Resource Adaptations with Servers for Hard Real-Time Systems*

Nikolay Stoimenov and Lothar Thiele
Computer Engineering and Networks Laboratory
ETH Zurich
8092 Zurich, Switzerland
{stoimenov,thiele}@tik.ee.ethz.ch

Luca Santinelli and Giorgio Buttazzo
RetisLab
SSSA Pisa
56124 Pisa, Italy
{l.santinelli,giorgio}@sssup.it

ABSTRACT

Many real-time applications are designed to work in different operating modes each characterized by different functionality and resource demands. With each mode change, resource demands of applications change, and static resource reservations may not be feasible anymore. Dynamic environments where applications may be added and removed online also need to adapt their resource reservations. In such scenarios, resource reconfigurations are needed for changing the resource reservations during runtime and achieve better resource allocations. There are a lot of results in the scientific literature of how to find the optimal amount of resources needed by an application in the different operating modes, or how an application can perform safe mode transitions. However, the problem of resource reconfigurations for systems with reservations has not been addressed. A resource scheduler should be reconfigured online in such a way that it still guarantees a certain amount of resources during the reconfiguration process, otherwise applications may miss deadlines. The paper proposes a framework for scheduling real-time applications through scheduling servers that provide resource reservations, and algorithms for changing the resource reservations online while still guaranteeing the feasibility of the system and the schedulability of applications. The framework analysis is integrated into a well-known modular performance analysis paradigm based on Real-Time Calculus. The results are illustrated with examples and a case study.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-Time and Embedded Systems

General Terms

Algorithms, Performance, Verification

1. INTRODUCTION

The server architecture paradigm has been seriously considered in the past years for its ability to separate the scheduling concerns between the system and the application levels. A server mechanism is strictly connected with the resource partition idea where a shared resource, e.g. CPU computation time, is used by several applications. Servers are used to isolate the temporal behavior of real-time tasks through resource reservations [17]. Abeni and Buttazzo [3] introduced a bandwidth reservation mechanism (the Constant Bandwidth Server - CBS) that allows real-time tasks to execute in dynamic environments under a temporal protection mechanism, so that a server never exceeds a predefined bandwidth regardless of the actual requests of the server tasks.

Server models can be classified into *event-driven servers*: the servers are driven by the application requirements. The CBS and sporadic server [22] are typical examples. And *time-triggered servers*: the server resource supply is driven by a predefined timing pattern that depends only on the server properties. An example is the Time Division Multiple Access (TDMA) server where the resource is periodically partitioned [31]. In particular, a TDMA server assigns time slots to its applications that repeat each cycle.

Nowadays, dynamic real-time applications ask for real-time systems that can adapt their behavior at run-time by changing their operating mode: the computing environment and the available resource of a system may change over time. For example, adding a new task into the system at runtime may result in a reduction of the computing resources being allocated to the existing tasks. Moreover a change in the operating mode of an application, e.g., from start-up to normal, or from normal to shut-down, may also demand reallocation of the computing resources among the tasks. That and many other scenarios require flexible workload management and resource allocation.

Whereas a server manages an application by supplying the resource it requires, adaptive applications must rely on adaptive servers to meet their changing resource requirements. Servers need to be reconfigured dynamically to adapt the resource reservations and reflect the changes in the system or its environment. Such reconfigurations need to be performed online without jeopardizing schedulability. It is therefore essential to develop appropriate resource reconfiguration criteria and algorithms to manage the criticality of the transition phase.

To cope with applications in which the computational demand is highly variable, fixed reservations could not be appropriate to achieve the desired performance, hence adaptive scheduling schemes need to be adopted. Buttazzo et al. [6] proposed an elastic scheduling methodology for adapting the rates of a periodic task set to different workload scenarios, without affecting the system schedulability. Abeni et al. [1] presented a framework for dynamically allocating the CPU resource to tasks whose execution times are not known a priori. Adaptive reservation techniques based on feedback scheduling have also been investigated by the authors in [2]. All of these frameworks are only suitable for soft real-time systems.

There are also systems in which the application is characterized by multiple execution modes, each consisting of a specific task set and workload requirement. For these systems, the feasibility of the schedule has to be guaranteed

*The work is partially supported by NCCR-MICS, a center supported by the Swiss National Science Foundation under grant number 5005-67322.

EMSOFT'10, October 24–29, 2010, Scottsdale, Arizona, USA.

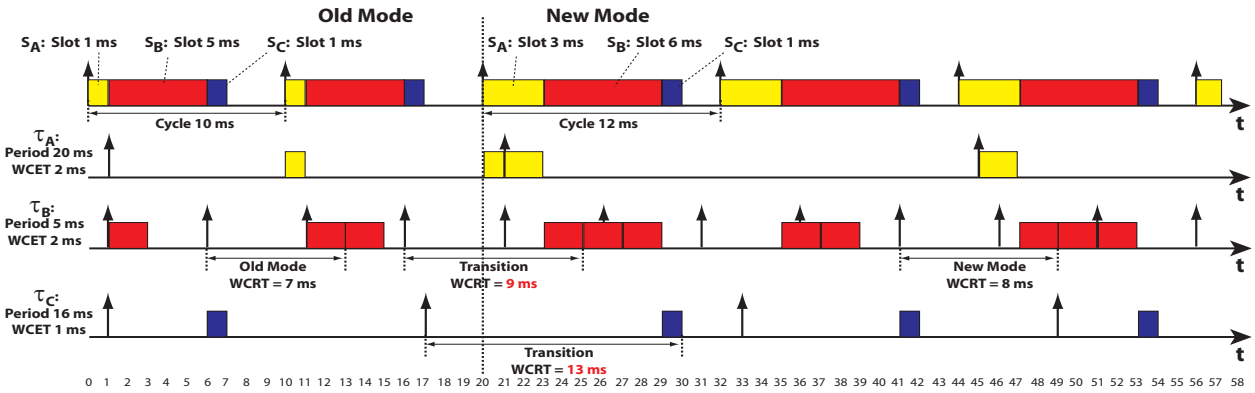

Figure 1: TDMA servers reconfigured at $t = 20$ms (dashed line) causes longer WCRTs for tasks τ_B and τ_C.

not only within each individual mode, but also during mode transitions. This problem has been deeply investigated in the real-time literature [18, 21, 24, 28]. Crespo et al. [19] presented a survey of mode change protocols for uniprocessor systems under fixed-priority scheduling and proposed a new protocol along with its schedulability analysis. Guangming [12] computed the earliest time at which a new task can be safely added to the system scheduled by the Earliest Deadline First (EDF), without jeopardizing the feasibility of the task set. All of these results address the problem of performing mode transitions in applications without violating their schedulability. None of them considers how to *change resource reservations* online without violating applications schedulability which is the goal of this paper.

In real-time operating systems, servers are a specific scheduling mechanism that handles aperiodic requests as soon as possible while preserving hard periodic tasks from missing their deadlines. Another classification distinguishes between fixed priority and dynamic priority servers, depending on the scheduling policy used to schedule them. Among fixed priority servers, deferrable server [26] and sporadic server [22] are the most well-known techniques that preserve their capacity when no request is pending upon the invocation of a server. Spuri et al. [23] presented a survey of dynamic priority servers that can efficiently work under EDF. It is also notable that time-triggered architectures play an increasingly important role in large distributed embedded systems as described in [14, 31]. Mainly, time-triggered servers offer high predictability with enormous benefits to the analysis of real-time systems.

However, classical server paradigms and models do not allow adaptations to changing conditions. To the best of our knowledge, none of the schedulers that provide isolation and real-time guarantees have mechanisms for online reconfiguration that can provide guarantees during the reconfiguration process. It may be possible to wait for an idle time in the system in order to reconfigure the scheduler as in [9], however, it is highly unlikely that idle times occur at the same time for all applications.

Several papers have tried to face and cope with this lack. Fohler [11] investigated the problem of mode changes in both the applications and the scheduler in the context of pre run-time scheduled hard real-time systems. Applications are specified with periodically activated graphs with precedence constraints for which safe switching points are pre-computed using heuristic search techniques. The FRESCOR project [13] has proposed a mode change protocol for a system with virtual resources based on the sporadic server and periodic tasks where budgets may change. Both frameworks are not as general as the results presented in this paper which can deal with hierarchically scheduled systems with mixed schedulers and complex task activation schemes.

New mechanisms have been proposed to change server models at run-time. de Olivera et al. [10] addressed the problem of finding optimal CBS parameters and dynamically reconfiguring the servers, offering support for multi-mode adaptive real-time applications. Valls et al. [29] presented an adaptation protocol based on the definition of a contract model for filtering peaks in resource demands where applications are modeled with periodic, continuous, and imprecise tasks. However, in both frameworks there are no algorithms and analysis of applications schedulability for the proposed online resource reconfigurations.

Brandt et al. [5] propose the rate-based earliest deadline (RBED) scheduler where servers are periodic tasks scheduled with EDF. The paper gives system schedulability conditions for adaptations in the periods and utilizations of the servers. However, the paper does not go neither into characterizing the service provided by the servers during reconfigurations, nor into algorithms for how to control this service.

Craciunas et al. [7] propose the variable-bandwidth server (VBS) which is based on CBS but allows for adaptations. Applications are specified as sequences of actions which execute on a VBS. Activations of actions change the parameters of the VBS, and schedulability is based on the maximum utilization from all actions of an application. Our framework is more general as server reconfigurations may happen at any time, are independent of application model, and can take advantage of application operating modes.

Contributions: We tackle the problem of scheduler adaptations in resource partitioned architectures, mainly from the perspective of adaptive servers that provide real-time guarantees. We develop a scheduling server framework based on the static TDMA partitioning paradigm. We establish criteria that need to be met during a reconfiguration of the framework, classify the possible reconfiguration scenarios, introduce algorithms for performing them while meeting real-time constraints of applications, and present schedulability analysis for the reconfigurations based on Real-Time Calculus [27].

Organization of the paper: Section 2 classifies the problems that may occur during reconfigurations of some common servers with examples. Section 3 describes the proposed Adaptive Server with Guarantees, and defines the service guarantees that it provides during operation and reconfiguration. Section 4 classifies the possible reconfiguration scenarios and analyzes schedulability for each of them. Section 5 illustrates the algorithms with a case study. Finally, Section 6 concludes this paper.

2. MOTIVATIONAL EXAMPLES

To illustrate the different problems that may occur during reconfigurations, we have chosen three examples of systems with TDMA servers [31], static polling servers [20], and CBSs [3]. Similar examples can be derived with other kinds of servers and show that a naive online change of parameters is not able to guarantee the system schedulability in hard real-time scenarios.

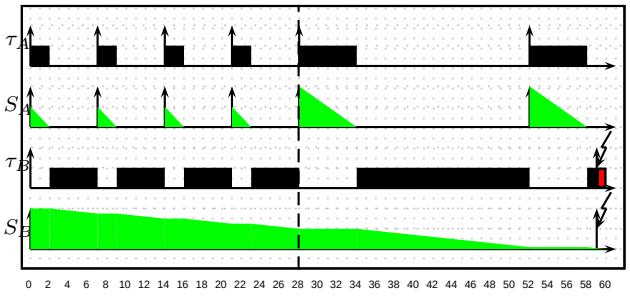

Figure 2: Polling server S_A reconfigured at $t = 28$ms (dashed line) causes a deadline miss for task τ_B and a capacity miss for server S_B.

EXAMPLE 1. *Consider Figure 1. Three TDMA servers, S_A, S_B, and S_C can operate in two modes, denoted as Old Mode and New Mode. We suppose that given an operating mode, all TDMA servers operate with the same period which equals the cycle of the TDMA. When there is a mode change, the allocated slots in the TDMA and the cycle of the TDMA may change. When a server slot becomes available, it is available regardless of whether there is workload to use it.*

Server S_A serves a single task τ_A with worst-case execution time (WCET) of 2ms and period of 20ms which we will denote as $(2, 20)$. In Old Mode, server S_A has a reserved slot of 1ms in a TDMA cycle of 10ms denoted as $(1, 10)$. In New Mode, server S_A has parameters $(3, 12)$. Server S_B serves a single task τ_B with parameters $(2, 5)$. The server in Old Mode has parameters $(5, 10)$ and in New Mode $(6, 12)$. Server S_C serves a single task τ_C with parameters $(1, 16)$. The server in Old Mode has parameters $(1, 10)$ and in New Mode $(1, 12)$.

Figure 1 shows a server reconfiguration performed at time $t = 20ms$. For task τ_B this means that it suffers longer worst-case response time (WCRT) of 9ms during the reconfiguration whereas its WCRT is 7ms in Old Mode and 8ms in New Mode. Similarly task τ_C has a longer WCRT during the reconfiguration equal to 13ms, whereas in Old Mode it is 10ms and in New Mode 12ms.

Hence, a reconfiguration of TDMA servers may cause several tasks to miss deadlines.

EXAMPLE 2. *Consider Figure 2. Two polling servers, S_A and S_B, are scheduled with the fixed priority policy. Server S_A has higher priority. It can operate in two modes. In Old Mode it has a budget of 2ms and a period of 7ms, denoted as $(2, 7)$. It serves a single task τ_A with WCET of 2ms and deadline equal to period of 7ms, denoted as $(2, 7)$. In New Mode S_A and τ_A have parameters $(6, 24)$ and $(6, 24)$, respectively. Server S_B and its task τ_B operate in a single mode and their parameters are $(40, 59)$ and $(40, 59)$, respectively. The system is schedulable separately in both modes.*

Figure 2 shows a server reconfiguration without a proper transition algorithm. Server S_A and task τ_A simultaneously enter Mode II at time $t = 28ms$ which leads to a capacity miss for server S_B and a deadline miss for task τ_B at time $t = 59ms$ even though the mode change was performed at the end of the periods for server S_A and task τ_A.

The example illustrates that reconfiguration of a server may cause other servers to not be able to deliver their guaranteed budgets.

EXAMPLE 3. *Consider Figure 3. CBS S_A can operate in two modes. In Old Mode it has a budget of 4ms with a period of 5ms denoted as $(4, 5)$. It serves a single task τ_A with WCET of 8ms and deadline equal to period of 10ms denoted as $(8, 10)$. In New Mode, the parameters for S_A are $(8, 10)$ and τ_A is unchanged. CBS S_B serves a single task τ_B with parameters $(2, 10)$ and $(2, 10)$, respectively. The system is schedulable when server S_A is either in Old Mode or in New Mode as $U = U_{S_A} + U_{S_B} = 1$.*

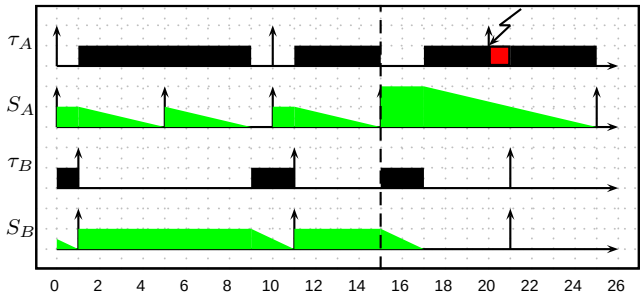

Figure 3: CBS S_A reconfigured at $t = 15$ms (dashed line) causes a missed deadline for task τ_A.

Figure 3 shows a reconfiguration for server S_A at the end of a server deadline at time $t = 15ms$ which leads to a missed deadline for task τ_A at time $t = 20ms$.

The example illustrates that reconfiguration of a server may cause the application that it serves to miss deadlines.

In summary, the problems observed during online reconfiguration of servers fall in two classes:

1. **Isolation violation:** a reconfiguration of one server may cause other servers to not be able to deliver their guaranteed capacities.
2. **Deadline violation:** a reconfiguration of a server may affect the application that it serves by making it miss deadlines.

Safe reconfiguration algorithms will have to address both problems in order to be suitable for hard real-time systems.

3. FRAMEWORK FOR ADAPTIVE SERVERS WITH GUARANTEES

In this section, we give an overview of a framework with adaptive resource reservations. There are many scenarios for the use of such a framework and many different ways to realize it. We focus on the scheduling servers and their properties. In our framework, applications share a common processor using servers and we refer to them as Adaptive Servers with Guarantees (ASG) as they guarantee resource reservations and can be reconfigured dynamically while still providing a guarantee even during the reconfiguration.

We consider a uniprocessor system that runs a set of applications. Each application is scheduled on an individual ASG. Servers provide resource reservations and guarantee isolation between applications. Applications can be of arbitrary complexity and they may even have their own schedulers, as in hierarchically scheduled systems [30]. An ASG is only concerned with guaranteeing a minimum service supply to its application. The system has a single Server Manager that can control the parameters of all servers (such as their budgets and period) and is able to communicate with the applications in order to accommodate their changing resource requirements. The overall system framework is illustrated in Figure 4.

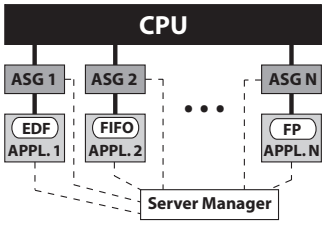

Figure 4: Overview of a system where the CPU is shared by applications through multiple ASGs.

3.1 The Adaptive Server with Guarantees

Servers are scheduled statically by a TDMA scheme. For each server a slot of fixed size Q called budget is reserved in the TDMA time-wheel. A server is activated, i.e., its budget becomes available, when the slot of the server arrives in the TDMA time-wheel. All servers in the system are activated periodically with the same period P which equals to the cycle of the TDMA. Servers can have different budgets but always a common period. An ASG is denoted with the tuple (Q, P). A schedule of four ASGs is illustrated in Figure 5.

Budgets are always given to applications regardless of whether they use them or not, like in a traditional TDMA schedule. In the following discussion, we assume that context switch overheads take negligible time but they can be trivially added to our analysis. The description of an ASG can be summarized in the following definition.

Definition 1. An ASG (Q, P) guarantees to an application access to a shared resource for $Q > 0$ time units every $P > 0$ time units, where $Q \leq P$.

The total utilization for a system with N ASGs is defined as the sum of the single server utilizations, i.e., $U = \sum_{i=1}^{N} Q_i/P$. Such a system is schedulable when the total utilization is smaller or equal to 1:

$$U \leq 1. \tag{1}$$

When the total utilization is less than 1, there is some unused budget in the system, Q_F, called the free budget. We suppose that all ASGs are scheduled from the beginning of every period one after the other, and the free budget is always at the end, as illustrated in Figure 5. The free budget may be given to non real-time applications on the basis that it can always be reclaimed by the system. The free budget is essential in our framework during reconfigurations as it will be shown in Section 4.

3.2 Resource Supply of an ASG

An ASG (Q, P) may not have access to the CPU for a time interval Δ that is upper bounded by $P - Q$. After this interval, the server will have guaranteed access to the resource for Q time units. Therefore, an ASG cannot guarantee resource access for any interval of size $0 \leq \Delta \leq P - Q$. However, it guarantees service of $S(\Delta - (P - Q))$, in any interval $(P - Q) \leq \Delta \leq P$, where S is CPU speed, e.g. cycles per time unit. Without loss of generality, we assume that $S = 1$, as all parameters in the system can be normalized according to this speed. Then the minimum resource supply of an ASG (Q, P) in any time interval Δ can be lower bounded by the following function:

$$\beta_{Q,P}(\Delta) = \max\left(\left\lfloor \frac{\Delta}{P} \right\rfloor Q, \Delta - \left\lceil \frac{\Delta}{P} \right\rceil (P - Q)\right),$$

or more compactly as:

$$\beta_{Q,P}(\Delta) = \sup_{0 \leq \lambda \leq \Delta} \left\{ \lambda - \left\lceil \frac{\lambda}{P} \right\rceil (P - Q) \right\}. \tag{2}$$

The minimum resource supply for an ASG (Q, P) is illustrated in Figure 6.

The minimum resource supply function in (2) is actually a lower *service curve* as known from Network and Real-Time Calculus [8, 15, 27]. Service curves are abstract representations for the availability of processing and communication resources. A service curve $\beta(\Delta)$ gives a lower bound on the

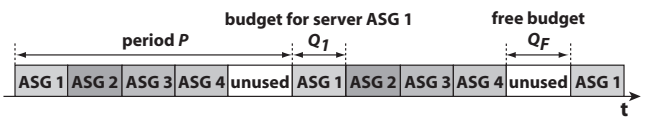

Figure 5: Schedule for four ASGs.

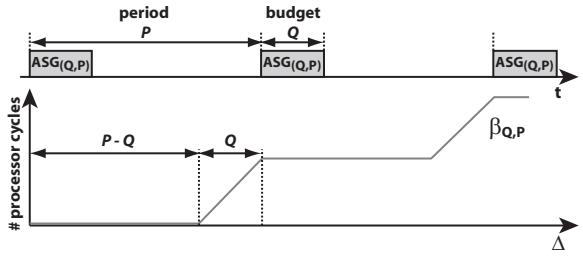

Figure 6: Resource supply of an ASG (Q, P).

available service in *any* time interval of length $\Delta > 0$ where for $\Delta \leq 0$, $\beta(\Delta) = 0$. The service is usually expressed in a suitable workload unit such as number of cycles for computing resources or bits for communication resources.

3.3 Performance Analysis

Application tasks are activated by the arrivals of events. The timing characteristics of event arrivals are described abstractly with *arrival curves* as known from Network and Real-Time Calculus. The arrival curve $\alpha(\Delta)$ denotes an upper bound on the number of events that arrive in *any* time interval of length $\Delta > 0$ where for $\Delta \leq 0$, $\alpha(\Delta) = 0$. Arrival curves substantially generalize traditional event stream models such as periodic, periodic with jitter, and sporadic. Often the domain of arrival curves are workload units. Event-based arrival curves can be converted to workload-based arrival curves by scaling with the best-case/worst-case execution demands of events. The units of the arrival and service curves used in an analysis need to be the same. In this paper, we will use the workload-based interpretation and assume that each event has a fixed execution demand. More general concepts for characterization of these units are discussed in [16].

Now given the minimum resource supply of an ASG and a characterization of the activation stream of the task, we can compute the worst-case response time (WCRT) for the task. To this end, we use results from Network and Real-Time Calculus where for a resource supply characterized with a service curve β and an input stream characterized with an arrival curve α, the WCRT of an event from the stream is the maximum horizontal distance between the arrival and the service curves computed as follows:

$$\sup_{\lambda \geq 0} \{\inf\{\tau \geq 0 : \alpha(\lambda) \leq \beta(\lambda + \tau)\}\} \triangleq \mathrm{Del}(\alpha, \beta). \tag{3}$$

Example 4. *To illustrate this, let us consider Example 1 from Section 2. Consider server S_B in Old Mode which is an ASG with budget $Q = 5ms$ and a period $P = 10ms$. The respective service curve can be computed with equation (2). It serves a single periodic task τ_B with a period of $5ms$ and WCET of $2ms$. The WCRT of the task computed with equation (3) is shown in Figure 7a. The computed WCRT is equal to the one observed on the trace in Figure 1.*

3.4 Schedulability of Applications

An application is schedulable if its real-time requirements are satisfied by the system. If we consider the case of a single task, we may have the requirement that all activations are processed within a relative deadline D. Given (3), this is expressed as $\mathrm{Del}(\alpha, \beta) \leq D$. Inverting it w.r.t. β, we can compute a lower bound on the minimum resource demand required to meet the deadline requirement. This is expressed as follows:

$$\underline{\beta}(\Delta) \geq \alpha(\Delta - D) \quad \forall \Delta \in \mathbb{R}^{\geq 0}. \tag{4}$$

In other words, the minimum resource demand has a lower service curve that equals to $\underline{\beta}(\Delta) = \alpha(\Delta - D)$.

By using previous results on demand bound functions by Baruah et al. [4] and interface-based design by Wandeler

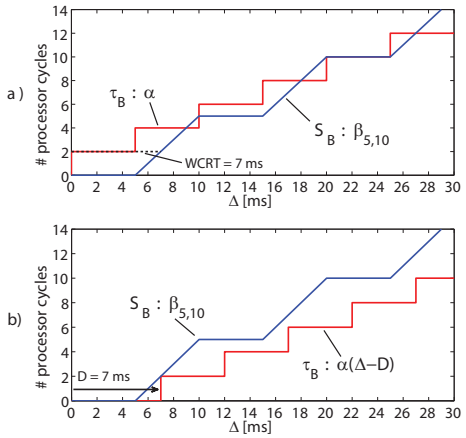

Figure 7: Server S_B and task τ_B WCRT analysis (a) and schedulability condition (b).

and Thiele [30] such a task is schedulable if a resource can supply service that is larger or equal to the demanded one. For an ASG (Q, P), schedulability would mean that:

$$\beta_{Q,P}(\Delta) \geq \underline{\beta}(\Delta) \quad \forall \Delta \in \mathbb{R}^{\geq 0} \tag{5}$$

where $\beta_{Q,P}$ is computed with (2).

In the case of task τ_B from Example 1, it is schedulable with a relative deadline $D = 7$ms by server S_B with Old Mode parameters $(5, 10)$. This can be seen in Figure 7b where the service curve of server S_B is above the shifted arrival curve of task τ_B which expresses the resource demand of the task.

The same schedulability condition applies not only for single tasks, but even for complex applications as we can compute the minimum resource demand of an application as a single service curve $\underline{\beta}$, for details see [30].

3.5 Schedulability during a Reconfiguration

A reconfiguration may change the server parameters such as their budgets and period from one mode to another. We consider a single reconfiguration. For a system with N ASGs before a reconfiguration they operate with parameters (Q_i^O, P^O), $1 \leq i \leq N$, (for Old Mode), and after the reconfiguration with parameters (Q_i^N, P^N), $1 \leq i \leq N$, (for New Mode). We assume that the system is schedulable in Old Mode and New Mode separately, i.e., condition (5) is satisfied by assumption for all servers in Old Mode:

$$\beta_{Q_i^O,P^O}(\Delta) \geq \underline{\beta_i}(\Delta) \quad \forall \Delta \in \mathbb{R}^{\geq 0} \quad \forall i,$$

and for all servers in New Mode:

$$\beta_{Q_i^N,P^N}(\Delta) \geq \underline{\beta_i}(\Delta) \quad \forall \Delta \in \mathbb{R}^{\geq 0} \quad \forall i.$$

During a reconfiguration or the changing from one set of server parameters to another, the system should not suffer

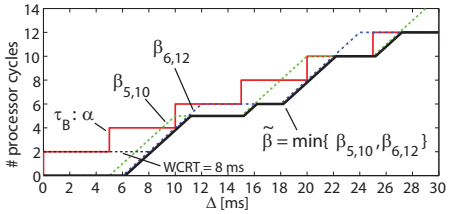

Figure 8: Condition (6) can guarantee a WCRT of 8ms for task τ_B during the reconfiguration in Example 1.

Table 1: Reconfiguration Scenarios

$P^O = P^N$	Remove a server
	Decrease of a budget
	Add a server
	Increase of a budget
$P^O \neq P^N$	Increase of period $P^O < P^N$
	Decrease of period $P^O > P^N$

a degraded performance. Let us consider the two problems described in Section 2. To prevent isolation violations, each server should be able to guarantee a service curve during a reconfiguration. To prevent deadline violations each server should be able to guarantee a service curve that is sufficiently large during a reconfiguration.

Let us denote as $\tilde{\beta}_i(\Delta)$ the service provided by an ASG during time intervals Δ that span Old Mode, the Reconfiguration, and New Mode. In order to prevent a degraded performance during a reconfiguration we need to have for all servers that:

$$\tilde{\beta}_i(\Delta) \geq \min\{\beta_{Q_i^O,P^O}(\Delta), \beta_{Q_i^N,P^N}(\Delta)\} \; \forall \Delta \in \mathbb{R}^{\geq 0} \; \forall i. \tag{6}$$

The above condition ensures that each server guarantees during a reconfiguration at least the minimum of the services guaranteed in Old and New Modes. This implies that each application served by an ASG during a reconfiguration is guaranteed that it will not violate the larger of the deadlines from Old and New Modes.

EXAMPLE 5. *To illustrate this, consider server S_B from Example 1. During the transition from Old Mode to New Mode, if it were able to meet condition (6), then the WCRT of task τ_B would have been at most the maximum of the WCRTs from the two modes which is 8ms, and it would not have experienced the WCRT of 9ms. This is illustrated in Figure 8.*

4. ALGORITHMS AND ANALYSIS

In this section, we classify the scenarios for feasible resource reconfigurations and provide schedulability analysis for each of them to show that they meet condition (6) with proofs available in [25]. The proposed algorithms are implemented in the Server Manager and executed by it. Initiation of a reconfiguration can be done by an application in order to request a different resource reservation, or by the Server Manager in order to achieve better resource allocation. The proposed algorithms work regardless of what the reason for reconfiguration is.

We distinguish between reconfigurations that change the period of the servers, i.e., $P^O \neq P^N$, and those that do not, i.e., $P^O = P^N$. The possible reconfiguration scenarios are summarized in Table 1.

Reconfigurations that do not require change of period have simple feasibility conditions and they do not require any precomputed information except budgets and period, and the decision for performing them can be made entirely online. For the case of changing periods, conditions are much more involved as we will see, and some parameters need to be pre-computed and stored in the Server Manager to be used online.

Notation. The time of the k-th activation of server (Q_i, P) is denoted as $s_{i,k}$. The time when the free budget starts is $s_{F,k}$. An activation frame k contains the k-th activations of all servers and the free budget. The time when activation frame k starts is the activation time of the first scheduled server (Q_1, P) denoted as $s_{1,k}$ and it ends when the same server is scheduled again $s_{1,k+1}$. When we would like to differentiate between any of the parameters and indicate that they belong to the Old Mode or the New Mode, we will add the superscripts O or N, respectively. In the

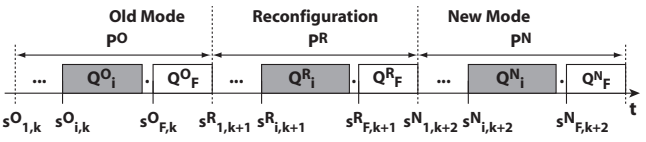

Figure 9: Notation. Three activation frames where activation frame k belongs to the Old Mode, frame $k+1$ to the Reconfiguration, and frame $k+2$ to the New Mode.

Old Mode, all activation frames have the same length which equals to the period, $P^O = s^O_{1,k+1} - s^O_{1,k}$ for frames k in the Old Mode, unless otherwise stated. Similarly for the New Mode.

Algorithms that change the period of servers will require an intermediate phase called Reconfiguration where budgets and period will be different than the ones in Old and New Modes. Parameters belonging to the Reconfiguration will carry the superscript R when necessary. The notation is illustrated in Figure 9.

4.1 No Change of Period

Here for brevity we do not differentiate between P^O and P^N but refer to the period as P. In these scenarios the last activation frame of the Old Mode which we denote as k is followed immediately by the first activation frame of the New Mode denoted as $k+1$.

4.1.1 Removing an Existing ASG

Removing a server from the schedule means that in the Old Mode, it has budget $Q^O > 0$, and in the New Mode, its budget is $Q^N = 0$. The budgets of all other servers are unchanged. This is an operation that can always be performed since it decreases the utilization of the system by Q^O/P, and increases the free budget, $Q^N_F = Q^O_F + Q^O$.

Algorithm 1 describes removing server (Q^O_i, P) from a schedule with N servers. When the server is removed, activations of all preceding servers are unchanged while activations of succeeding servers are shifted earlier by the removed budget. This is illustrated in Figure 10.

Algorithm 1 `Removing an ASG`

Input: $s^O_{j,k},\ 1 \le j \le N$ ▷ Schedule in last frame (k) of Old Mode
Input: P ▷ Current period
Input: (Q^O_i, P) ▷ Server to be removed
Output: $s^N_{j,k+1},\ 1 \le j \le N, j \ne i$ ▷ Schedule in first frame ($k+1$) of New Mode

1: **for** $j \leftarrow 1$ to N **do**
2: **if** $j < i$ **then**
3: $s^N_{j,k+1} \leftarrow s^O_{j,k} + P$
4: **else if** $j > i$ **then**
5: $s^N_{j,k+1} \leftarrow s^O_{j,k} + P - Q^O_i$
6: **end if**
7: **end for**

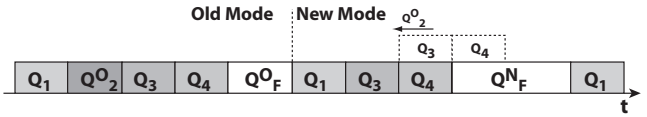

Figure 10: Removing server (Q^O_2, P) from a schedule of four ASGs. The activation times of servers (Q^N_3, P) and (Q^N_4, P) have been shifted to the left by Q^O_2 in New Mode, and Q^O_2 has been used to increase the free budget. The dashed boxes show where servers (Q^O_3, P) and (Q^O_4, P) would have been scheduled if there were no reconfiguration.

THEOREM 1. *Removing server (Q^O_i, P) from a schedule of N servers using Algorithm 1 satisfies condition (6) for all other servers in the system as each of them gets at least a guaranteed service during the reconfiguration of $\tilde{\beta}_j \ge \beta_{Q_j,P}$, $1 \le j \le N$, $j \ne i$.*

PROOF. All proofs are omitted and can be found online in a technical report [25]. □

4.1.2 Decreasing the Budget of an Existing ASG

Decreasing the budget of a server means that in Old Mode, the server has budget $Q^O > 0$, and in New Mode, its budget is $0 < Q^N < Q^O$. The budgets of all other servers are unchanged. This is an operation that can always be performed since it decreases the utilization of the system by $(Q^O - Q^N)/P$, and increases the free budget, $Q^N_F = Q^O_F + (Q^O - Q^N)$.

Algorithm 2 describes decreasing the budget of server (Q_i, P) from Q^O_i in Old Mode to Q^N_i in New Mode in a schedule of N servers. In the first frame when the budget is decreased, activations of all preceding servers are unchanged while activations of succeeding servers are shifted earlier by the amount of decrease of budget. This is illustrated in Figure 11.

THEOREM 2. *Decreasing the budget of a server from (Q^O_i, P) to (Q^N_i, P) in a schedule of N servers using Algorithm 2 satisfies condition (6) for all servers in the system. Unchanged servers get at least a guaranteed service during the reconfiguration of $\tilde{\beta}_j \ge \beta_{Q_j,P}$, $1 \le j \le N$, $j \ne i$. For the decreased server, this is $\tilde{\beta}_i \ge \beta_{Q^N_i,P}$.*

4.1.3 Adding a New ASG

Adding a server to the schedule means that in Old Mode, it has budget $Q^O = 0$, while in New Mode, its budget is $Q^N > 0$. Budgets of all other servers are unchanged. From condition (1), this is an operation that is feasible if there is sufficient free budget in the system:

$$Q^N \le Q^O_F.$$

Algorithm 2 `Decreasing the budget of an ASG`

Input: $s^O_{j,k},\ 1 \le j \le N$ ▷ Schedule in last frame (k) of Old Mode
Input: P ▷ Current period
Input: (Q^O_i, P) ▷ Server to be modified with Old Mode parameters
Input: (Q^N_i, P) ▷ Server to be modified with New Mode parameters
Output: $s^N_{j,k+1},\ 1 \le j \le N$ ▷ Schedule in first frame ($k+1$) of New Mode

1: **for** $j \leftarrow 1$ to N **do**
2: **if** $j \le i$ **then**
3: $s^N_{j,k+1} \leftarrow s^O_{j,k} + P$
4: **else if** $j > i$ **then**
5: $s^N_{j,k+1} \leftarrow s^O_{j,k} + P - (Q^O_i - Q^N_i)$
6: **end if**
7: **end for**

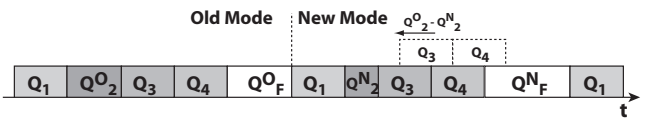

Figure 11: Decreasing the budget from Q^O_2 to Q^N_2 in a schedule of four ASGs. The activation times of servers (Q_3, P) and (Q_4, P) have been shifted earlier in New Mode by $(Q^O_2 - Q^N_2)$, and $(Q^O_2 - Q^N_2)$ has been used to increase the free budget. The dashed boxes show where servers (Q_3, P) and (Q_4, P) would have been scheduled if there were no reconfiguration.

Algorithm 3 `Adding an ASG`

Input: $s^O_{j,k},\ 1 \le j \le N$ ▷ Schedule in last frame (k) of Old Mode
Input: $s^O_{F,k}$ ▷ Start of free budget in frame (k) in Old Mode
Input: P ▷ Current period
Input: Q^O_F ▷ Free budget in Old Mode
Input: (Q^N_{N+1}, P) ▷ Server to be added in New Mode
Require: $Q^N_{N+1} \le Q^O_F$
Output: $s^N_{j,k+1},\ 1 \le j \le N+1$ ▷ Schedule in first frame ($k+1$) of New Mode

1: **for** $j \leftarrow 1$ to N **do**
2: $\quad s^N_{j,k+1} \leftarrow s^O_{j,k} + P$
3: **end for**
4: $s^N_{N+1,k+1} \leftarrow s^O_{F,k} + P$

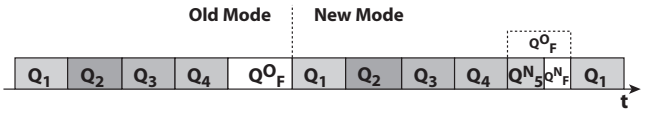

Figure 12: Addition of server (Q_5, P) to a schedule of four ASGs. Activation times of existing servers do not change as the added server is scheduled after all other servers in New Mode. Free budget has been decreased by the budget of the server, $Q^N_F = Q^O_F - Q^N_5$. The dashed box shows where the free budget Q^O_F would have been if there were no reconfiguration.

The reconfiguration decreases the free budget in the system, $Q^N_F = Q^O_F - Q^N$, and increases the utilization by Q^N/P.

Algorithm 3 describes adding server (Q^N_{N+1}, P) to a schedule of N servers. In the first frame where the server is added, it is scheduled at the beginning of the free budget slot. This is illustrated in Figure 12.

THEOREM 3. *Adding server (Q^N_{N+1}, P) to a schedule of N servers using Algorithm 3 satisfies condition (6) for all other servers in the system as each of them gets at least a guaranteed service during the reconfiguration of $\tilde{\beta}_j = \beta_{Q_j,P}$, $1 \le j \le N$.*

4.1.4 Increasing the Budget of an Existing ASG

Increasing the budget of a server means that in Old Mode it has budget $Q^O > 0$, and in New Mode it has budget $Q^N > Q^O$. Budgets of all other servers are unchanged. From condition (1), this is an operation that is feasible if there is sufficient free budget in the system:

$$Q^N - Q^O \le Q^O_F.$$

The reconfiguration decreases the free budget in the system, $Q^N_F = Q^O_F - (Q^N - Q^O)$, and increases the utilization of the system by $(Q^N - Q^O)/P$.

Algorithm 4 shows increasing the budget of a server from (Q^O_i, P) to (Q^N_i, P) in a schedule of N servers. In the first frame where the budget is increased, all preceding servers are activated earlier in the free budget of the previous frame by the amount of the increase of budget, and all succeeding servers are activated without change. This is illustrated in Figure 13.

THEOREM 4. *Increasing the budget of a server from (Q^O_i, P) to (Q^N_i, P) in a schedule of N servers using Algorithm 4 satisfies condition (6) for all servers in the system. Unchanged servers get at least a guaranteed service during the reconfiguration of $\tilde{\beta}_j \ge \beta_{Q_j,P}$, $1 \le j \le N$, $j \ne i$. For the increased server this is $\tilde{\beta}_i \ge \beta_{Q^O_i,P}$.*

Algorithm 4 `Increasing the budget of an ASG`

Input: $s^O_{j,k},\ 1 \le j \le N$ ▷ Schedule in last frame (k) of Old Mode
Input: P ▷ Current period
Input: Q^O_F ▷ Free budget in Old Mode
Input: (Q^O_i, P) ▷ Server to be modified with Old Mode parameters
Input: (Q^N_i, P) ▷ Server to be modified with New Mode parameters
Require: $Q^N_i - Q^O_i \le Q^O_F$
Output: $s^N_{j,k+1},\ 1 \le j \le N$ ▷ Schedule in first frame ($k+1$) of New Mode

1: **for** $j \leftarrow 1$ to N **do**
2: $\quad$ **if** $j \le i$ **then**
3: $\quad\quad s^N_{j,k+1} \leftarrow s^O_{j,k} + P - (Q^N_i - Q^O_i)$
4: $\quad$ **else if** $j > i$ **then**
5: $\quad\quad s^N_{j,k+1} \leftarrow s^O_{j,k} + P$
6: $\quad$ **end if**
7: **end for**

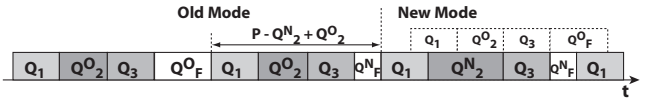

Figure 13: Increasing the budget of server (Q^O_2, P) to Q^N_2 in a schedule of three ASGs. Last frame of Old Mode has a decreased length, $P - Q^N_2 + Q^O_2$. This causes the activation times of server (Q_1, P) to be shifted earlier. Activation times of server (Q_3, P) do not change as the shorter activation frame cancels with the increased budget for all New Mode activations. Free budget has been decreased by the increase of server budget, $Q^N_F = Q^O_F - Q^N_2 + Q^O_2$. The dashed boxes show where the activations of servers (Q_1, P), (Q_2, P), (Q_3, P) and the free budget Q_F would have been if there were no reconfiguration.

4.2 Change of Period

We perform analysis given the configurations of the system (such as budgets and periods) in Old and New Modes. The results of the analysis are whether a transition is feasible with the given configurations, and in the case of feasibility with what parameters it can be executed online.

4.2.1 Increase of Period

We suppose that there are N servers in the system. In the Old Mode they operate with parameters (Q^O_i, P^O), $1 \le i \le N$, and in the New Mode with (Q^N_i, P^N), $1 \le i \le N$, where $P^O < P^N$. Assume that for every server we have that $Q^O_i \le Q^N_i$. If this is not the case, namely there is a server that requires a smaller budget in the bigger period, $Q^O_i > Q^N_i$, we can reduce its budget first by using the algorithm proposed in Section 4.1.2 as we can be sure that schedulability is satisfied with the new budget in the smaller period, and then perform the reconfiguration involving increase of period.

The proposed reconfiguration algorithm is subject to the feasibility condition that the sum of all New Mode server budgets is smaller than the Old Mode period which is expressed as follows:

$$\sum_{i=1}^{N} Q^N_i \le P^O. \tag{7}$$

The condition ensures that the increase of budgets does not lead to service guarantee violations in intervals of time beginning P^O time units before the reconfiguration and ending P^O time units after the reconfiguration. It can be related to the feasibility condition from Section 4.1.4, $\sum_{i=1}^{N}(Q^N_i - Q^O_i) \le Q^O_F$.

Algorithm 5 `Increase of Period`

Input: $s^O_{j,k},\ 1 \le j \le N$ ▷ Schedule in last frame (k) of Old Mode
Input: P^O ▷ Old Mode period
Input: P^N ▷ New Mode period
Input: $(Q^O_i, P^O),\ 1 \le i \le N$ ▷ Servers in Old Mode
Input: $(Q^N_i, P^N),\ 1 \le i \le N$ ▷ Servers in New Mode
Input: K ▷ Number of activation frames during the Reconfiguration
Require: $\sum_{i=1}^{N} Q^N_i \le P^O$
Output: $s^R_{j,k+p},\ 1 \le j \le N,\ 1 \le p \le K$ ▷ Schedule in all frames during the Reconfiguration
Output: $s^N_{j,k+K+1},\ 1 \le j \le N$ ▷ Schedule in first frame $(k+K+1)$ of New Mode

(* First frame of Reconfiguration - increase budgets *)
$s^R_{1,k+1} \leftarrow s^O_{1,k} + P^O - \sum_{i=1}^{N}(Q^N_i - Q^O_i)$
for $j \leftarrow 2$ to N **do**
  $s^R_{j,k+1} \leftarrow s^R_{j-1,k+1} + Q^N_{j-1}$
end for

(* All subsequent frames of Reconfiguration *)
for $p \leftarrow 2$ to K **do**
  for $j \leftarrow 1$ to N **do**
    $s^R_{j,k+p} \leftarrow s^R_{j,k+p-1} + P^O$
  end for
end for

(* First frame of New Mode - increase period *)
for $j \leftarrow 1$ to N **do**
  $s^N_{j,k+K+1} \leftarrow s^R_{j,k+K} + P^N$
end for

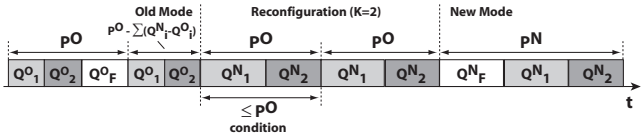

Figure 14: Increase of period with $K = 2$.

When condition (7) is not satisfied, i.e. staying in the most pessimistic configuration is not feasible, the reconfiguration algorithm would need to go through *one or more intermediate* modes (budgets and periods) where for each successive pair of them condition (7) holds. We will not discuss this further and assume that the feasibility condition is met.

The algorithm for performing safely the increase of period can be summarized in three steps: (1) Increase to New Mode budgets following Algorithm 4. (2) Schedule the ASGs for $K \ge 1$ activation frames using the New Mode budgets and Old Mode period. (3) Increase to New Mode period by increasing free budget. The second step of the algorithm we denote as the Reconfiguration phase which is K activation frames long. We suppose that it has Old Mode period but it can actually have a shorter period which would require a small modification of our analysis. At the moment we assume that K is given as input to the algorithm, later we will show how to compute it. Algorithm 5 describes the details for performing the increase of period. It is illustrated in Figure 14.

The following theorem gives a lower bound for the guaranteed resource supply of an ASG during an increase of period reconfiguration.

THEOREM 5. *Reconfiguring a server from (Q^O_i, P^O) to (Q^N_i, P^N) in a schedule of N servers using Algorithm 5 provides at least a guaranteed service of:*[1]

$$\tilde{\beta}_i(\Delta) = \min\{\beta_{Q^O_i,P^O}(\Delta),\ \beta_{Q^N_i,P^N}(\Delta),$$
$$(\beta_{Q^O_i,P^O} \otimes \beta_{Q^N_i,P^N})(\Delta - K\cdot P^O + P^O - Q^O_i) \quad (8)$$
$$+ \beta_{Q^N_i,P^O}(K\cdot P^O)\}, \quad (9)$$

[1]The $\otimes$ is the min-plus convolution operator which is defined as: $(a \otimes b)(\Delta) = \inf_{0\le\lambda\le\Delta}\{a(\Delta-\lambda) + b(\lambda)\}$

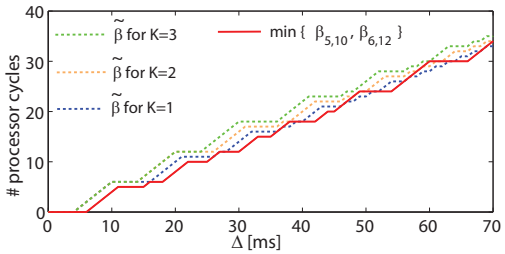

Figure 15: Effect of $K = \{1,2,3\}$ for server S_B from Example 1. Only $K = 3$ is feasible.

which satisfies condition (6) when $K \ge 1$ is found as:

$$K = \max_{1\le i\le N}\Big\{\min\Big\{\kappa \mid \forall\Delta \in \mathbb{R}^{\ge 0},\ \kappa \in \mathbb{Z}^+,$$
$$(\beta_{Q^O_i,P^O} \otimes \beta_{Q^N_i,P^N})(\Delta - \kappa\cdot P^O + P^O - Q^O_i) + \beta_{Q^N_i,P^O}(\kappa\cdot P^O)$$
$$\ge \min\{\beta_{Q^O_i,P^O}(\Delta),\ \beta_{Q^N_i,P^N}(\Delta)\}\Big\}\Big\}.$$

The guaranteed service in the above theorem can be explained informally as follows. It is computed as the minimum of the services from Old Mode, New Mode, and an expression which describes the service in time intervals that span Old Mode, Reconfiguration, and New Mode. The last one consists of two subexpressions. Expression (8) lower bounds the service guaranteed in the time window part that is *outside* of the Reconfiguration time window and hence the service curve depends only on the Old and the New Modes parameters, and it is 'shifted to the right' by the size of the Reconfiguration time window which is at most $K \cdot P^O$ time units. Expression (9) lower bounds the service guaranteed only in the Reconfiguration time window which uses New Mode budgets with Old Mode period, and the service is defined for a fixed length interval of size $K \cdot P^O$.

In expressions (8) and (9), we can increase the size of the Reconfiguration phase by increasing the number of activation frames in it K. In order to meet condition (6) for each server, we have to find the *minimum* K that will make the guaranteed service $\tilde{\beta}_i$ greater or equal to the minimum of the Old and New Modes services. After doing this for all servers, we have to take the *maximum* K which will make the reconfiguration feasible for the whole system.

We can find the minimum K for a server efficiently by starting with an initial value of $K = 1$. If this is not feasible, we choose successive values of K by using binary search until the smallest one is found that is feasible. With bigger K we are increasing the service guaranteed in the Reconfiguration which is service greater than Old Mode and New Mode services (it has the larger New Mode budget and the smaller Old Mode period), therefore we are guaranteed to find a finite value for K which will make condition (6) satisfied.

EXAMPLE 6. *We can illustrate this by considering server S_B from Example 1. It will need $K = 3$ to perform a safe reconfiguration from $(5,10)$ to $(6,12)$. This is illustrated in Figure 15 as well as the violations of condition (6) for $K = \{1,2\}$. The trace showing the violation for $K = 2$ for server S_B is in Figure 16.*

4.2.2 Decrease of Period

This scenario is very similar to the one for increasing the period. Because of the lack of space, we only give the main points. In the Old Mode servers operate with parameters (Q^O_i, P^O), $1 \le i \le N$, and in the New Mode with (Q^N_i, P^N), $1 \le i \le N$, where $P^O > P^N$. We assume that for every server we have that $Q^O_i \ge Q^N_i$.

It is subject to the feasibility condition that the sum of Old Mode budgets is smaller than the New Mode period which is expressed as $\sum_{i=1}^{N} Q^O_i \le P^N$.

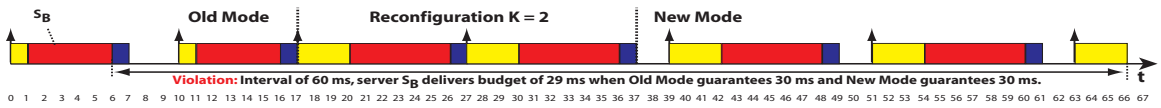

Figure 16: Violation of condition (6) when increasing period with $K = 2$ for server S_B from Example 1.

The algorithm can be summarized in three steps: (1) Decrease to New Mode period by decreasing free budget. (2) Schedule the ASGs for $K \geq 1$ activation frames using Old Mode budgets and New Mode period. (3) Decrease budgets by using Algorithm 2. Algorithm 6 describes the details for performing the decrease of period. It is illustrated in Figure 17.

THEOREM 6. *Reconfiguring a server from (Q_i^O, P^O) to (Q_i^N, P^N) in a schedule of N servers using Algorithm 6 provides at least a guaranteed service of:*

$$\tilde{\beta}_i(\Delta) = \min\{(\beta_{Q_i^O,P^O} \otimes \beta_{Q_i^N,P^N})(\Delta - K \cdot P^N + P^N - Q_i^N) + \beta_{Q_i^O,P^N}(K \cdot P^N),\ \beta_{Q_i^O,P^O}(\Delta),\ \beta_{Q_i^N,P^N}(\Delta)\},$$

which satisfies condition (6) when $K \geq 1$ is found as:

$$K = \max_{1 \leq i \leq N} \Big\{ \min \Big\{ \kappa \mid \forall \Delta \in \mathbb{R}^{\geq 0},\ \kappa \in \mathbb{Z}^+, (\beta_{Q_i^O,P^O} \otimes \beta_{Q_i^N,P^N})(\Delta - \kappa \cdot P^N + P^N - Q_i^N) + \beta_{Q_i^O,P^N}(\kappa \cdot P^N) \geq \min\{\beta_{Q_i^O,P^O}(\Delta),\ \beta_{Q_i^N,P^N}(\Delta)\} \Big\} \Big\}.$$

5. CASE STUDY

Here, we consider a multi-mode real-time system that executes two applications. Application 1 can run in two modes denoted as mode 1 and mode 2. In mode 1, there is a single task which processes a single event stream described by a period $p = 5$ms, jitter $j = 10$ms, and minimum interarrival time between two events $d = 1$ms. Each event has a worst-case execution time of $c = 2$ms, and it needs to be processed within a relative deadline of $D = 9$ms. Similarly, in mode 2 there is a single task but it processes an event stream with parameters $p = 40$ms, $j = 20$ms, $d = 20$ms, $c = 7$ms, and $D = 25$ms. Application 2 is a single mode application, it has a single task that processes one event stream with parameters $p = 20$ms, $j = 15$ms, $d = 5$ms, $c = 1$ms, and $D = 30$ms. The system schedules the two applications using two servers (Q_1, P) and (Q_2, P). We suppose that each context switch takes 0.3ms. The utilization of the system, U, can be computed as $U = (Q_1 + 0.3 + Q_2 + 0.3)/P$.

The designer of this system needs to select the configuration parameters of the ASG schedule such as the *minimum* required budgets that make the two applications schedulable, and the size of the servers period. The design objective is to minimize utilization because other soft real-time applications use the unused resources while guaranteeing the real-time requirements. Then the solution depends on the mode that application 1 is currently in. Figure 18 shows the total utilization of the system as a function of the period of the servers considering the two modes of application 1, where the period varies from 1ms to 50ms. When application 1 is in mode 1, the system has the minimum utilization ($U = 0.768$) with servers period $P = 12.5$ms, and allocated budgets for application 1 and application 2, $Q_1 = 8$ms and $Q_2 = 1$ms, respectively. When application 1 is in mode 2, however, the system has the minimum utilization ($U = 0.427$) achieved for period $P = 22.5$ms, and budgets $Q_1 = 7$ms and $Q_2 = 2$ms.

Since the mode of application 1 changes dynamically during runtime, it is not possible to fix the parameters of the scheduler at design time. If the parameters are set to the optimal ones for mode 1, when operating in mode 2 the system would have a 15% utilization overhead. Similarly fixing the parameters optimally for mode 2, the utilization overhead would be 14% when the system is in mode 1.

We can solve the above problem by using the algorithms proposed in this paper. Let us consider two scenarios.

Scenario 1: When application 1 is in mode 1, we run the two ASGs corresponding to the two applications with parameters $(8, 12.5)$ and $(1, 12.5)$ which give us the lowest system utilization. When application 1 switches to mode 2, it notifies the Server Manager (SM) and it requests a switch to the minimum budget for mode 2 of $(4.7, 12.5)$. The SM can grant this budget using Algorithm 2. Afterwards the SM can reconfigure the two ASGs and increase their period to the one which makes the system utilization the smallest. The SM can use Algorithm 5 with $K = 1$ to reconfigure the system from $(4.7, 12.5)$ and $(1, 12.5)$ to $(7, 22.5)$ and $(2, 22.5)$.

Scenario 2: When application 1 has to switch back to mode 1, it first notifies the SM which by using Algorithm 6 with $K = 1$ reconfigures the two servers from $(7, 22.5)$ and $(2, 22.5)$ back to $(4.7, 12.5)$ and $(1, 12.5)$. Then the SM increases the budget for application 1 using Algorithm 4 from 4.7 to 8. Afterwards, application 1 is notified and can safely switch to mode 1.

Note that the SM takes advantage of the fact that mode 1 is more heavily loaded than mode 2 for application 1. Therefore, the SM optimizes the server period when the application is in the lightly loaded mode. This means that in Scenario 1, the application mode change is done before the resource optimization. And in Scenario 2, it is done after the resource optimization. This is feasible with our algorithms as they are completely deterministic and the time

Algorithm 6 `Decrease of Period`

Input: $s_{j,k}^O,\ 1 \leq j \leq N$ ▷ Schedule in last frame (k) of Old Mode
Input: P^O ▷ Old Mode period
Input: P^N ▷ New Mode period
Input: $(Q_i^O, P^O),\ 1 \leq i \leq N$ ▷ Servers in Old Mode
Input: $(Q_i^N, P^N),\ 1 \leq i \leq N$ ▷ Servers in New Mode
Input: K ▷ Number of activation frames during the Reconfiguration
Require: $\sum_{i=1}^N Q_i^O \leq P^N$
Output: $s_{j,k+p}^N,\ 1 \leq j \leq N,\ 1 \leq p \leq K$ ▷ Schedule in all frames during the Reconfiguration
Output: $s_{j,k+K+1}^N,\ 1 \leq j \leq N$ ▷ Schedule in first frame $(k + K + 1)$ of New Mode

(* First frame of Reconfiguration - with decreased period *)
1: **for** $j \leftarrow 1$ to N **do**
2: $\quad s_{j,k+1}^R \leftarrow s_{j,k}^O + P^O$
3: **end for**

(* All subsequent frames of Reconfiguration *)
4: **for** $p \leftarrow 2$ to K **do**
5: $\quad$ **for** $j \leftarrow 1$ to N **do**
6: $\quad\quad s_{j,k+p}^R \leftarrow s_{j,k+p-1}^R + P^N$
7: $\quad$ **end for**
8: **end for**

(* First frame of New Mode - decrease budgets *)
9: $s_{1,k+K+1}^N \leftarrow s_{1,k+K}^R + P^N$
10: **for** $j \leftarrow 2$ to N **do**
11: $\quad s_{j,k+K+1}^N \leftarrow s_{j-1,k+K+1}^N + Q_{j-1}^N$
12: **end for**

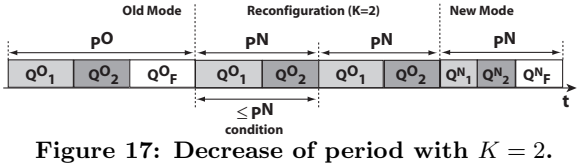

Figure 17: Decrease of period with $K = 2$.

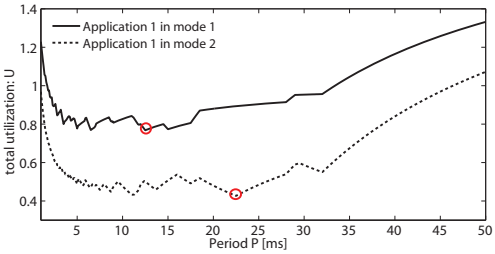

Figure 18: Total utilization for period varying from 1ms up-to 50ms considering the two different modes of application 1. The circles on the graphs denote the points of minimum utilization.

needed for a reconfiguration can be safely and accurately upper bounded in advance. It is also possible to perform the resource optimization when the system is more heavily loaded however, the reconfiguration process will take longer.

In summary, we can guarantee an optimal resource allocation in environments where applications are added or removed dynamically, or perform mode-changes. With the proposed algorithms, the schedulability of the applications is never compromised during the reconfiguration process.

Setup: The servers and applications have been modeled with the Matlab Real-Time Calculus Toolbox [32]. The exploration of the minimum required budgets for different periods in Figure 18 has been done with the Real-Time Interfaces methodology as described in [31]. The exploration took less than 15s to perform on a commodity laptop considering discretization of the period with steps of 0.1ms. The feasibility check for the value of K took less than 1s.

6. CONCLUSION

The paper considers the problem of adaptive resource reservations using servers in hard real-time systems. It classifies the possible problems that may occur during online server reconfigurations and establishes conditions of how to avoid them. It defines a statically TDMA scheduled adaptive server that provides resource guarantees not only during operation, but also during reconfigurations. The paper identifies the possible reconfiguration scenarios for such a server and provides algorithms and schedulability analysis for each of them. The analysis is based on Real-Time Calculus which even for the simplest case of TDMA scheduled servers is not trivial. The future direction of this work is to explore the problem for other kinds of servers such as the deferrable server and the CBS, and establish similar algorithms and analysis for their online reconfigurations.

7. REFERENCES

[1] L. Abeni and G. Buttazzo. Adaptive bandwidth reservation for multimedia computing. In *Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA)*, pages 70–77, 1999.

[2] L. Abeni and G. Buttazzo. Hierarchical QoS management for time sensitive applications. In *Proceedings of the Seventh Real-Time Technology and Applications Symposium (RTAS)*, pages 63–72, 2001.

[3] L. Abeni and G. Buttazzo. Resource reservation in dynamic real-time systems. *Real-Time Systems*, 27(2):123–167, 2004.

[4] S. Baruah, D. Chen, S. Gorinsky, and A. Mok. Generalized multiframe tasks. *Real-Time Systems*, 17(1):5–22, 1999.

[5] S. A. Brandt, S. Banachowski, C. Lin, and T. Bisson. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS)*, pages 396–407, 2003.

[6] G. Buttazzo and L. Abeni. Adaptive rate control through elastic scheduling. In *Proceedings of the 39th IEEE Conference on Decision and Control (CDC)*, volume 5, pages 4883–4888, 2000.

[7] S. Craciunas, C. Kirsch, H. Payer, H. Rock, and A. Sokolova. Programmable temporal isolation through variable-bandwidth servers. In *IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 171 –180, 2009.

[8] R. Cruz. A calculus for network delay, Parts 1 & 2. *IEEE Transactions on Information Theory*, 37(1), 1991.

[9] T. Cucinotta, L. Palopoli, and G. Lipari. FRESCOR Deliverable D-AQ2v2: control algorithms for coordinated resource-level and application-level adaptation v2, 2008.

[10] A. B. de Oliveira, E. Camponogara, and G. Lima. Dynamic reconfiguration in reservation-based scheduling: An optimization approach. In *15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 173–182, 2009.

[11] G. Fohler. Changing operational modes in the context of pre run-time scheduling. *IEICE Transactions on Information and Systems*, 76(11):1333–1340, 1993.

[12] Q. Guangming. An earlier time for inserting and/or accelerating tasks. *Real-Time Systems*, 41(3):181–194, 2009.

[13] M. G. Harbour, D. Sangorrín, and M. T. de Esteban. FRESCOR Deliverable D-AT2: schedulability analysis techniques for distributed systems, 2009.

[14] T. A. Henzinger, B. Horowitz, and C. M. Kirsch. Giotto: A time-triggered language for embedded programming. In *Proceedings of the First International Workshop on Embedded Software (EMSOFT)*, pages 166–184, 2001.

[15] J. Y. Le Boudec and P. Thiran. *Network calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer, 2001.

[16] A. Maxiaguine, S. Künzli, and L. Thiele. Workload characterization model for tasks with variable execution demand. In *Design, Automation Test in Europe Conference (DATE)*, volume 2, pages 1040–1045, 2004.

[17] C. Mercer, R. Rajkumar, and J. Zelenka. Temporal protection in real-time operating systems. In *Proceedings 11th IEEE Workshop on Real-Time Operating Systems and Software (RTOSS)*, pages 79–83, 1994.

[18] P. Pedro and A. Burns. Schedulability analysis for mode changes in flexible real-time systems. In *Proceedings 10th Euromicro Workshop on Real-Time Systems*, pages 172–179, 1998.

[19] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 26(2):161–197, 2004.

[20] L. Sha, J. P. Lehoczky, and R. Rajkumar. Solutions for some practical problems in prioritized preemptive scheduling. In *Real-Time Systems Symposium (RTSS)*, pages 181–191, 1986.

[21] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems*, 1(3):243–264, 1989.

[22] B. Sprunt, L. Sha, and J. P. Lehoczky. Aperiodic task scheduling for hard real-time systems. *Real-Time Systems*, 1(1):27–60, 1989.

[23] M. Spuri and G. Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *Real-Time Systems*, 10(2):179–210, 1996.

[24] N. Stoimenov, S. Perathoner, and L. Thiele. Reliable mode changes in real-time systems with fixed priority or EDF scheduling. In *Design, Automation Test in Europe Conference (DATE)*, pages 99–104, 2009.

[25] N. Stoimenov, L. Thiele, L. Santinelli, and G. Buttazzo. Resource adaptations with servers for hard real-time systems. ftp://ftp.tik.ee.ethz.ch/pub/publications/TIK-Report-320.pdf, ETH Zurich, 2010.

[26] J. K. Strosnider, J. P. Lehoczky, and L. Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, 44(1):73–91, 1995.

[27] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proceedings of the 2000 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 4, pages 101–104, 2000.

[28] K. W. Tindell, A. Burns, and A. J. Wellings. Mode changes in priority pre-emptively scheduled systems. In *Real-Time Systems Symposium (RTSS)*, pages 100–109, 1992.

[29] M. G. Valls, A. Alonso, and J. A. de la Puente. Mode change protocols for predictable contract-based resource management in embedded multimedia systems. In *Second International Conference on Embedded Software and Systems*, pages 221–230, 2009.

[30] E. Wandeler and L. Thiele. Interface-based design of real-time systems with hierarchical scheduling. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 243–252, 2006.

[31] E. Wandeler and L. Thiele. Optimal TDMA time slot and cycle length allocation. In *Proceedings of the 2006 Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 479–484, 2006.

[32] E. Wandeler and L. Thiele. Real-Time Calculus (RTC) Toolbox. http://www.mpa.ethz.ch/Rtctoolbox, 2006.