Resource Reservation for Mixed Criticality Systems

Giuseppe Lipari[1]*, Giorgio C. Buttazzo[2]

[1] LSV, ENS - Cachan, France
[2] Scuola Superiore Sant'Anna, Italy

Abstract. This paper presents a reservation-based approach to schedule mixed criticality systems in a way that guarantees the schedulability of high-criticality tasks independently of the behaviour of low-criticality tasks. Two key ideas are presented: first, to reduce the system uncertainty and advance the time at which a high-criticality task reveals its actual execution time, the initial portion of its code is handled by a dedicated server with a bandwidth reserved for the worst-case, but with a shorter deadline; second, to avoid the pessimism related to off-line budget allocation, an efficient reclaiming mechanism, namely the GRUB algorithm [6], is used to exploit the budget left by high-criticality tasks in favor of those low-criticality tasks that can still complete within their deadline.

1 Introduction

With the progress of computer architectures, embedded computing systems are required to execute more and more concurrent activities on the same hardware platform. In mission-critical systems, computational activities may have different levels of criticality, and therefore different guarantee requirements imposed by certification authorities. In particular, more critical tasks are required to have more conservative estimations for their computational requirements, with respect to less critical activities. Such more conservative estimations increase system predictability by over allocating computational resources to more critical tasks, but also decrease the overall efficiency.

To partially compensate for such pessimistic estimations, Vestal [16] proposed a new task model where each task can be specified with different levels of criticality, each characterised by a different computation time estimate, depending on the criticality level: the higher the criticality level, the higher the computation time estimate. Slightly different models have been also proposed by other authors.

In this paper, we consider a system with two criticality levels that must execute a task set Γ of n periodic or sporadic tasks. Each task τ_i is characterised

* The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 246556.

by a criticality level X_i, which can be either high (HI) or low (LO), a worst-case computation time (WCET) C_i (which depends on its criticality level), a period (or minimum interarrival time) T_i, and a relative deadline D_i. Tasks with low criticality, denoted as LO-tasks (τ_i^{LO}), have a single WCET estimate C_i, whereas tasks with high criticality, denoted as HI-tasks (τ_i^{HI}), have a normal WCET estimate C_i and a more conservative one, C_i^{ov}, to take overruns into account, where $C_i^{ov} > C_i$. Each task generates an infinite sequence of jobs, $\tau_{i,1}$, $\tau_{i,2}$, ..., where each job $\tau_{i,j}$ is characterised by a release time $r_{i,j}$, a computation time $c_{i,j}$, and an absolute deadline $d_{i,j}$. The actual computation time requested by a job $\tau_{i,j}$ is denoted by $e_{i,j}$.

According to such a model, the task set Γ is partitioned in two subsets Γ^{LO} and Γ^{HI} and the mixed criticality (MC) feasibility problem is formulated as follows:

Definition 1. *A task set Γ is MC-feasible if and only if both the following conditions are verified:*

1. *If all HI-tasks execute for no more than their optimistic computation time C_i, then there exists a schedule where all tasks in Γ complete within their deadline.*
2. *If one or more HI-tasks exceed their optimistic computation time C_i (but not their conservative estimate C_i^{ov}), then there exists a schedule where all tasks in Γ^{HI} complete within their deadline.*

The problem of optimally scheduling such mixed-criticality systems has been shown to be highly intractable even under very simple system models [2]. The complexity comes from the fact that optimal scheduling decisions depend on the knowledge of the actual tasks execution times, which are not known off-line, but will be available only when a task completes or exceeds its optimistic estimate. Given such a dependency of scheduling decisions from future knowledge, it is clear that optimality can only be achieved by an ideal clairvoyant scheduler. To better clarify this issue, consider the task set reported in Table 1.

	X_i	C_i	C_i^{ov}	T_i	D_i
τ_1	LO	3	3	8	4
τ_2	HI	2	4	8	6

Table 1. Sample mixed criticality task set.

As illustrated in Figure 1, the task set is MC-feasible, since both conditions stated in Definition 1 are verified.

Nevertheless, Figure 2 illustrates that no online algorithm can guarantee the MC-schedulability of the task set, because for each decision taken at time $t = 0$ (to schedule τ_1 or τ_2), there exists a situation in which a task misses its deadline. This example shows that the correct decision that produces an MC-feasible schedule can be taken only by a clairvoyant scheduler that knows (at time $t = 0$) how much task τ_2 will execute.

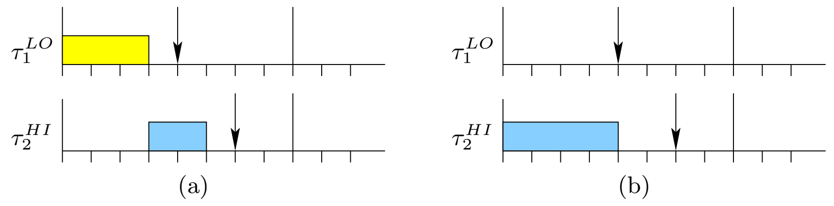

Fig. 1. If τ_2 does not exceed C_1, Γ is feasible (a), and if τ_2 executes for C_2^{ov}, then Γ^{HI} is feasible.

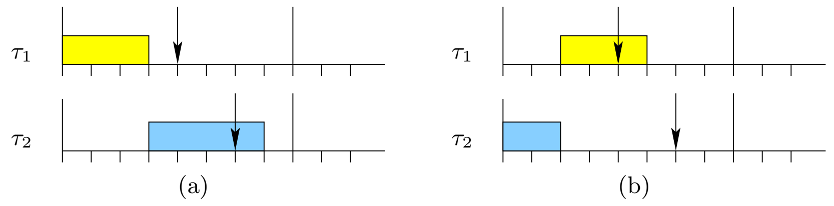

Fig. 2. If τ_1 is scheduled at $t = 0$, τ_2 can miss its deadline (a); and if τ_2 is scheduled at $t = 0$, τ_1 will miss its deadline (b).

Contribution In this paper we propose a reservation-based approach to schedule mixed criticality systems in a way that guarantees the schedulability of HI-tasks independently of the behavior of LO-tasks. Two key ideas are presented: first, to reduce the system uncertainty and advance the time at which a HI-task reveals its actual execution time, the initial portion of its code is handled by a dedicated server with a bandwidth reserved for the worst-case, but with a shorter deadline; second, to avoid the pessimism related to off-line budget allocation, an efficient reclaiming mechanism, namely the GRUB algorithm [6], is used to exploit the budget left by HI-tasks in favor of those LO-tasks that can still complete within their deadline.

Paper structure The rest of the paper is organized as follows. Section 2 formally presents the general approach. Section 3 presents the details of the reservation server. Section 4 illustrates the simulation results obtained with the proposed approach. Section 5 presents some related work. Section 6 concludes the paper and presents some future work.

2 General approach

We consider a reservation-based real-time system where each task (or group of tasks) can be assigned a reservation server that allocates a budget Q_i every period P_i. To better exploit the available computational resources, we assume that all the servers are scheduled by the Earliest Deadline First (EDF) scheduling algorithm [11].

Since HI-tasks must be guaranteed under all operating conditions and we cannot know in advance how much they will execute, each HI-task τ_i^{HI} is assigned a dedicated reservation server (HI-server) with bandwidth α_i^{HI} sufficient to satisfy its more conservative execution time C_i^{ov}. Therefore, each HI-task is handled by a periodic reservation (Q_i, P_i), where $Q_i = C_i^{ov}$ and $P_i = T_i$, thus having a bandwidth

$$\alpha_i^{HI} = \frac{C_i^{ov}}{T_i}. \tag{1}$$

The bandwidth remaining for serving all the LO-tasks is then:

$$\alpha^{LO} = 1 - \sum_{\tau_i \in \Gamma^{HI}} \alpha_i^{HI}. \tag{2}$$

Let U^{LO} be the total bandwidth required by the LO-tasks, that is,

$$U^{LO} = \sum_{\tau_i \in \Gamma^{LO}} \frac{C_i}{T_i}. \tag{3}$$

Note that, if $U^{LO} \leq \alpha^{LO}$, then there is enough bandwidth to complete all LO-tasks within their deadline, even when the system runs in high-criticality mode, hence the problem is trivially solved. In this paper, we consider the more interesting case where $U^{LO} > \alpha^{LO}$, so that not all LO-tasks can be guaranteed to complete before their deadlines in all modes. Nevertheless, when the system runs in low-criticality mode, that is, when all the HI-tasks do not exceed their optimistic estimate C_i, we propose to use a reclaiming mechanism for distributing the spare bandwidth saved by HI-task to LO-tasks. In particular, whenever a HI-job $\tau_{i,j}^{HI}$ completes before its optimistic estimate $(e_{i,j} < C_i)$, the following bandwidth can be reclaimed:

$$U_{i,j}^{rec} = \frac{C_i^{ov} - e_{i,j}}{T_i}. \tag{4}$$

When the system switches to high-criticality mode, there are two ways of dealing with LO-tasks: they may be dropped, as proposed in most of the previous research on mixed-criticality scheduling, or they can continue executing as soft real-time tasks, without interfering with the HI-tasks. In this paper, we adopt this second approach.

To schedule LO-tasks, we can follow two alternative approaches:

1. All LO-tasks are assigned a single reservation server (LO-server) with bandwidth α^{LO}.
2. Each LO-task is assigned a different server such that the sum of the bandwidths of these servers does not exceed α^{LO}.

In this paper, these two approaches are compared to see whether any one dominates the other.

3 Server mechanism

As discussed in the previous section, we assign each HI-task a dedicated server with bandwidth α_i^{HI} sufficient to cover the largest computational requirements C_i^{ov} of the task. At the same time, as soon as each job of a HI-task τ_i completes, the remaining budget is reclaimed and assigned to the LO-tasks. To be able to reclaim such an extra budget in advance, we are interested in advancing the completion time of HI-tasks as soon as possible. To achieve this goal, we use a technique similar to the EDF-VD algorithm, proposed in [4,3]. This technique consists in using two different deadlines and budgets for a HI-task, depending on whether it completes before or after its normal worst-case execution time C_i. The following section describes how to modify the GRUB server to achieve this objective.

3.1 High-criticality servers

A HI-server for a HI-task τ_i is defined as a tuple (Q_i, Q_i^{ov}, P_i), where $Q_i = C_i$, $Q_i^{ov} = C_i^{ov}$, and $P_i = T_i$. The server is assigned a bandwidth $\alpha_i^{HI} = Q_i^{ov}/P_i$ sufficient to guarantee the more conservative execution time. At run time, the server manages a capacity q_i, a *virtual time* v_i, a scheduling deadline d_i, and a criticality mode γ_i. Moreover, the server has an internal *state* which can be IDLE, READY, EXECUTING, RECHARGING and RELEASING. The server is initially in the IDLE state and its criticality mode is $\gamma_i = LO$. The state diagram for the server is shown in Figure 3.

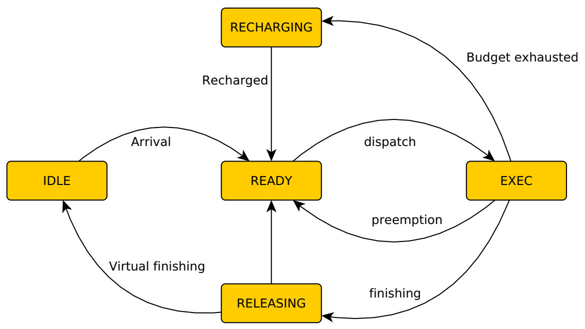

Fig. 3. State diagram for the HI-Server.

A server that is not IDLE is said to be *active*. Let $\mathcal{A}$ be the set of active servers. The algorithm maintains a global variable

$$U^{act} = \sum_{S_i \in \mathcal{A}} \alpha_i^{HI}. \tag{5}$$

The server uses the following rules:

1. When a HI-task is activated at time t, the server moves from the IDLE to the READY state. Correspondingly, its budget is replenished at $q_i = Q_i$ and its deadline is set at:

   $$d_i = t + \frac{Q_i}{Q_i^{ov}} P_i = t + \frac{Q_i}{\alpha_i^{HI}} = t + \frac{q_i}{\alpha_i^{HI}}.$$

   Note that such a deadline is shorter than the usual server deadline $(t + P_i)$. Moreover, the capacity $(Q_i = C_i)$ is also less than the maximum one $(Q_i^{ov} = C_i^{ov})$. However, the server bandwidth is still α_i^{HI}.
   Also, the server moves to the active set $\mathcal{A}$, therefore the value of U^{act} is updated to $U^{act} + \alpha_i^{HI}$.
2. When in READY, the HI-server with the earliest scheduling deadline is executed and moves to EXECUTING.
3. When in EXECUTING, the server capacity is decreased as:

   $$dq_i = -U^{act} dt$$

   If the system is fully utilised and all servers are active $(U^{act} = 1)$, this translates in reducing the capacity of the server at unit rate. When the system is not fully utilized, or when some server is IDLE, the capacity is decreased as a lower rate, so that the server can actually *reclaim* the free system bandwidth.
4. If, while in EXECUTING, the server is preempted by another server with earlier deadline, it moves back to READY, and its capacity is not decremented anymore.
5. If, while in EXECUTING, the capacity is exhausted, then the server behaves according to the values of the criticality mode:
   (a) If the criticality mode $\gamma_i = LO$, then the capacity is immediately recharged to $q_i = Q_i^{ov} - Q_i$, and the deadline is postponed to

   $$d_i \leftarrow d_i + \frac{Q_i^{ov} - Q_i}{\alpha_i^{HI}} = d_i + \frac{q_i}{\alpha_i^{HI}}.$$

   (Notice that this corresponds to the deadline of the original HI-task). Also, the criticality level is raised to $\gamma_i \leftarrow HI$.
   (b) If the criticality mode is $\gamma_i = HI$, a system exception is raised, as a HI-task should never exceed its budget Q_i^{ov}.
6. If, while in EXECUTING, the task completes the execution of the current job, the server moves to the RELEASING state. Correspondingly, the *virtual time* is computed for the server as follows:

   $$v_i \leftarrow d_i - \frac{q_i}{\alpha_i^{HI}}$$

7. The server remains in state RELEASING until $t \geq v_i$. At that point, the server moves to IDLE state and the criticality level is set to $\gamma_i \leftarrow LO$. If $t > v_i$, then an extra capacity of $\alpha_i^{HI}(v_i - t)$ is *donated* to the first server in the ready queue. Also, the server is removed from the set of active servers $\mathcal{A}$, and the overall utilization is decreased to $U^{act} \leftarrow U^{act} - \alpha_i^{HI}$.

3.2 Low-criticality servers

A similar algorithm is used for implementing the LO-server for serving LO-tasks, with a few modifications to the rules. A LO-Server is defined by only two parameters, the budget Q_i and the period P_i. As stated in Section 2, the LO-server is assigned a bandwidth α_i^{LO} given by Equation (2). At run-time, the server manages a capacity q_i, a scheduling deadline d_i, and has a state. The state diagram for a LO-server is the same as a HI-server, and is shown in Figure 3. The following rules change:

1 If the server is IDLE, when one of the LO-tasks served by the server is activated at time t, the server moves from IDLE to the READY state. Correspondingly, its budget is replenished at the value $q_i = Q_i$ and the server deadline is set at:
$$d_i = t + P_i = t + \frac{Q_i}{\alpha_i^{LO}} = t + \frac{q_i}{\alpha_i^{LO}}.$$
Notice that this is the same equation as the HI-server. Also, the server moves to the active set $\mathcal{A}$, therefore the value of U^{act} is updated to $U^{act} \leftarrow U^{act} + \alpha_i^{LO}$.

5 If, while in EXECUTING, the capacity is exhausted, then the server moves to the RECHARGING state and a recharging time is set at d_i. The server is suspended from execution until the capacity replenishment.

6 If, while in EXECUTING, the task completes the execution of the current job, and there are no more tasks in the server local queue, the server moves to the RELEASING state. Correspondingly, the *virtual time* is computed for the server as follows:
$$v_i \leftarrow d_i - \frac{q_i}{\alpha_i^{LO}}.$$

8 If, while the server is in RECHARGING state, $t = d_i$, then the server budget is replenished at $q_i \leftarrow Q_i$, the server deadline is postponed at $d_i \leftarrow d_i + P_i$, and the server is moved to the READY state.

9 If, while the server is in RELEASING state, a new LO-task is activated locally in the server, then the server moves to the READY state, with the same capacity and deadline.

3.3 An example

Consider a simple MC task set consisting of only two tasks: a LO-task $\tau_1 = (C_1 = 4, T_1 = 6)$ and a HI-task $\tau_2^{HI} = (C_2 = 2, C_2^{ov} = 4, T_2 = 8)$. To schedule this task set, we start by defining a HI-server $S_2 = (Q_2 = 2, Q_2^{ov} = 4, P_2 = 8)$; the server bandwidth is $\alpha_2^{HI} = 0.5$. The first question is: how much budget we can reserve for τ_1? Clearly, we cannot reserve $(Q_1 = 4, P_1 = 6)$, as there is not enough bandwidth left. Therefore, the LO-server is defined with a bandwidth $\alpha_1^{LO} = 1 - \alpha_2^{HI} = 0.5$, a period $P_1 = T_1 = 6$, and a budget $Q_1 = \alpha_1^{LO} P_1 = 3$. The schedule produced by the proposed algorithm is shown in Figure 4.

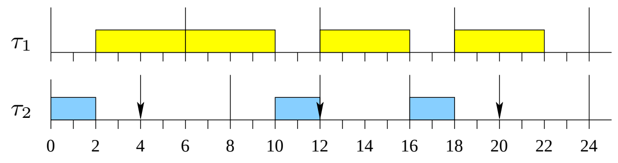

Fig. 4. Example of reclamation.

- At time 0, task τ_2 starts executing inside its server. Both servers are active (because they both arrive at time $t = 0$), so the capacity is updated at the following rate while it executes:

  $$\frac{dq_2}{dt} = -U^{act} = -1$$

  Therefore, when task τ_2 completes, the server has budget $q_2 = 0$. The server moves to state RELEASING and virtual time is computed as $v_2 = 4$. Therefore, the server remains active until time $t = 4$, and from then it becomes inactive until time $t = 8$.
- At time $t = 2$, task τ_1 starts executing. Its budget is initially $q_1 = 3$. Again, all servers are active (remember that Server S_2 will become inactive at time 4), hence capacity is decreased at unit rate.
- At time $t = 4$, the server S_2 becomes IDLE. Therefore, the capacity rate for task τ_1 changes. First of all, let us observe that at time $t = 4$ its value is $q_1 = 1$. The new rate is:

  $$\frac{dq_2}{dt} = -U^{act} = -0.5.$$

  This means that, at the current rate, the task can still execute for 2 units of time. That is exactly what we need to complete the task at time $t = 6$.
- At time $t = 6$, the first instance of task τ_1 completes, but the second one is activated. Therefore, the server recharges its budget at $q_1 = 3$, and continues to execute with deadline at $d_1 = 12$. Since the other server is still inactive, the rates for the virtual time and the deadline do not change.
- At time $t = 8$, task τ_2 is activated again and S_2 becomes active. The current value of the capacity is $q_1 = 2$. Suppose the scheduler decides to continue executing τ_1. Therefore, the new rates is now:

  $$\frac{dq_2}{dt} = -1$$

  And this means that τ_1 can execute for two more units of time, completing its executing at time $t = 10$.

In this example we have seen that τ_1 is able to complete execution of all its jobs within its deadline, even if the assigned budget is less than its execution requirements, thanks to the reclaiming mechanism.

However, it is still to be understood how much capacity can be reclaimed by *LO*-tasks. In fact, it is easy to build an example in which the *LO*-server does not receive enough extra capacity to complete all its tasks before their deadlines. For example, if the LO-tasks have very short relative deadlines compared to the HI-tasks, they will executed before them most of the times, and hence they will not be able to reclaim any capacity.

Computing the amount of capacity reclaimed by *LO*-tasks is a difficult and open problem that will be the subject of our future research. In this paper we just compare two methods for scheduling *LO*-tasks inside a *LO*-server: using a single server for serving all the *LO*-tasks; or using a dedicated *LO*-server for each *LO*-task.

4 Experimental results

To evaluate the performance of the reclamation algorithm presented in Section 3, we performed a set of simulation experiments. The server algorithms have been implemented in RTSim [12], a scheduling simulation tool for modeling real-time systems.

In each simulation run, we generated 4 HI-tasks and 4 LO-tasks. Computation times and the periods of the HI-tasks have been randomly selected to get a cumulative utilization equal to 50%:

$$\sum_{\tau_i \in \Gamma^{HI}} \frac{C^{ov}}{T_i} = 0.5.$$

In the first set of experiments, the periods of the HI-tasks have been chosen according to a uniform distribution in the range $[1000, 5000]$, in multiples of 100. In the second set of experiments, they were chosen in the interval $[6000, 10000]$.

Computation times in low-criticality mode were computed as a fixed fraction of the computation times in high-criticality mode:

$$\forall \tau_i \in \Gamma^{HI} \quad C_i = r \cdot C_i^{ov}$$

where r was varied between $[0.2, 0.75]$.

Computation times and periods of the LO-tasks were chosen to achieve a cumulative utilization equal to $\frac{0.5}{0.75}$. In this way, for all values of parameter r, we have:

$$\sum_{\tau_i^{HI}} \frac{C_i}{T_i} + \sum_{\tau_i^{LO}} \frac{C_i}{T_i} \leq 1.$$

In other words, we made sure that the system is never overloaded in low-criticality mode. In the first set of experiments, the periods of the LO-tasks were chosen in the range $[6000, 10000]$, whereas in the second set of experiments periods were chosen in $[1000, 5000]$.

For each HI-task we prepared a HI-server with bandwidth $\alpha_i^{HI} = C_i^{ov}/T_i$. For the LO-tasks, we made two different choices: assigning all LO-tasks to a single LO-server with utilization equal to 50% and period $P = 100$; or assign each LO-task to a different LO-server with bandwidth proportional to its computational requirements, scaled down so that the sum of the bandwidth assigned to the LO-server was 50%. In the single server case, LO-tasks inside the server were scheduled by the EDF local scheduler.

For each combination of parameters, we generated 30 different task sets with random periods and computation times. Each simulation was run for 10.000.000 units of simulation time. We only analysed the low-criticality mode, to test the effectiveness of the reclaiming algorithm in that condition. Therefore, HI-tasks never execute more than their optimistic computation time C_i. We measured the average number of deadlines missed and the tardiness of the LO-tasks. Simulation results are reported in the following sections.

4.1 First set of experiments

In this first set of experiments, the periods of the HI-tasks have been chosen to be all lower than the periods of the LO-tasks. Therefore, at least at the beginning of the schedule, LO-tasks execute after HI-tasks have been completed, and so they can immediately take advantage of the reclaimed bandwidth.

The deadline miss percentage of the LO-tasks is shown in Figure 5 for the case of separate servers, and single server. It is evident that the deadline miss percentage is very low in all cases. Even for the case of $r = 0.75$ (where the total system utilization is very close to 100%), it never goes above 5% of the total number of deadlines. Also notice that putting all LO-tasks in a single server is very effective, as we observed 0 deadlines missed in all the experiments.

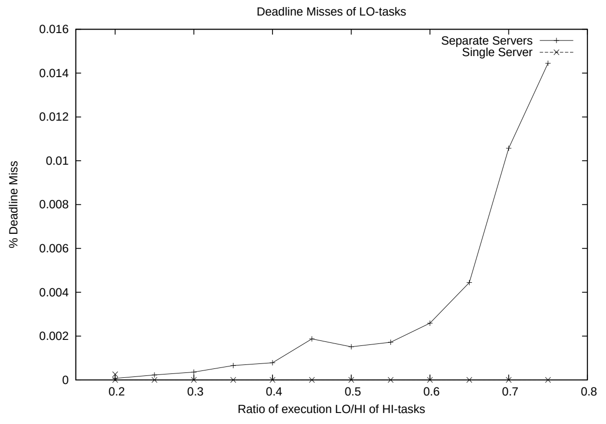

Fig. 5. Deadline miss percentage of LO-tasks, using dedicated servers, or a single cumulative server. The LO-tasks have larger periods than the HI-tasks.

A very similar trend can be observed for the tardiness reported in Figure 6. Such small values of the tardiness indicate that the reclaiming mechanism is very effective, even with very highly loaded systems.

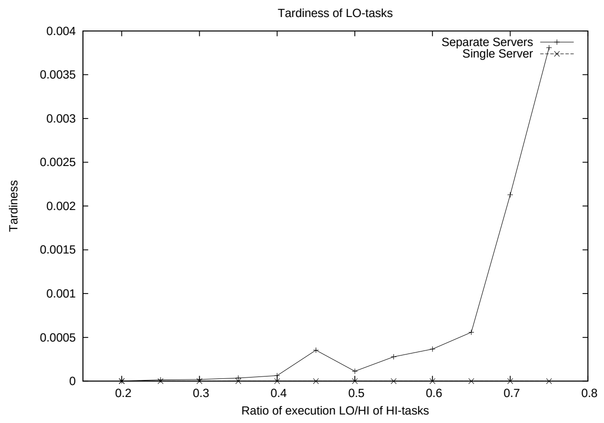

Fig. 6. Tardiness of LO-tasks, using dedicated servers or a single cumulative server. The LO-tasks have larger periods than the HI-tasks.

4.2 Second set of experiments

In this second set of experiments, the periods of the HI-tasks have been chosen to be all higher than the periods of the LO-tasks. Therefore, at least at the beginning of the schedule, LO-tasks execute before HI-tasks, so they cannot immediately take advantage of the reclaimed bandwidth.

The deadline miss percentage of the LO-tasks is shown in Figure 7 for the case of separate servers and single server.

Once again, a very similar trend can be observed for the tardiness in Figure 8. Notice that in this case we detected very small values of the tardiness for $r \geq 0.6$ due to a very small number of deadline misses (not visible in the graphs).

While these simulations cannot conclusively establish the theoretical performance guarantees for LO-tasks, they indicate that it is indeed worthwhile to perform further investigation on reclamation techniques for mixed criticality systems.

5 Related work

The problem of scheduling mixed criticality systems has been addressed by several authors under slightly different models and assumptions.

Lakshmanan et al. [5,8,9] proposed a slack-aware approach on top of fixed priority scheduling, providing a schedulability test that guarantees that all deadlines

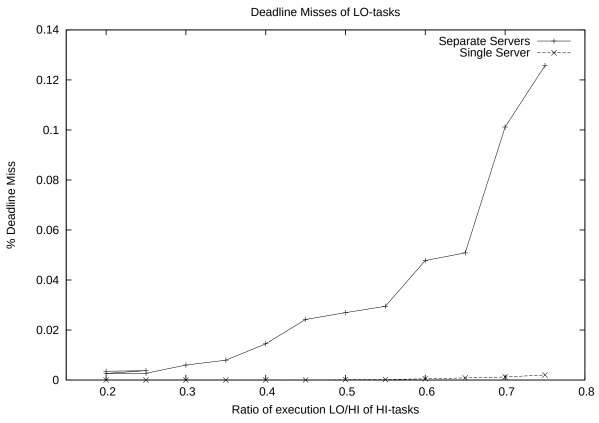

Fig. 7. Deadline miss percentage of LO-tasks, using dedicated servers, or a single cumulative server. The LO-tasks have smaller periods than the HI-tasks.

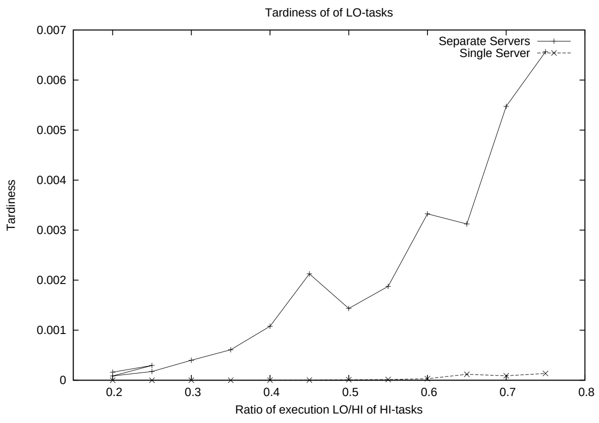

Fig. 8. Tardiness of LO-tasks, using dedicated servers or a single cumulative server. The LO-tasks have smaller periods than the HI-tasks.

of HI-tasks are met regardless of the runtime behavior of LO-tasks, provided the execution of at most one HI-task overruns its lower WCET estimate.

Baruah et. al. [1] proposed an effective algorithm, called OCBP (Own Criticality Based Priority), to schedule a set of non-recurrent jobs. Although OCBP is able to achieve the highest speedup factor among all the fixed-job-priority algorithms, it cannot be applied to recurrent tasks.

Li and Baruah [10] proposed an algorithm, referred to as LB, to extend OCBP to sporadic tasks, but it relies on very pessimistic schedulability tests based on load bound conditions. Moreover, it introduces large run-time overhead as it needs on-line pseudo-polynomial priority assignment recomputation. To overcome the limitations of LB, Guan et al. [7] presented a new algorithm, referred to as PLRS (Priority List Reuse Scheduling) to schedule certifiable mixed-criticality sporadic task systems.

Pellizzoni et al. [13] proposed a reservations-based approach to ensure strong isolation among subsystems of different criticality. Petters et. al. [14] also considered the use of temporal isolation of subsystems for mixed-criticality systems, and addressed many practical issues in building such systems in reality. In general, the drawback of the resource/temporal isolation approach is that it relies on severely over-provisioning computing resources, which may result in significant cost and energy waste.

Baruah et al. proposed the EDF-VD algorithm [3,4], which is similar to the server mechanism proposed in this paper. In particular, they propose to anticipate the deadlines of HI-tasks so that when executing in LO-criticality mode their completion is anticipated. Their formula for computing the anticipated deadline is global, i.e. it relies on the global utilisation of HI-tasks, whereas in this paper we propose a different formula that accounts for the bandwidth of each different HI-task separately and independently. In [3], the authors also propose a test for checking the schedulability of LO-tasks in LO-criticality mode, and compute the speed-up factor of their algorithm to be 4/3.

Santy et al. [15] propose an algorithm for letting some of the LO-criticality task execute even after the system has switched to HI-criticality mode, as long as their execution does not compromise the schedulability of HI-tasks. Also, they propose a method to reset the system criticality level at certain specified idle intervals. In this paper, we also propose to continue executing LO-criticality tasks as soft real-time tasks even after the system switches to HI-criticality: the temporal isolation mechanism guarantees that the HI-tasks will not be influenced.

6 Conclusions

We presented a reservation-based approach to schedule mixed criticality systems in a way that guarantees the schedulability of high-criticality tasks independently of the behavior of low-criticality tasks. Pessimism related to off-line budget allocation is avoided by an efficient reclaiming mechanism that exploits the budget left by high-criticality tasks for those active low-criticality tasks that can still complete within their deadlines.

We are currently investigating a test for guaranteeing the schedulability of LO-tasks in LO-criticality mode, by computing a lower bound on the amount of reclaiming available in any time interval. We will then compare the schedulability test with the one proposed in [3] to evaluate the performance of the two approaches.

References

1. S. Baruah, H. Li, and L. Stougie. Towards the design of certifiable mixed-criticality systems. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2010)*, 2010.
2. S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. Scheduling real-time mixed-criticality jobs. *IEEE Transactions on Computers*, 61(8):1140 – 1152, 2012.
3. S. K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *ECRTS'12*, pages 145–154, 2012.
4. S. K. Baruah, V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie. Mixed-criticality scheduling of sporadic task systems. In *Proceedings of the 19th European conference on Algorithms*, ESA'11, pages 555–566, Berlin, Heidelberg, 2011. Springer-Verlag.
5. D. de Niz, K. Lakshmanan, and R. Rajkumar. On the scheduling of mixed-criticality real-time task sets. In *Proceedings of the 30th IEEE Real-Time Systems Symposium (RTSS 2009)*, December 2009.
6. G.Lipari and S. Baruah. Greedy reclaimation of unused bandwidth in constant bandwidth servers. In *IEEE Proceedings of the 12th Euromicro Conference on Real-Time Systems*, Stokholm, Sweden, June 2000.
7. N. Guan, P. Ekberg, M. Stigge, and W. Yi. Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems. In *Proceedings of the 32nd IEEE Real-Time Systems Symposium (RTSS 2011)*, 2011.
8. K. Lakshmanan, D. de Niz, and R. Rajkumar. Resource allocation in distributed mixed-criticality cyber-physical systems. In *Proceedings of the 30th International Conference on Distributed Computing Systems (ICDCS 2010)*, 2010.
9. K. Lakshmanan, D. de Niz, and R. Rajkumar. Mixed-criticality task synchronization in zero-slack scheduling. In *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2011)*, 2011.
10. H. Li and S. Baruah. An algorithm for scheduling certifiable mixedcriticality sporadic task systems. In *Proceedings of the 31st IEEE Real-Time Systems Symposium (RTSS 2010)*, December 2010.
11. C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.
12. L. Palopoli, G. Lipari, L. Abeni, M. D. Natale, P. Ancilotti, and F. Conticelli. A tool for simulation and fast prototyping of embedded control systems. In S. Hong and S. Pande, editors, *LCTES/OM*, pages 73–81. ACM, 2001.
13. R. Pellizzoni, P. Meredith, M. Nam, M. Sun, M. Caccamo, and L. Sha. Handling mixed criticality in soc-based real-time embedded systems. In *Proceedings of the ACM International Conference on Embedded Software (EMSOFT 2009)*, 2009.

14. S. Petters, M. Lawitzky, R. Heffernan, and K. Elphinstone. Towards real multi-criticality scheduling. In *15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2009)*, 2009.
15. F. Santy, L. George, P. Thierry, and J. Goossens. Relaxing mixed-criticality scheduling strictness for task sets scheduled with fp. In I. C. Society, editor, *ECRTS*, pages 155–165, July 2012.
16. S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007)*, pages 239–243, December 2007.