An Algorithm for Online Reconfiguration of Resource Reservations for Hard Real-Time Systems

Pratyush Kumar, Nikolay Stoimenov, Lothar Thiele
Computer Engineering and Networks Laboratory,
ETH Zurich, Switzerland
firstname.lastname@tik.ee.ethz.ch

***Abstract*—Nowadays, real-time applications expect the supporting computing system to be reconfigured at run-time. Even during such reconfiguration, timing requirements of the applications must be met. By extension, such requirements are relevant in the design of resource reservations techniques. In this work, we consider such a reconfiguration of the reservation provided by a constant bandwidth server (CBS). Firstly, we define an exact notion of correctness of a server's reconfiguration. Then we design a provably correct server algorithm R-CBS that allows for run-time reconfiguration of a standard CBS. The algorithm maintains specific information about the execution trace and uses it to efficiently perform the reconfiguration at the earliest possible time. We highlight the advantages of R-CBS in comparison to reconfiguration of TDMA servers and in reconfiguring multiple servers simultaneously.**

I. Introduction

Nowadays, real-time applications expect the supporting computing systems to reconfigure or change *modes* at run-time. There are several reasons why such reconfiguration is expected. In systems with multiple applications, mode change may be needed to support different application scenarios. Even otherwise, a single application by itself may have several operating modes which may need to be changed at run-time for instance in response to input data. In addition, hardware systems may also exhibit run-time reconfiguration, for instance a processor that is getting hot may reduce its speed of execution or offload tasks to another processor.

In hard real-time systems, it is essential that certain timing properties of the system are guaranteed, even during such run-time reconfiguration. Several research works have studied this problem. These studies focus on three key aspects: (a) characterizing the change in configuration, (b) designing a runtime reconfiguration scheme, and (c) analyzing guarantees on timing properties before, during and after the reconfiguration.

Most of this research considers direct execution of tasks on resources. Consequently, the reconfiguration reflects in the mode of the jobs, which are labelled either *old-mode* or *new-mode*. For fixed priority systems, early results by Tindell and others [1] and [2] proposed delaying the new-mode tasks until all buffered old-mode tasks are completed. Extensions allowing simultaneous execution of old-mode and new-mode tasks, with and without periodicity were later proposed. A survey of these works is presented in [3]. For the Earliest Deadline First (EDF) policy, Buttazzo et al. [4] considered insertion of a new task and/or acceleration of existing tasks using an elastic scheduling technique. The results have been improved in [5].

In these and similar studies, the three aspects we mentioned earlier, namely, characterizing the change in configuration, designing the reconfiguration scheme and analyzing timing properties, are application-dependent. In other words, the above steps can only be done given the exact characterization of the tasks. It is natural, thus, that the above-mentioned and similar studies consider periodic task arrivals with fixed worst-case execution times. Indeed, it is unclear how, if at all, such studies can be extended to be applicable in systems that exhibit variability (such as jitter) or cannot be accurately characterized at design-time.

The challenge of executing applications with variability or imprecise models is not new in the real-time domain. The use of *servers* has long been advocated to create reservations on shared resources and then to execute applications within those reservations. Focus has also been rightly laid on ensuring timing isolation, such that one application is not affected by the variability or imprecise models of other applications.

In dynamically reconfiguring systems, this care to be resilient to variation and imprecision can only become more significant. Some research works have considered this specific motivation and we review some of them here. In [6], the rate-based earliest deadline (RBED) scheduler has been proposed to reconfigure by changing the utilization reserved for tasks. In particular, results are presented on when and by how much the utilization of tasks can be changed. However, focus has not been laid on specifying the timing guarantee that the scheduler provides the application that it serves. As a related point, the algorithm for reconfiguration does not consider self-interference: the effect of some jobs of the application (or task) on other jobs of that very application. In our approach, we define and prove the timing guarantee that is provided across the reconfiguration. In another work, Craciunas et al. [7] propose the variable-bandwidth server (VBS) which allows reconfiguration of the constant bandwidth server (CBS) [8].

However, only specific reconfiguration schemes are allowed and the analysis of the system schedulability is based on the worst-case modes of the different applications. In contrast, our focus is on defining the reconfiguration algorithm in greater detail, and providing an exact characterization of the system schedulability.

Recently, Santinelli et al. [9] presented an analysis technique to verify schedulability of a multi-moded CBS. The authors propose a reconfiguration mechanism wherein the reconfiguration happens after a fixed transition delay. The authors extend the demand bound function schedulability test to be applicable to such a reconfiguring CBS and compute the values of the minimum delay. While, such a reconfiguration mechanism is light-weight to implement, it does not fully exploit the working of the CBS algorithm. As we will show, it is possible to develop a reconfiguration algorithm that considers the state of the system and performs the reconfiguration at the earliest time. Furthermore, a robust notion of correctness of the mode-change has not been established in [9].

Also recently, Fisher et al. [10] proposed tractable schedulability analysis of reservation mechanisms under mode change. The authors propose a model of the reconfiguration where the time between the reconfiguration is statically specified. Given such a model the authors propose interesting ways to perform the schedulability check efficiently. In contrast, our work is directed towards providing an algorithm to perform the reconfiguration, and by construction provide a schedulability check.

Stoimenov et al. [11] consider the problem of reconfiguring the TDMA server while defining explicitly the service provided by the server. We feel that our work closely rivals that in [11]. In contrast, we consider the Constant Bandwidth Server (CBS). The close match motivates us to present results comparing the reconfigurability of either server. We present such results in Section VI-A while concluding that CBS is more suited to reconfigurations than TDMA, on several accounts.

Let us revisit again the three key aspects in multi-moded systems, with reference to reconfiguration of a server such as the CBS. Firstly, characterizing the change in configuration involves only specifying the server parameters in either mode, and is independent of the application model. Secondly, we need to provide a reconfiguration scheme whereby the server algorithm is modified after the initiation of the mode-change. Such a scheme must consider the prevailing conditions and complete the reconfiguration promptly and with small overhead. Thirdly, we have to define a notion of correctness in the timing guarantee provided by the server to the application and prove that this guarantee is met by the designed server, for *all* possible application models. Such a guarantee would enable us to retain the advantage of using servers, namely its independence form the application model and by extension, any variability. Furthermore, in all the above aspects, it is desirable to retain the isolation amongst servers sharing a common resource. This will enable us to compositionally perform the reconfiguration of several servers.

Contributions Our contributions are in the following five fronts:

1) We define correctness of a reservation reconfiguration with respect to (a) schedulability of the system, and (b) service guarantees provided by the server.
2) We present a server algorithm called R-CBS which can reconfigure a standard CBS while satisfying the defined notion of correctness.
3) We show that the R-CBS works efficiently by maintaining minimal details about the execution trace and completes the reconfiguration at the earliest possible instance.
4) We highlight the advantages of reconfiguring CBS in comparison to the reconfiguration of TDMA servers as presented in [11].
5) We demonstrate and analyze simultaneous reconfiguration of several R-CBS due to the isolation-preserving design.

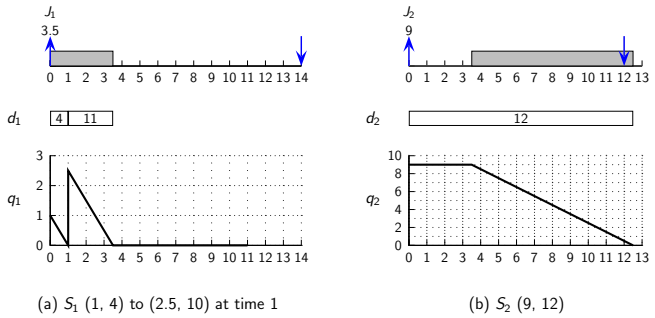
Outline The rest of the paper is organized as follows. In Section II we illustrate with examples how intuitive methods for reconfiguration of CBS do not work. We state some known results on CBS in Section III. In Section IV, we formalize the correctness of a reconfiguration of a server. In Section V we present our algorithm R-CBS. We show that the algorithm is correct and minimal. Finally, in Section VI we present some quantitative results and conclude in Section VII.

II. Motivating Examples

In this section, we present examples to highlight that intuitive methods to reconfigure CBS can result in unexpected timing violations. In a latter section, we will see how using our proposed reconfiguration scheme, these examples are correctly scheduled.

We first establish some notation. A constant bandwidth server is characterized by a maximum budget Q and period P. We denote this pair as a tuple (Q, P). The server deadline and remaining capacity are denoted as d and q, respectively. We defer a more complete definition to Section III.

Example 1: Immediate mode change does not work: In the first example, we show that making a mode-change immediately can lead to unexpected timing violations. We consider two servers S_1 and S_2 with parameters $(1, 4)$ and $(9, 12)$, respectively. At time 1, server S_1 is requested to be reconfigured to parameters $(2.5, 10)$. The system is schedulable with either set of parameters.



(a) S_1 (1, 4) to (2.5, 10) at time 1 (b) S_2 (9, 12)

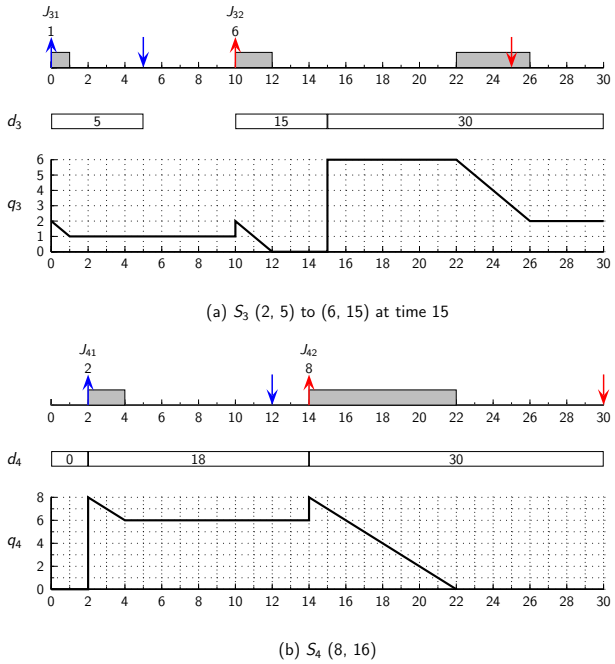
Figure 1. Reconfiguration violating deadline of another task (isolation guarantee violation)

In Figure 1, we show a trace of job arrivals. Arrival of jobs are denoted with upwards arrows and their deadlines with downwards arrows. The execution time of the job is mentioned on top of the upward arrow. Also shown are the values of the server deadline and remaining budget as functions of time.

In the shown trace, S_1 reconfigures immediately at time 1. As a consequence, Job J_2 misses its deadline of 12. This is unexpected since this job is being served by a server with a period equal to the relative deadline and maximum budget equal to the execution time of the job. Independent of the reconfiguration of S_1, as long as schedulability is guaranteed, job J_2 must complete within its deadline. The observed timing discrepancy is an example of *isolation guarantee violation*: the reconfiguration of the server affects another server or task in the system, thereby violating the isolation property expected to be guaranteed by servers.

Example 2: Mode-change at end of period does not work: One may expect that in the Example 1 waiting until the end of the server period, before reconfiguring S_1, might lead to correct operation. This is indeed true. But, as we show in the second example, reconfiguring at the end of the period does not always work.

Consider two servers S_3 and S_4 with parameters $(2, 5)$ and $(8, 16)$, respectively. Server S_3 is requested to reconfigure at time 15, to the new set of parameters $(6, 15)$. Again, the system is schedulable with both sets of parameters. We plot the system behavior for a specific job arrival pattern, such that the reconfiguration request comes at the end of the period. As shown, job J_{32} misses its deadline. This is unexpected since job J_{32} has an execution time of 6 and must complete within a relative deadline of 15 for *either* set of parameters of S_3. Independent of the working of the rest of the system, given system schedulability, job J_{32} must complete within its deadline. The observed timing anomaly is an example of a *service guarantee violation*: the reconfiguration of a server is such that a job served by the server misses its deadline, though for either set of



(a) S_3 (2, 5) to (6, 15) at time 15

(b) S_4 (8, 16)

Figure 2. Reconfiguration that violates deadline of a job that must be met in either configuration (service guarantee violation)

the parameters of the server the job is expected to meet its deadline.

With the examples we have shown that there are timing properties expected from the system during a reconfiguration. These properties relate to either (a) isolating the effects of the reconfiguration from other tasks or servers in the system, or (b) meeting of deadlines of jobs served by the server which straddle the two configurations of the server. Our goal thus will be to first characterize these expectations quantitatively and thereby identify properties of a correct reconfiguration scheme. Then, using this definition of correctness we will proceed to design a correct reconfiguration scheme.

III. Constant Bandwidth Server

Constant Bandwidth Server (CBS) was proposed by Abeni and Buttazzo in [8] as an efficient implementation of a bandwith reservation startegy. In this section, we present the algorithm of CBS using a general template for dynamic priority servers. We then present two known results on CBS.

A. Template to Specify Dynamic Priority Servers

To compactly specify the algorithms for dynamic priority servers, a general template of a dynamic priority server was presented in [12]. We briefly summarize this template. Each server has a task queue associated with it where the jobs it executes are queued. The system may have several such task queues which are served by an EDF scheduler. The scheduler maintains an EDF queue where task queues contending for

the resource are queued. A task queue (or equivalently the server) has a remaining capacity q and a deadline d used by the EDF scheduler to arbitrate among task queues. In addition, it has a re-insertion time r which denotes the time when the task queue must again be inserted into the EDF queue, if not in it already. The remaining capacity of the task queue is decremented upon execution by the EDF scheduler. When either the remaining capacity is exhausted, or the task queue becomes empty, a signal `TQRemoved` is raised. When a job arrives while the task queue is empty a signal `TQNonEmpty` is raised. The role of the server is to update the task queue parameters (q, d and r) when either of the two signals are raised.

B. Server Algorithm

A CBS is characterized by two parameters: a maximum budget Q and a period P, represented as a tuple (Q, P). The quantity (Q/P) is called the utilization of the server and is denoted by U. Two variants of CBS exist, namely Soft- and Hard-CBS, depending on when the task queue is re-inserted into the EDF queue after depleting the remaining budget. We describe the server algorithm of both variants in Figure 3. In all notation, t denotes the current time.

1) Initial conditions, i.e., at $t = 0$:
   a) The task queue is not in the EDF queue,
   b) $q := 0$, $d := 0$, $r := 0$.
2) When `TQNonEmpty` is raised:
   a) If $q \geq (d - t) \cdot U_s$, then
      i) $q := Q$,
      ii) $d := t + P$.
   b) If $r < t$ then
      i) $r := t$.
3) When `TQRemoved` is raised:
   a) If $q = 0$, then
      i) (Soft CBS) $r := t$,
         (Hard CBS) $r := d$,
      ii) $q := Q$,
      iii) $d := d + P$.

Figure 3. CBS algorithm

C. Existing results on CBS

We now present two existing results on CBS which characterize the timing properties of the server.

Theorem 1 (Utilization of CBS [8])**.** *The CPU utilization of a CBS with parameters (Q, P) is $U = (Q/P)$, independently of the computation times and the arrival pattern of the served jobs.*

From the above theorem, the following lemma on the schedulability of CBS follows.

Lemma 2 (Schedulability of CBS)**.** *Let a CBS with a utilization U be scheduled in a system, where the sum of the utilization of all other tasks and/or servers is $U_{\texttt{other}}$. Then, the system is schedulable if*

$$U_{\texttt{other}} + U \leq 1. \tag{1}$$

Let us interpret the above result. In any time interval of some length, say Δ, if the sum of the execution times demanded by all other tasks and/or servers with deadlines within this time interval, is no more than $U_{\texttt{other}} \cdot \Delta$, then this demand would indeed be provided by the system, in the presence of the CBS with utilization $U \leq 1 - U_{\texttt{other}}$. This guarantee provided by the CBS to other tasks and servers is independent of the execution times and arrival patterns of the jobs served by the CBS, and can be viewed as the *isolation guarantee*.

To describe the next result we define the notion of an arrival function. Consider a stream of jobs arriving to be served by a server. Then the input arrival function, denoted as $R(t)$, is the total execution demand of all jobs arriving in the time interval $[0, t)$. Similarly, the output arrival function, denoted as $R'(t)$, is the total execution provided by the server to the jobs in R in the time interval $[0, t)$. We can now present the following result.

Theorem 3 (Service Guarantee of CBS [13])**.** *If a stream of jobs with an input arrival function R is served in First-Come-First-Serve (FCFS) manner by a CBS with parameters (Q, P), then the output arrival function R' is bounded as follows:*

$$R'(t) \geq R(t) \otimes \beta(Q, P, t). \tag{2}$$

where,

$$(f \otimes g)(t) \stackrel{def}{=} \inf_{0 \leq u \leq t} \{f(u) + g(t - u)\}, \tag{3}$$

$$\beta(Q, P, t) \stackrel{def}{=} \left\lfloor \frac{t}{P} \right\rfloor Q. \tag{4}$$

The above bound on the output arrival function can be used to compute the worst-case delay suffered by the jobs of the input stream [13] as the largest horizontal distance between the functions R and R'. The theorem specifies the service guaranteed to the jobs served by the CBS, independent of the behavior of the rest of the system, and can be viewed as a *service guarantee*.

The above properties quantify the guarantees provided by CBS to the rest of the system and to jobs it serves. The efficient server algorithm with clear guarantees make CBS a popular choice for reserving resources.

IV. Properties of a Reconfigurable CBS

In this section, we discuss the details of a reconfiguration: the interface signals, the guarantees and desirable properties, which lay the ground for the proposed algorithm.

A. Interface for reconfiguration

Reconfiguration of a server is initiated by the scheduler by raising a signal denoted as `ReConfReq`. Thus, along with the two original signals `TQNonEmpty` and `TQRemoved`(defined in Section III-A), we have a new signal `ReConfReq` to which the server must respond to. We denote the time at which signal `ReConfReq` is raised as t_R. The scheduler also provides the new set of server parameters denoted (Q', P'). We represent the new utilization (Q'/P') as U'.

On its part, the server raises a signal `ReConfAck`, at time t_A, which communicates to the system that the reconfiguration has been performed with respect to utilization provided to the rest of the system. We will precisely define this later in this section. In addition, the server raises a signal `ReConfFin` at time t_F. This signal indicates that the server is capable of performing a new reconfiguration without affecting the real-time properties.

B. Guarantees of a reconfigurable CBS

In Section III, we presented the two guarantees, namely isolation and service guarantees, that a standard (without reconfiguration) CBS provides. For a CBS with reconfiguration, these guarantees must be modified. We now define these modified guarantees.

During reconfiguration, the maximum allowed utilization of other tasks and servers of the system will vary over time. We represent the sum of utilization of all other tasks and servers as a function of time denoted as $U_{\texttt{other}}(t)$. Then, we define the following schedulability requirement.

Definition 1 (Schedulability guarantee of CBS with reconfiguration)**.** *Let a CBS with a utilization U be scheduled in a system with the utilization of all other tasks and servers is $U_{\texttt{other}}(t)$. Let U' be the utilization of the CBS in the new configuration. Then, the system is schedulable if*

$$\begin{aligned} U_{\texttt{other}}(t) &\leq 1 - U, && t < t_R, \\ &\leq 1 - \max(U, U'), && t \in [t_R, t_A], \\ &\leq 1 - U', && t > t_A. \end{aligned} \tag{5}$$

The above condition says that, for a correctly reconfiguring CBS, the utilization before the reconfiguration and after time t_A equals the old and new utilizations, respectively. In the transition phase, if any, in the interval $[t_R, t_A]$, the higher of the two utilizations is consumed.

Due to the reconfiguration, the service guarantee proved in Theorem 3 should also be refined. Intuitively, during the reconfiguration, the server must provide at least the minimum of service provided by a standard CBS in either of the two configurations. We formalize with the following definition.

Definition 2 (Service guarantee of CBS with reconfiguration)**.** *Let a CBS with parameters (Q, P) be reconfigured to have parameters (Q', P'). Let the input arrival function of the stream of jobs served by the CBS be R. Then the output arrival function should be lower-bounded as*

$$\begin{aligned} R'(t) &\geq R(t) \otimes \beta(Q, P, t), && t < t_R, \\ &\geq R(t) \otimes \beta_{\min}(t), && t \in [t_R, t_F], \\ &\geq R(t) \otimes \beta(Q', P', t), && t > t_F. \end{aligned} \tag{6}$$

where,

$$\beta_{\min}(t) = \min(\beta(Q, P, t), \beta(Q', P', t)). \tag{7}$$

The above condition provides a service guarantee and holds for *any* given input arrival trace $R(t)$. It says that before the reconfiguration and after time t_F the service guaranteed is as in a standard CBS with respective parameters. In the transition phase, if any, in the interval $[t_R, t_F]$ the lower of the two service functions is guaranteed.

C. Desirable properties of a reconfigurable CBS

Apart from the above guarantees that a reconfigurable CBS must provide, there are some desirable properties that can be identified.

Promptness: The reconfigurable server must generate the `ReConfAck` signal at the earliest, as it marks the point when the external system would witness the effects of the reconfiguration, for instance admitting a new task. The server must also generate the `ReConfFin` signal at the earliest, as this marks the transition in the service guarantee of the server and the point when a new reconfiguration can be started. In other words, intervals $[t_R, t_A]$ and $[t_R, t_F]$ must be minimized.

Overhead: The reconfigurable server should have an efficient implementation: the additional overhead for reconfiguration both in terms of computation time and in terms of storage space must be small.

System-wide reconfiguration: The reconfiguration of the server must be suited to effect, simultaneously, the reconfiguration of several servers.

V. R-CBS

In this section, we present a server algorithm R-CBS which is a modified CBS that can perform reconfigurations. We show that the algorithm satisfies the guarantees established in the last section.

1) Initial conditions, i.e., at time $t = 0$:
   a) The task queue is not in the EDF queue,
   b) $q := 0$, $d := 0$, $r := 0$, $\sigma := 0$, $\Theta := 0$.
2) When `TQNonEmpty` is raised and $\Theta = 0$:
   a) At time t, if $q \geq (d - t) \times U$, then
      i) $q := Q$, $d := t + P$,
      ii) $\sigma := 0$, $\tau := t$.
   b) If $r < t$, then
      i) $r := t$.
3) When `TQRemoved` is raised and $\Theta = 0$:
   a) If $q = 0$, then
      i) (Soft CBS) $r := t$,
      (Hard CBS) $r := d$.
      ii) $q := Q$, $d := d + P$.
4) When `ReConfReq` is raised:
   a) $t_R := t$.
   b) $\Theta := 1$.
   c) $v := t + \dfrac{\max(0, (\sigma - (t - \tau)U)}{(\max(U, U'))}$.
   d) (Soft CBS) $r := t$
   (Hard CBS) $r := v$.
   e) If $(U' \geq U)$ then
      i) $t_A = t$.
      Else
      ii) $t_A = v$.
   f) Signal `ReConfAck` is raised at time t_A.
   g) If $(v > t)$, then
      i) $d := \inf_u\{u \geq v : \beta_{\min}(u - \tau) > \sigma\}$
      ii) $q := (d - v) \cdot U'$.
      Else
      iii) $q := q + ((d - t) \cdot (U' - U))$.
5) When `TQNonEmpty` is raised, and $\Theta = 1$:
   a) If $\sigma \leq \big((t_R - \tau) \cdot U + (t_A - t_R) \cdot \max(U, U') + (t - t_A) \cdot U'\big)$, then
      i) $\Theta := 0$,
      ii) $(Q, P) := (Q', P')$,
      iii) $\sigma := 0$, $\tau = t$,
      iv) Signal `ReConfFin` is raised,
      v) $q := Q$, $d := t + P$.
   b) If $r < t$, then
      i) $r := t$.
6) When `TQRemoved` is raised, and $\Theta = 1$:
   a) If $q = 0$, then
      i) (Soft CBS) $r := t$
      (Hard CBS) $r := d$.
      ii) $d' := \inf_u\{u \geq v : \beta_{\min}(u - \tau) > \sigma\}$
      iii) $q := (d' - d) \cdot U'$.
      iv) $d := d'$.

Figure 4. Algorithm of R-CBS

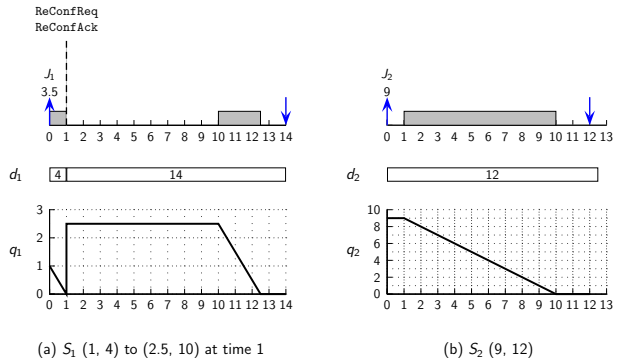


Figure 5. No isolation violation in Example 1 (Figure 1) when using R-CBS

A. Algorithm

To enable reconfiguration, R-CBS needs to maintain and update details not considered in the CBS. To do this, we introduce three new server variables, namely τ, σ and Θ. The variable τ denotes the latest time, on or before the current time, when steps 2(a)(i-ii) of the server algorithm (Figure 3) are executed. The variable σ denotes the amount of execution received by the task queue in the time interval $[\tau, t]$, where t is the current time. Θ is a binary variable that is 0 if the server is not reconfiguring and 1 if it is reconfiguring. The EDF scheduler has two additional roles in the setup. It increments the value of σ along with decrementing the remaining budget q of the task queue. Also when `ReConfReq` is raised, it removes the task queue from the EDF queue. The algorithm for both soft and hard variants is presented in Figure 4.

Notice that when reconfiguration is not active, i.e., when $\Theta = 0$, R-CBS is the same as the original CBS algorithm presented in Section III, with the minor difference of maintaining the variables σ and τ. When $\Theta = 1$, the server algorithm responds to the signals `TQNonEmpty` and `TQRemoved`, differently.

B. Examples from Section II

Now we revisit the examples discussed in Section II, and demonstrate how the R-CBS performs the reconfiguration, correctly.

Consider Example 1 (Figure 1), where we demonstrated that isolation violation occurs due to the immediate reconfiguration of the system. R-CBS, as expected, waits until the end of the current server period, before making the switch to the new parameters. This is shown in Figure 5.

For Example 2 (Figure 2), we had a service guarantee violation even when the server waited till the end of the server period before switching to the new set of parameters. However, when using R-CBS, at time $t = 15$, the server S_3 chooses a deadline of 25 and adds an additional budget of 4, according to item 4 of the algorithm. This indeed results in no service guarantee violation.

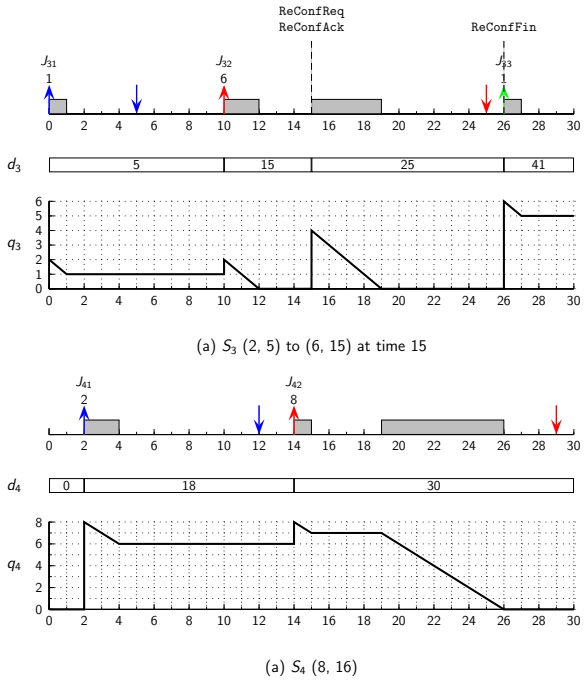


Figure 6. No deadline violation in Example 2 (Figure 2) when using R-CBS.

C. Guarantees of R-CBS

We now prove that R-CBS satisfies the guarantees of Definitions 1 and 2.

Theorem 4. *R-CBS satisfies the schedulability guarantee presented in Definition 1.*

Proof: The utilization claimed by the server is given by $q/(d-t)$ at times t when the server parameters are changed. When $\Theta = 0$, i.e., $t < t_R$, this is simply given by U (items 2(a)(i), 3(a)(ii)). For $\Theta = 1$, for $t > t_A$, this is given by U' (items 4(g)(ii), 6(a)(iii)). The only case when $t_A > t_R$ is when $U' < U$ and $v > t_R$. For this case, v is set in item 4(c) such that in $[t_R, t_A]$, the reserved utilization is U. The most interesting case is that of item 4(g)(iii), where the reserved utilization can be larger than $\max(U, U')$. Two subcases arise depending on the state of the task queue at time t_R. If at time t_R, the task queue is empty, then `TQNonEmpty` would be raised at some later time and the condition of item 5(a) would be true and the system would be reset to work in the new mode. Let at time t_R the task queue is not empty. This implies that the server has been contending for resource but has not been lagging behind its utilization. Then at time t_R the server has an additional promised execution time of $q - (d-t) \cdot U$. From t_R until the deadline at d the server will be promised another $(d-t) \cdot U'$ amount of execution time. Adding these two components we have the remaining budget as set in 4(g)(iii). ■

Theorem 5. *R-CBS satisfies the service guarantee presented in Definition 2.*

Proof: To prove the service guarantee, given any input arrival function R and any time t, we need to find a time $s \leq t$, such that

$$R'(t) - R(s) \geq \beta_{\min}(t-s). \tag{8}$$

For time t when $\Theta = 0$, the value of s is simply equal to the value of τ at t. From the results of [13], this satisfies the above constraint. For time t when $\Theta = 1$, we show that s can again be set equal to the value of τ at t and satisfy the above constraint. Note that at time τ a new job arrives while the task queue is empty, and thus $R(\tau) = R'(\tau)$. Thus to show (8) it is sufficient to show that the total execution provided by the server in $[\tau, t]$ is not lesser than $\beta_{\min}(t-\tau)$. Also note that since we have shown the schedulability of the system, any reservation requested by the server would be provided on or before the specified deadline.

First consider the case when $t_A > t_R$ and $t \in (t_R, t_A]$. For this case, it is easy to see that from item 4(c) that $\sigma = (t_A - \tau) \cdot U$. Note that $v = t_A$. Thus, for all $t \in (t_R, t_A]$, $s = \tau$ satisfies (8).

Now consider the case of $t > t_A$. Here the updates to the budget and deadline are performed in items 4(g)(i-iii) and 6(a)(ii-iii). In the case of items 4(g)(i-ii) and 6(a)(ii-iii), the deadline is set explicitly such that (8) is satisfied for $s = \tau$. Again, item 4(g)(iii) is the most interesting case. Let $t \in (t_R, d]$, where d is the deadline of the server at time t_R. For any $t < d$, (8) holds because until the previous server period, the server behaved like a standard CBS. For $t = d$, we have $R'(t) - R'(\tau) = (t_R - \tau) \cdot U + (t - t_R) \cdot U'$. This can be adapted to show that (8) holds for $s = \tau$. ■

D. Other properties

Signal `ReConfAck` *cannot be generated any earlier:* As discussed earlier, the signal `ReConfAck` marks the point when the rest of the system can respond to the reconfiguration of a server. In our algorithm, the signal `ReConfAck` is raised immediately at time t_R for every case except when $U' < U$ and $\sigma > (t - \tau)U$. The second condition is evaluated at time t_R, and it states that at time t_R the server has executed jobs for larger than the fraction of utilization reserved for it. In other words, it has executed its share of the reservation in the future. As a result the server cannot immediately reduce its utilization and provide it to the rest of the system. The value of v computed in 4(c) determines the earliest time when the server can reduce its utilization while providing the service guarantee of Definition 1.

Signal `ReConfFin` *cannot be generated any earlier:* As discussed earlier, the signal `ReConfFin` marks the point when the server completely transitions to the new configuration. A complete transition to the new mode implies that the values such as σ and τ which characterize the past of the server can be reset. The earliest time when this can be done,

is while the task queue is empty and the server has executed for not more than its reservation since τ. Since, the server is called only when either `TQNonEmpty` or `TQRemoved` is raised, the earliest such time is indeed when the condition of 5(a) is satisfied. Note that this condition depends on the input arrival stream. A stream that continuously keeps the server in a busy window would lengthen the delay in generating the `ReConfFin` signal.

Overhead: Space overhead is negligible with only three extra scalar server variables. Timing overhead arises due to (a) call of item 4 when `ReConfReq` is called and (b) the larger complexity of item 6(ii) in comparison to item 3(ii). When `ReConfReq` is called, the server executes item 4 which has only a constant-time overhead with few simple computations. The higher complexity of item 6(ii) is minimized by ensuring the system is prompt, as we discuss below.

Simultaneous reconfiguration of several servers: In a system, it is likely that a reconfiguration involves several servers. By defining the correctness of the server reconfiguration in terms of the service and isolation guarantees we have enabled modularity such that servers can reconfigure simultaneously as long as the overall system schedulability is guaranteed at all points of time. We discuss with an example in Section VI-B.

E. Discussion

We briefly discuss some noteworthy points about the two guarantees of R-CBS. With regards to isolation guarantee, the time t_A is significant in the way Definition 1 is formulated. If the utilization of the reconfiguring server is increasing, then the scheduler must ensure that the difference in utilization is available at t_R and thus t_A is trivially equal to t_R. When the utilization of the server is decreasing, t_R again equals t_A if the server has been under-utilizing its share of the resource at t_R. Otherwise, the server will generate the `ReConfAck`signal after a delay in order to first regain the difference in utilization that is due. This is similar in spirit to results in [5] and [6], where the changes to EDF schedulers are considered. Indeed most existing research has focussed on isolation guarantee. One interesting question is: When is the value of $(t_A - t_R)$ large? This would be the case if the relative deadline of the server is large, which may happen if either the server has a larger period P or the server is a soft-variant and has consumed more than its share of resource at the price of increasing its deadline. Thus, while soft-CBS allows the system to be workload conserving, it can potentially delay reconfiguration of the server even indefinitely.

The service guarantee, is equally, if not more interesting. We motivated the use of servers in multi-moded systems to allow for variability and imprecise models in the application. For any input arrival trace of the application, R, Definition 2 gives a service guarantee as a lower bound on the output

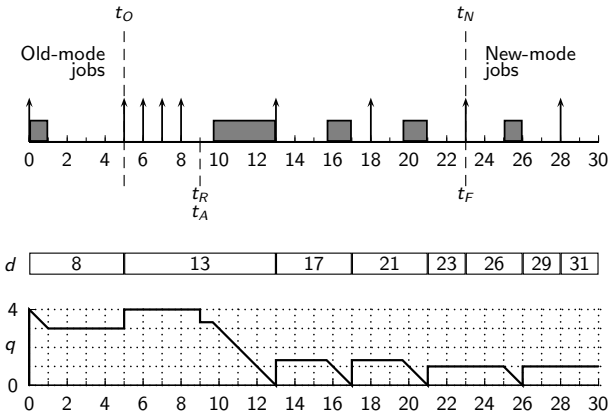


Figure 7. Relating server reconfiguration to job modes

arrival trace. The price of this generality is that the server reconfiguration algorithm must dynamically adapt to the input arrival stream to provide the service guarantee. In principle, this seems to be computationally difficult as the condition (6) of Definition 2, requires that for every time t there must exist an earlier time s, such that (6) holds. Potentially this requires that the server algorithm maintains a log of the history of the trace and its execution, and compute upon it. However, in R-CBS this is decidedly simple: the value of s for any time t is the value of the server variable τ at time t. In addition, we only need to store the amount of execution received since τ in the variable σ. This ease of logging in providing the service guarantee is the key to the simplicity of R-CBS.

As mentioned, earlier works considered old-mode and new-mode jobs. In the context of reconfiguring servers, one may ask which jobs experience service under (a) the old configuration, and (b) the new configuration? Let t_O denote the time of arrival of the first job that is not served by the old configuration and t_N denote the time of arrival of the first job executing under the new configuration. Consider an example of a task with period 5 and execution time 1, with maximum jitter of 4, experienced by a maximum of 4 consecutive jobs. This task is served by a R-CBS with parameters (4, 8), and at time $t_R = 9$ it is to be reconfigured to the new parameters (1, 3). The system behavior for a specific trace is plotted in Figure 7. While t_N is equal to t_F, t_O is earlier than t_A. Can we compute a bound on the value of t_O? To this end, we need to characterize the input arrival rate α for the possible input arrival traces R as

$$\alpha(\Delta) = \max_t (R(t + \Delta) - R(t)), t, \Delta \geq 0. \tag{9}$$

Then, the value of t_O is bounded as

$$t_O \geq t_R - d_{\max}, \tag{10}$$

where $d_{\max}$ is the worst-case delay in the old mode

$$d_{\max} = \sup_{t \geq 0} \{\inf\{\Delta \geq 0 : \ \alpha(t-\Delta) \leq \beta(Q,P,t)\}\}.$$

Indeed, in the above example, $d_{\max} = 4$ and $t_O = t_R - 4$. This worst-case value of t_O is experienced when the stream of jobs has a burst just before the reconfiguration.

VI. Experimental Results

A. Comparison to TDMA server

As mentioned earlier, Stoimenov et al. [11] consider the problem of TDMA server reconfiguration. Similar to our approach, the authors of [11] define a notion of correctness of the reconfiguration and then detail a correct reconfiguration algorithm. The similarity of approach motivates a close comparison. To this end, we consider Example 1 from [11]. For the detailed problem setting and the solution of TDMA reconfiguration, we refer the reader to [11]. It has been concluded that the reconfiguration of the TDMA servers cannot be done immediately. In fact, for a correct reconfiguration in this example, $K = 3$ periods of accelerated execution are necessary, wherein the TDMA server consumes more utilization that it is allowed to.

We can replace the TDMA servers by R-CBS, where the period equals the time-wheel period and the maximum budget equals the slot-length. For the same trace of job arrivals as in Example 1 of [11], we plot the behavior of the system with R-CBS in Figure 8. To aid brevity, we have dropped `ReConf` from the signal names. Firstly, the reconfiguration is correctly performed without the need for consuming utilization that is not reserved for the servers. Secondly, the reconfiguration of the servers completes (`Fin` signals are generated) by time 35 which is earlier than in the case of TDMA servers. Thirdly, the analysis technique and server algorithm for R-CBS is computationally more efficient than that of the reconfiguration of TDMA servers.

It is instructive to understand the reasons for the above differences. The underlying difference between the two server mechanisms is that TDMA is static, time-triggered, while CBS adapts to job arrivals. Thus, the service guarantees provided by the TDMA have to be independent of the input

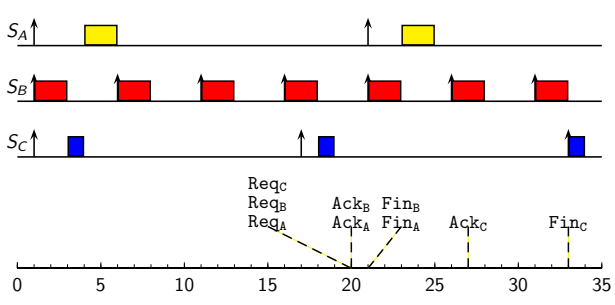


Figure 8. Example 1 from [11]: S_A (1,10) to (3,12), S_B (5,10) to (6,12), S_C (1,10) to (1,12)

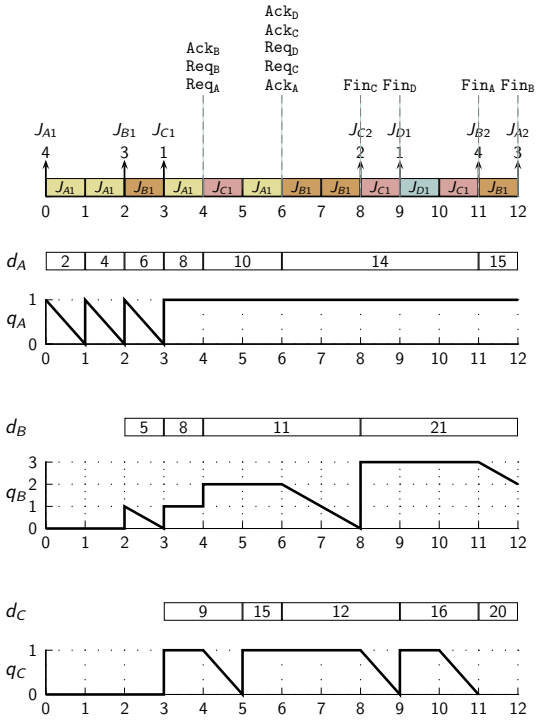


Figure 9. Multiple servers reconfiguring simultaneously

arrival function, unlike Definition 2. To cope with this, the authors of [11] use the strict service function [14] and the convolution operator to provide the service guarantee across the reconfiguration. As a consequence, for some settings of reconfigurations, statically computed number of accelerated execution periods are required, which are not required in the case of R-CBS. On the other hand, R-CBS performs the reconfiguration dynamically in response to the input stream which is efficiently logged in variables τ and σ.

B. Reconfiguration of several servers

In the second case study, we will illustrate, more elaborately, how several CBS can be reconfigured simultaneously. Consider a system with three servers S_A, S_B and S_C with parameters (1,2), (1,3) and (1,6) respectively. The system is requested to reconfigure at time 4. In the new configuration the S_A reduces its utilization with parameters (1,4), S_B retains its utilization with new parameters (3, 9), and S_C increases its utilization with parameters (1,4). Additionally, a new server S_D is inserted with parameters (1,6).

The evolution of the system in response to a specific trace of jobs is shown in Figure 9. We have again dropped `ReConf` from the signal names. At time 4, the system starts the reconfiguration. However, `Req` can be generated only for servers S_A and S_B as the additional utilization required for the reconfiguration of S_C and insertion of S_D is not available. Server S_A has been executing tasks from 0 to 3 and thus has consumed beyond its reserved budget. To

generate the `Ack` signal it has to wait until time 6. S_B, on the other hand, only retains its utilization and thus can generate the `Ack` signal immediately. At time 6, when S_A generates its `Ack` signal, the system uses the additional utilization to reconfigure and create S_C and S_D, respectively. The `Ack` signals of these servers are created immediately. The `Fin` signals are then generated and the reconfiguration concludes.

This example illustrates that the defined service and isolation guarantees of R-CBS enable each server to operate independent of the behavior of the rest of the system. The role of a central controller to orchestrate this reconfiguration is limited only to generating the `Req` signal when the spare utilization needed by the server is available. This can be a significant advantage in systems with hierarchical schedulers.

VII. Conclusions and Future Work

The ability to reconfigure resource reservations of servers during run-time while guaranteeing timing properties is a motivating challenge. We showed that straightforward approaches to such reconfiguration of the constant bandwidth server (CBS) violate some expected behavior. We characterized these expectations quantitatively in terms of the isolation and service guarantees. We then presented the server algorithm, R-CBS, that is capable of performing reconfiguring the standard CBS. A noteworthy point was that the R-CBS maintains some state of the system (denoted as τ and σ) to be able to reconfigure correctly and efficiently. The comparison with TDMA server reconfiguration highlighted that a dynamic server is more efficient in terms of the utilization requirements and the complexity of the reconfiguration algorithm. Also, the modularity of the reconfiguration guarantees enabled several reconfiguration of multiple R-CBS with minimal central control to reconfigure simultaneously.

There are two directions of future work. One is to consider more sophisticated mechanisms such as resource reclamation while performing the reconfiguration more efficiently. For instance, in Figure 6, S_A executes during [5, 6] though the servers S_C and S_D await for their `ReConfReq` signals. Instead, one may *reclaim* this available utilization to quicken the reconfiguration of S_C and S_D. The second direction is to optimize the reconfiguration of a set of servers with the objective of minimizing the time to reconfigure the entire system. Though this breaks the spirit of modularity, it might have practical significance in systems which have to reconfigure often and would like to minimize the system-wide reconfiguration time. For instance, in Figure 6, if the spare utilization was not sufficient at time 6, for the reconfiguration of S_C and creation of S_D, then we would have a *choice* of which reconfiguration to do. An efficient optimization step that considers the state of the servers would be an interesting goal.

VIII. Acknowledgements

This work was supported by the EU FP7 projects with grant numbers 248776 and 288175.

References

[1] K. Tindell, A. Burns, and A. J. Wellings, "Mode changes in priority pre-emptively shceduled systems," in *In Proc. Real Time Systems Symposium*, pp. 100–109, 1992.

[2] K. Tindell and A. Alonso, "A very simple protocol for mode changes in priority preemptive systems," tech. rep., Universidad Politecnica de Madrid, 1996.

[3] J. Real and A. Crespo, "Mode change protocols for real-time systems: A survey and a new proposal," *Real-Time Systems*, vol. 26, no. 2, pp. 161–197, 2004.

[4] G. C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni, "Elastic scheduling for flexible workload management," *IEEE Trans. Computers*, vol. 51, no. 3, pp. 289–302, 2002.

[5] Q. Guangming, "An earlier time for inserting and/or accelerating tasks," *Real-Time Systems*, vol. 41, no. 3, pp. 181–194, 2009.

[6] 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)*, pp. 396–407, 2003.

[7] S. Craciunas, C. Kirsch, H. Payer, H. Rock, and A. Sokolova, "Programmable temporal isolation through variable-bandwidth servers," in *SIES*, pp. 171 –180, 2009.

[8] L. Abeni and G. C. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *IEEE Real-Time Systems Symposium*, pp. 4–13, 1998.

[9] L. Santinelli, G. C. Buttazzo, and E. Bini, "Multi-moded resource reservations," in *RTAS*, 2011.

[10] N. Fisher and M. Ahmed, "Tractable real-time schedulability analysis for mode changes under temporal isolation," in *ESTImedia*, pp. 130–139, IEEE, 2011.

[11] N. Stoimenov, L. Thiele, L. Santinelli, and G. C. Buttazzo, "Resource adaptations with servers for hard real-time systems," in *EMSOFT* (L. P. Carloni and S. Tripakis, eds.), pp. 269–278, ACM, 2010.

[12] P. Kumar, J.-J. Chen, and L. Thiele, "Demand bound server: Generalized resource reservation for hard real-time systems," in *In Proc. of the 11th International. Conference on Embedded software, EMSOFT*, 2011.

[13] P. Kumar, J.-J. Chen, L. Thiele, A. Schranzhofer, and G. Buttazzo, "Real-time analysis of servers for general job arrivals," in *In Proc. of the 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Algorithms, RTCSA*, 2011.

[14] J.-Y. L. Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, vol. 2050 of *Lecture Notes in Computer Science*. Springer, 2001.