tests as part of a more extensive experimental campaign Maggio et al. (2014) with a `stm3210-eval` board, equipped with a 72 MHz ARM microcontroller and a 1 MB external RAM, from which the kernel code executes. To minimize the monitoring overhead, the context switch time were evaluated with an oscilloscope reading an input/output signal raised when the scheduler starts its execution and finishes it.

In a 10 min. run, where the scheduler was assigning budgets to $N = 5$ threads and executing them accordingly, the average o_0 was 205.6 μs while o_1 was 43.4 μs. The computation time of a single budget o_{ctrl} results to be 32.64 μs. Moreover, the scheduler execution overhead results to be very predictable. The difference between the average o_0 and the maximum value measured during the run is only 4 μs. More precisely, the difference between the maximum o_1 and its average value has an upper bound of 1 μs and the maximum deviation from the average value for o_{ctrl} is 3 μs.

The measured overhead is comparable with the direct context switch overhead obtained with Linux on ARM platforms (David et al. 2007). Notice also that with the SAS scheduler, the budget computation can be done only when necessary, therefore keeping the overall overhead low.

7 Conclusion and future work

Motivated by the need of compensating run-time disturbances in budget allocation, in this paper we have presented SAS servers. Together with the standard capability to respond to variations of feed-back mechanisms, we demonstrated that SAS servers can also guarantee hard real-time tasks, by computing its supply function. We showed that the supply function of SAS servers is tightly related to the l_1 norm of the response to a unitary step disturbance. This observation, has lead to the solution of the server design problem. In fact we demonstrated that by setting the controller gain $L = \frac{3-\sqrt{5}}{3}$, the supply function is maximal.

In the near future, we plan to refine the supply function computation by taking into account some constraints which may exist on the type of disturbance. For example, it may happen that the server logic requires to always allocate at least $Q(k)$ and possibly exceed this budget if the application is unable to release the processor. This case can be analyzed by constraining disturbances to be non-negative and the resulting optimal controller gain may be different. In addition, in presence of time-varying load, some adaptation may also be needed to adjust the target budget. Also this investigation may be performed in the future.

On a longer perspective, we plan to analyze the multiprocessor case, by determining the parallel supply function (Bini et al. 2009) of a set of SAS servers concurrently running over a multiprocessor platform.

Acknowledgments The authors would like to acknowledge the help by Alessandro Biondi at Scuola Superiore Sant'Anna for the fruitful discussion on existing protocols for the management of shared resources. This work was partially supported by the Swedish Research Council (VR) for the projects "Cloud Control" and "Power and temperature control for large-scale computing infrastructures", through the LCCC Linnaeus and ELLIIT Excellence Centers, and the Marie Curie Intra European Fellowship within the 7th European Community Framework Programme.

References

Abeni L, Buttazzo G (1998) Integrating multimedia applications in hard real-time systems. In: Proceedings of the 19th IEEE Real-Time Systems Symposium, pp. 4–13. Madrid, Spain

Abeni L, Palopoli L, Lipari G, Walpole J (2002) Analysis of a reservation-based feedback scheduler. In: Proceedings of the 23rd IEEE Real-Time Systems Symposium, pp. 71–80. Austix (TX), USA

Åström KJ, Hägglund T (2005) Advanced PID Control. ISA—the Instrumentation, Systems, and Automation Society

Baruah SK, Cohen NK, Plaxton G, Varvel DA (1996) Proportionate progress: a notion of fairness in resource allocation. Algorithmica 15(6):600–625

Behnam M, Nolte T, Sjödin M, Shin I (2010) Overrun methods and resource holding times for hierarchical scheduling of semi-independent real-time systems. IEEE Trans Ind Inf 6(1):93–104

Bertogna M, Fisher N, Baruah S (2009) Resource-sharing servers for open environments. IEEE Trans Ind Inf 5(3):202–219

Bini E, Bertogna M, Baruah S (2009) Virtual multiprocessor platforms: specification and use. In: Proceedings of the 30th IEEE Real-Time Systems Symposium, pp. 437–446. Washinghton, DC, USA

Cervin A, Eker J (2000) Feedback scheduling of control tasks. In: Proceedings of the 39th IEEE Conference on Decision and Control, vol. 5, pp. 4871–4876

Corbató FJ, Merwin-Daggett M, Daley RC (1962) An experimental time-sharing system. In: Proceedings of the Spring Joint Computer Conference, vol. 21, pp. 335–344

Cruz RL (1991) A calculus for network delay, part I: network elements in isolation. IEEE Trans Inf Theory 37(1):114–131

David FM, Carlyle JC, Campbell RH (2007) Context switch overheads for Linux on ARM platforms. In: Proceedings of the 2007 Workshop on Experimental Computer Science. ACM, New York, NY, USA

Desoer CA, Vidyasagar M (1975) Feedback systems: input-output properties. Academic Press, New York

Dewan F, Fisher N (2010) Approximate bandwidth allocation for fixed-priority-scheduled periodic resources. In: Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 247–256. Stockholm, Sweden

Easwaran A, Anand M, Lee I (2007) Compositional analysis framework using EDP resource models. In: Proceedings of the 28th IEEE International Real-Time Systems Symposium, pp. 129–138. Tucson, AZ, USA

Faggioli D, Checconi F, Trimarchi M, Scordino C (2009) An EDF scheduling class for the Linux kernel. In: Proceedings of the Real-Time Linux Workshop

Gresser K (1993) An event model for deadline verification of hard real-time systems. In: Proceedings of the 5th Euromicro Workshop on Real-Time Systems, pp. 118–123. Oulu, Finland

Leva A, Maggio M (2010) Feedback process scheduling with simple discrete-time control structures. Control Theory Appl IET 4(11):2331–2342

Lipari G, Bini E (2005) A methodology for designing hierarchical scheduling systems. J Embed Comput 1(2):257–269

Lu C, Abdelzaber T, Stankovic J, Son S (2001). A feedback control approach for guaranteeing relative delays in web servers. In: Proceedings of 7th IEEE Real-Time Technology and Applications Symposium, pp. 51–62

Maggio M, Terraneo F, Leva A (2014) Task scheduling: a control-theoretical viewpoint for a general and flexible solution. ACM Trans Embed Comput Syst 13(4):76:1–76:22

Mercer CW, Savage S, Tokuda H (1993) Processor capacity reserves: an abstraction for managing processor usage. In: Proceedings of the Fourth Workshop on Workstation Operating Systems, pp. 129–134

Mok AK, Feng X, Chen D (2001) Resource partition for real-time systems. In: Proceedings of the 7th IEEE Real-Time Technology and Applications Symposium, pp. 75–84. Taipei, Taiwan

Richter K, Ernst R (2002) Event model interfaces for heterogeneous system analysis. In: Design, Automation and Test in Europe (DATE), pp. 506–513. Paris, France

Stankovic JA, Lu C, Son SH, Tao G (1999) The case for feedback control in real-time scheduling. In: Proceedings of the 11th Euromicro Conference on Real-Time, pp. 11–20. York, UK

Stigge M, Ekberg P, Guan N, Yi W (2011) On the tractability of digraph-based task models. In: Proceedings of the 23rd Euromicro Conference on Real-Time Systems, pp. 162–171. Porto, Portugal

Storm AJ, Garcia-Arellano C, Lightstone SS, Diao Y, Surendra M (2006) Adaptive self-tuning memory in db2. In: Proceedings of the 32rd International Conference on Very Large Data Bases, VLDB '06, pp. 1081–1092. VLDB Endowment

Thiele L, Chakraborty S, Naedele M (2000) Real-time calculus for scheduling hard real-time systems. In: Proceedings of the IEEE International Symposium on Circuits and Systems, vol. 4, pp. 101–104. Geneva, Switzerland

Alessandro Vittorio Papadopoulos is a postdoctoral researcher at the Department of Automatic Control, Lund University, Sweden. He is also a member of the Lund Center for Control of Complex Engineering Systems (LCCC) Linnaeus Center. He received a BSc and a MSc in Computer Engineering from Politecnico di Milano, Italy. In 2013, he received a PhD in Information Technology, Systems and Control with honors from the Politecnico di Milano, Italy, and he was awarded with the European doctorate certificate. His research interests include model reduction for hybrid systems, event-based control, and the application of control theory for the design and the implementation of computing systems, with a particular focus on cloud computing, and real-time systems.

Martina Maggio is an Assistant Professor at the Department of Automatic Control, Lund University, where she started as a postdoctoral researcher in 2012. She was awarded her PhD from the Dipartimento di Elettronica ed Informazione at Politecnico di Milano under the supervision of Alberto Leva, and she was a visiting PhD student at the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, supervised by Anant Agarwal. Her main research interest is the application of control theory to computer engineering, especially to cloud computing and to the design of operating systems and resource management framework.

Alberto Leva received the Laurea degree in Electronic Engineering in 1989 at the Politecnico di Milano, where he is at present Associate Professor of Automatic Control. His main research interest concern methods and tools for the automatic tuning of industrial controllers, process modeling, simulation and control, particularly within the object-oriented paradigm, energy systems, and control education. In the last years, he has been concentrating on control-based design of computing systems, addressing in a system- and control-theoretical manner problems like scheduling, resource allocation, time synchronization, thermal and power/performance management, and service composition. Alberto Leva is author or co-author of around 200 publications in peer-reviewed international journals and conferences.

Enrico Bini is Assistant Professor at Scuola Superiore Sant'Anna. In 2012–13, he was Marie-Curie fellow at Lund University (Sweden) investigating on Cyber-Physical Systems. In 2000 he received the Laurea degree in Computer Engineering from University of Pisa. In 2004, he completed the doctoral studies on Real-Time Systems at Scuola Superiore Sant'Anna. During the PhD, he visited the University of North Carolina at Chapel Hill (under the supervision of Sanjoy Baruah). In January 2010 he also completed a Master degree in Mathematics with a thesis on optimal sampling for linear control systems. He has published more than 80 papers (two best-paper awards) on real-time scheduling, and design and optimization methods for real-time and control systems. His recent research interests are on optimal management of distributed and parallel resources.