Real-Time Syst
DOI 10.1007/s11241-015-9229-9

A framework for memory contention analysis in multi-core platforms

Dakshina Dasari[1] · **Vincent Nelis**[1] · **Benny Akesson**[2]

Abstract The last decade has witnessed a major shift towards the deployment of embedded applications on multi-core platforms. However, real-time applications have not been able to fully benefit from this transition, as the computational gains offered by multi-cores are often offset by performance degradation due to shared resources, such as main memory. To efficiently use multi-core platforms for real-time systems, it is hence essential to tightly bound the interference when accessing shared resources. Although there has been much recent work in this area, a remaining key problem is to address the diversity of memory arbiters in the analysis to make it applicable to a wide range of systems. This work handles diverse arbiters by proposing a general framework to compute the maximum interference caused by the shared memory bus and its impact on the execution time of the tasks running on the cores, considering different bus arbiters. Our novel approach clearly demarcates the arbiter-dependent and independent stages in the analysis of these upper bounds. The arbiter-dependent phase takes the arbiter and the task memory-traffic pattern as inputs and produces a model of the availability of the bus to a given task. Then, based on the availability of the bus, the arbiter-independent phase determines the worst-case request-release scenario that maximizes the interference experienced by the tasks due to the contention for the bus. We show that the framework addresses the diversity problem by applying

✉ Dakshina Dasari
dakshina.dasari@gmail.com; dandi@isep.ipp.pt

Vincent Nelis
nelis@isep.ipp.pt

Benny Akesson
kessoben@fel.cvut.cz

1 CISTER-Research Unit, Porto, Portugal

2 Czech Technical University in Prague, Prague 6, Czech Republic

Published online: 09 June 2015

it to a memory bus shared by a fixed-priority arbiter, a time-division multiplexing (TDM) arbiter, and an unspecified work-conserving arbiter using applications from the MediaBench test suite. We also experimentally evaluate the quality of the analysis by comparison with a state-of-the-art TDM analysis approach and consistently showing a considerable reduction in maximum interference.

Keywords Multicore · Timing analysis · Bus contention · Real-time embedded systems · Worstcase execution time · Bus arbitration · Memory contention

1 Introduction

Embedded systems are increasingly based on multi-core platforms to cater to increasing performance demands while satisfying power constraints (Kollig et al. 2009; van Berkel 2009; Benini et al. 2012; Nowotsch and Paulitsch 2012). These platforms reduce cost by sharing resources, such as buses and external memories, between the applications executing on the cores. Many embedded applications have been successfully deployed on these platforms and are harnessing the benefits of their computational capabilities. However, system designers are unable to leverage the entire potential provided by these platforms to deploy hard real-time applications, for which upper bounds on worst-case execution times (WCET) must be determined at design time and deadlines must be strictly met at run time. Although techniques to determine the WCET of tasks executing on a single-core architecture (Wilhelm et al. 2008) exist, there are still many open issues in a multi-core setting due to the resource sharing between the cores (Dasari et al. 2013). This paradigm of resource sharing does not adhere to the temporal and spatial isolation of components desired by the system designers, because it results in *contention* between tasks executing asynchronously on different cores, which in turn further complicates the process of computing the WCETs of the tasks. Additionally, resource sharing also introduces a circular dependence between WCET and inter-core interference and complicates the WCET analysis process. This problem is important since memory-intensive tasks are stalled for considerable time during data transfers between the cores and the memory (Nowotsch and Paulitsch 2012). Failure to capture this contention at design time results in non-conservative bounds, while pessimistic analyses may result in substantial over-estimation and lead to underutilized resources. A particular challenge is that the contention is heavily dependent on the arbitration policy of the memory bus, which ranges from work-conserving priority-based policies in high-performance soft real-time systems to non-work-conserving time-division-multiplexing (TDM) for critical systems that require robust partitioning.

Existing work addresses the problem of deriving upper bounds on memory bus contention, but the analysis is tightly coupled to a particular arbitration policy, such as TDM (Rosén et al.2007; Chattopadhyay et al. 2010, 2014; Kelter et al. 2011; Schranzhofer et al. 2010, 2011) or non-specified work-conserving arbiters (Dasari et al. 2011; Schliecker et al. 2010), and a generic framework to handle different arbitration mechanisms does not exist. As a result, a change of memory arbiter currently implies adopting a new analysis with different inputs and assumptions.

This article addresses this problem by proposing a general framework for memory bus contention analysis that addresses the range of arbitration policies in multi-core systems. The three main contributions of this work are: (1) A model that captures the best-case and worst-case availability of the shared memory bus. This model can be applied to a range of arbitration policies in a streamlined manner, and we demonstrate its flexibility by applying it to two very different cases, being non-work-conserving TDM and work-conserving fixed-priority arbitration. We also show how round-robin arbitration and unspecified work-conserving arbiters can be captured as special-cases of these two arbiters. (2) An algorithm that uses the proposed bus model and leverages the task request-profiles to compute the maximum memory bus contention that a given task can incur. (3) A method to tighten the computed bounds and increase the efficiency and scalability of the algorithm by splitting the task request-profile into multiple smaller sampling regions. We experimentally evaluate the proposed approach by applying it to a multi-core system providing access to an external DRAM via a shared bus. The flexibility of the framework is demonstrated by applying it to different arbiters on a set of applications from the MediaBench benchmark (Lee et al. 1997). The tightness of the WCET bounds is also evaluated for different sample region sizes. Lastly, our results are compared to a state-of-the-art approach (Schranzhofer et al. 2010), showing that our framework provides tighter bounds for TDM arbitration.

The rest of this article is organized as follows: Sect. 2 presents the system model, followed by an overview of our four-step approach in Sect. 3. The different steps of our approach are then discussed in detail, starting with our novel bus availability model in Sect. 4. We then proceed by showing how to capture the worst-case interference caused by the shared memory bus in Sects. 5 and 6. This is followed by a method to improve the accuracy of the analysis and reduce its computation time in Sect. 7. Related work is discussed in Sect. 8, before we experimentally evaluate the approach in Sect. 9. The article is concluded in Sect. 10.

2 System model

First, we present the platform model, followed by a characterization of the tasks and their corresponding request profiles. We then explain the assumptions on the task scheduler, before formulating the exact problem studied in this work.

2.1 Platform model

The considered multicore platform contains m cores denoted by $\pi_1, \pi_2, \ldots, \pi_m$. It is assumed that the cores do not share cache memory or that all levels of shared cache are disabled or partitioned. This assumption of a private/partitioned cache aligns with the recommendations for certification of hard real-time systems (IEC 61508 2010). A cache miss in the last-level cache results in a memory request to the shared DRAM. The cores are assumed to be fully timing compositional (Wilhelm et al. 2009), such as the ARM7, and stall on every memory request. These assumptions are consistent with current state-of-the-art approaches (Pellizzoni et al. 2010; Schranzhofer et al. 2010, 2010; Rosén et al. 2007) for bus contention analysis.

The cores communicate with the memory through a shared memory bus. The bus controller grants access to the bus in units of *bus slots*, where each slot is of constant length TR that corresponds to an upper bound on the time to serve a memory request, expressed in clock cycles at the frequency of the processor. Contention between the cores is resolved by the memory bus arbitration policy, which depends on the considered platform. Fixed-priority arbitration is typically used in systems with diverse response time requirements, TDM in systems that require robust partitioning between applications, and round robin when a simple notion of fairness between applications executing on different cores is required.

2.2 Task model

The applications are modeled by a set, τ, of sporadic and constrained-deadline tasks in which a task $\tau_i \in \tau$ is characterized by three parameters: C_i, T_i, and $D_i \leq T_i$. The parameter C_i denotes an upper bound on the execution time of task τ_i when it executes uninterrupted in *isolation*, i.e., with no contention on the shared memory bus; T_i denotes the minimum interval between two consecutive activations of τ_i (the period) and D_i is the deadline of the task. In other words, every task τ_i releases a (potentially infinite) sequence of jobs, each such job must execute for at most C_i time units within D_i time units from its release and two successive jobs of the same tasks are released at least T_i time units apart.

The parameter C_i can be computed by well-known techniques in WCET analysis (Wilhelm et al. 2008). This work focuses on computing C_i' , which denotes an upper bound on the execution time when τ_i executes *with* contention on the memory bus, i.e., when co-scheduled tasks are running on the other cores. Clearly, the value of C_i' is not an inherent property of τ_i but depends on the arbitration policy of the memory bus and on the memory request pattern of the tasks executing concurrently on the other cores during τ_i's execution. Note that the proposed application model is very general, as it only assumes sporadic constrained-deadline tasks executing in parallel on cores and accessing a shared memory. As such, the work applies to many application domains of real-time systems, e.g. automotive and avionics.

2.3 Request and region modeling

We proceed by introducing the notations required for modeling the memory traffic generated by the tasks. To gain a deeper insight into the request distribution, the execution time span of the job is divided into *sampling regions* or sampling intervals. The entire execution of each job of task τ_i is divided into $x_i = C_i / L_i^{\text{reg-size}}$ sequential temporal sampling regions , where $L_i^{\text{reg-size}}$ is the duration (in time units, for e.g. processor cycles) of each region. For brevity, we shall use 'task' (instead of job of a task) to refer to the execution instance in the rest of the document.

For each job of task τ_i, the maximum number of memory requests issued *within* each region is captured by executing the task a significant number of times over different inputs and taking the maximum value. Some measurement-based analysis techniques

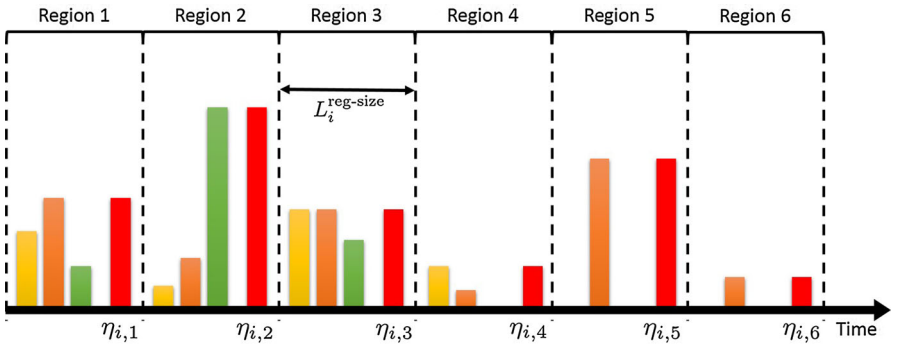


Fig. 1 Illustration of task-region profiles, each with length $L_i^{\text{reg-size}}$ time units

have been proposed to generate test data that would target good code coverage (Wenzel et al. 2009). Figure 1 depicts this task segmentation. It illustrates a same task run three times over different sets of inputs. During the first run, the number of requests issued in each region is recorded and depicted as a yellow box. In that run, the task completes within the fourth region. The green and orange boxes represent two other runs of the same task that complete during the third and sixth region, respectively. The red boxes are the maximum values observed in all runs in each region. Note that the flow-control path leading to the maximum number of memory requests in a region may not be the one that results in the maximum execution time of the application, which introduces some pessimism in our analysis. It is key to understand that the regions defined in our analysis are a sequence of equidistant sampling *points in time* and not the start and end points of a function or any other chunk of code commonly referred to as basic blocks in the literature.

This measurement-based method returns a set, $\mathbb{G}_i = \{\eta_{i,1}, \eta_{i,2}, \ldots, \eta_{i,x_i}\}$, where each $\eta_{i,g}$ ($g \in [1 \ldots x_i]$) is the observed maximum number of requests that task τ_i can generate *within* its g'th sampling region. Note that $\sum_{g=1}^{x_i} \eta_{i,g}$ denotes an upper bound on the number of requests that task τ_i can generate during the entire execution of one of its jobs and, for simplicity, we sometimes use the notation $\eta_{(i)}$ to denote this value, i.e., $\eta_{(i)} \stackrel{\text{def}}{=} \sum_{g=1}^{x_i} \eta_{i,g}$.

We next denote by $\mathbb{R}_i = \{\text{req}_{i,1}, \text{req}_{i,2}, \ldots, \text{req}_{i,\eta_{(i)}}\}$, the set of all requests that τ_i can generate during its execution. Each request $\text{req}_{i,k}$ is modeled by the tuple $\langle \text{rel}_{i,k}, \text{srv}_{i,k} \rangle$, where $\text{rel}_{i,k}$ and $\text{srv}_{i,k}$ denote the release and service time of request $\text{req}_{i,k}$ during τ_i's execution, respectively. Together, these values enable the *cumulative delay* of τ_i due to shared resource accesses to be computed as $\sum_{k=1}^{\eta_{(i)}} (\text{srv}_{i,k} - \text{rel}_{i,k})$. Obviously, the exact values of $\text{rel}_{i,k}$ and $\text{srv}_{i,k}$ are not known before run time.

2.4 Scheduler specification

We consider a partitioned scheme of task assignment in which each task is assigned to a core at design time and is not allowed to migrate from its assigned core to another one at run time, i.e. a fully partitioned non-migrative scheduling scheme. Regarding

the scheduling policy on each core, we consider a non-preemptive scheduler and hence do not deal with cache-related and task-switching overheads. We make the non-work-conserving assumption as follows: whenever a task completes earlier than its WCET (say on its assigned CPU π_p), the scheduler idles the core π_p up to the theoretical WCET of the task. This assumption is made to ensure that the number of bus requests within a time window computed at design time is not higher at run time due to early completion of a task and the subsequent early execution of the following tasks.

2.5 Problem statement

After specifying the model for the platform, the tasks, and their request release and service patterns, we are now ready to state the problem addressed in this work. Given:

1. a multi-core platform conforming to the model described in Sect. 2.1,
2. a set of tasks and their WCET C_i *in isolation*, as described in Sect. 2.2, and
3. the region-profiles of all these tasks, described in Sect. 2.3,

the problem is to compute the WCET C'_i of τ_i when τ_i executes concurrently with other tasks. This implies finding a tight upper bound on the cumulative delay incurred by all memory requests of τ_i considering the contention for the memory bus. For each task τ_j, we must choose a region length $L_j^{\text{reg-size}}$ and obtain as described in Sect. 2.3, the set of region-profiles $\mathbb{G}_j = \{\eta_{j,1}, \eta_{j,2}, \ldots, \eta_{j,x_j}\}$. Then, we must compute the release times and service times, $\text{rel}_{j,k}$ and $\text{srv}_{j,k}$, $\forall\, \text{req}_{j,k} \in \mathbb{R}_j$, under different arbitration policies, that result in the *maximum cumulative delay* $\mathcal{D}_i(\eta_{(i)}) = \sum_{k=1}^{\eta_{(i)}} (\text{srv}_{i,k} - \text{rel}_{i,k})$.

3 Overview

We proceed by giving a high-level overview of the proposed analysis framework. The analysis is presented in four main steps, illustrated in Figure 2, to gradually build up complexity. The four steps are briefly summarized in this section, while the following sections present each step in detail.

3.1 Step 1: modeling the availability of the bus

Given that several tasks are co-scheduled on different cores and contend for the same shared bus, a given task τ_i may not get access to the bus immediately after generating a request. That is, when a task τ_i generates a memory request and therefore requests access to the bus, the bus may or may not be immediately available to serve that request. Section 4 shows how we propose to model the availability of the bus to a given task τ_i using a generic model $\mathbb{B}_i = \langle \mathcal{T}_i^{\min}(), \mathcal{T}_i^{\max}() \rangle$. The functions $\mathcal{T}_i^{\min}(j)$ and $\mathcal{T}_i^{\max}(j)$ are an abstraction of the shared resource that represent the earliest and latest instants at which the bus is available to τ_i for the jth time, as stated in Definitions 1 and 2, respectively. From now on, we refer to the bus slots that are available to the task τ_i as *the free bus slots* of τ_i. We also re-state that each bus-slot is of duration TR and refers to an upper bound on the time to make one memory access. Hence, we assume

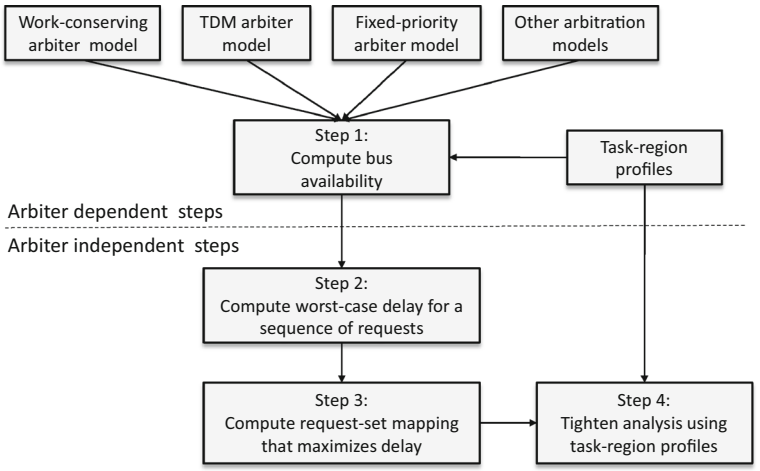


Fig. 2 The main steps of the general analysis framework

a discrete time-line (in units of TR) in which a request can be serviced only starting at the beginning of a time slot.

Definition 1 The function $\mathcal{T}_i^{\min}(j)$ represents the earliest time-instant at which the bus may be available to task τ_i for the jth time, or in other words, the earliest time-instant of the jth free bus slot of τ_i.

Definition 2 The function $\mathcal{T}_i^{\max}(j)$ represents the latest time-instant at which the bus may be available to task τ_i for the jth time, or in other words, the latest time-instant of the jth free bus slot of τ_i.

The order in which the bus slots are granted to the tasks depends on the arbitration mechanism. It hence follows that the two functions $\mathcal{T}_i^{\min}()$ and $\mathcal{T}_i^{\max}()$ also depend on the arbitration mechanism, as shown in Fig. 2. As we shall see in the next section, for some arbitration policies like Time-Division Multiplexing, the availability of the bus modeled by $\mathcal{T}_i^{\min}()$ and $\mathcal{T}_i^{\max}()$ is independent from the traffic generated from the other cores, while for many other arbitration mechanisms, such as fixed-priority scheduling, the interfering requests do influence the time at which the analyzed task τ_i can access the bus and hence play an important role in the computation of the functions $\mathcal{T}_i^{\min}()$ and $\mathcal{T}_i^{\max}()$.

Figure 3 illustrates the earliest and the latest instants at which a given slot is available to task τ_i. As seen in the figure, the earliest instant at which slot 1 may be available to task τ_i , $\mathcal{T}_i^{\min}(1)$, is at time 0, meaning that either there are no pending requests from the other tasks that are co-scheduled on the interfering cores in $\bar{\pi}(i)$, or the task τ_i is executing in isolation and thus there is no contention on the bus. In contrast, we have $\mathcal{T}_i^{\max}(1) = 5$ in this example, which means that τ_i may have to wait at most 4 slots before getting access to the bus. Similarly, the availability for the subsequent slots is depicted in the figure. Note that this illustrative example is not representative of any particular arbitration mechanism.

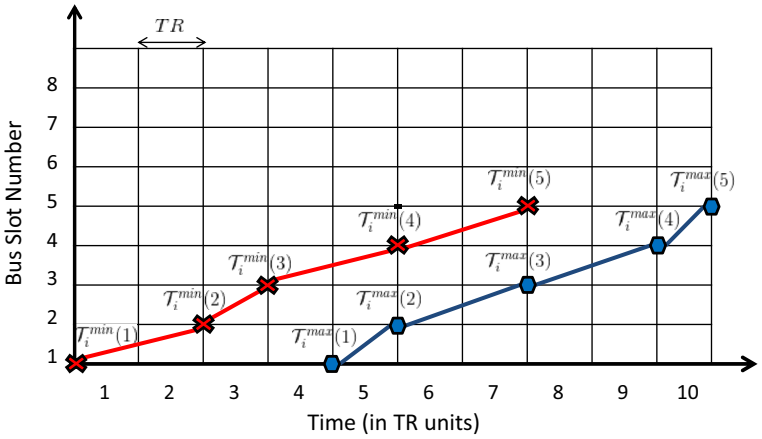


Fig. 3 Illustration of the functions $\mathcal{T}_i^{\min}()$ and $\mathcal{T}_i^{\max}()$

3.2 Step 2: computing the maximum cumulative delay

Given the bus availability model $\mathbb{B}_i = \langle \mathcal{T}_i^{\min}(), \mathcal{T}_i^{\max}() \rangle$ of task τ_i, we propose an algorithm to compute the maximum cumulative delay incurred by memory requests of τ_i considering contention on the shared bus. To achieve this, we define the two concepts of *request-to-slot assignment* and *request-set mapping* as follows.

Definition 3 A request-to-slot assignment in the context of a *single request* $\text{req}_{i,k}$ is denoted by $\sigma_i(k)$ and defines that the kth request generated by τ_i, i.e. $\text{req}_{i,k}$, is served in the $\sigma_i(k)$th bus slot available to task τ_i, i.e. in the $\sigma_i(k)$th free bus slot of τ_i.

Definition 4 For a given task τ_i, a request-set mapping $\mathbb{M}_i = \{\sigma_i(1), \sigma_i(2), \ldots, \sigma_i(\eta_{(i)})\}$ defines that $\forall k \in \mathbb{R}_i$: $\text{req}_{i,k}$ is assigned to the $\sigma_i(k)$th free bus slot of τ_i.

Given these definitions, we further divide this step into two phases; the first phase focuses on the maximum delay incurred by a single request and the second focuses on the maximum cumulative delay incurred by a set of consecutive requests.

Phase 1. Given the bus availability model $\mathbb{B}_i = \langle \mathcal{T}_i^{\min}(), \mathcal{T}_i^{\max}() \rangle$ of task τ_i, its kth request $\text{req}_{i,k}$ and a request-to-slot assignment $\sigma_i(k)$ for that request, we compute the maximum delay that $\text{req}_{i,k}$ can incur by computing a lower-bound on its release time, $\text{rel}_{i,k}$, and an upper-bound on its service time, $\text{srv}_{i,k}$, with the objective of maximizing its waiting time (i.e., $\text{srv}_{i,k} - \text{rel}_{i,k}$).

Phase 2. In the second phase, given the bus availability model represented as $\mathbb{B}_i = \langle \mathcal{T}_i^{\min}(), \mathcal{T}_i^{\max}() \rangle$ of task τ_i and a request-set mapping $\mathbb{M}_i = \{\sigma_i(1), \sigma_i(2), \ldots, \sigma_i(\eta_{(i)})\}$ for all its requests, we compute the overall maximum cumulative delay that can be incurred by these requests.

This proposed analysis to compute the maximum cumulative delay for a given request-set mapping is presented in Sect. 5.

3.3 Step 3: finding the worst-case request-set mapping

While the previous step provides an algorithm to compute the maximum cumulative delay for a *given request-set mapping*, the goal of this third step is to find a request-set mapping for which the maximum cumulative delay is the largest among all feasible mappings. We propose an algorithm in Sect. 6 to determine such a request-set mapping for all the requests of a given task τ_i. Our technique first computes an upper bound, UBslot_i, on the number of free bus slots that can possibly be used by task τ_i. This upper bound gives us a conservative range $[1, \text{UBslot}_i]$ of free bus slots within which all the requests of the analyzed task τ_i will be served. Note UBslot_i, may be much greater than the number $\eta_{(i)}$ of requests to be served.

A naive approach to maximize the cumulative delay incurred by the (at most) $\eta_{(i)}$ requests of τ_i is to apply brute force, i.e. all the request-set mappings are explored and a maximum cumulative delay is computed for each of them using the method proposed in Step 2; At the end, only the largest cumulative delay from all mappings is returned. However, such a method does not scale and is computationally inefficient due to the exhaustive exploration of all the possible mappings. We significantly reduce the computation time of the proposed analysis by eliminating the request-set mappings that cannot possibly lead to the worst-case delays at an early stage of the analysis.

3.4 Step 4: tightening the analysis using sampling regions

Having shown how to determine the worst request-set mapping in Step 3, and bounding the maximum cumulative delay for that mapping using the technique explained in Step 2, there is further scope of tightening the analysis by exploiting the information about the maximum number of requests in each of the constituent regions of the analyzed task. The region-based analysis limits the range of the potential free bus slots used by a set of requests. For example, if the kth request of τ_i is generated in the g'th region, then it *cannot* be served in the jth free bus slot of τ_i if $\mathcal{T}_i^{\max}(j) < L_i^{\text{reg-size}} \times (g-1)$, where $L_i^{\text{reg-size}}$ is the size of the sampling interval. From these constraints, we define a range $[\text{LBslot}_{i,g}, \text{UBslot}_{i,g}]$ for each region g, representing the first and last free bus slots in which requests from region g can possibly be served. These bounds are employed by the proposed algorithm to tighten the analysis by defining a request-set mapping for each individual region. The maximum delays incurred by the requests of each region are computed successively and the overall WCET is subsequently computed. This process is described in detail in Sect. 7.

4 Modeling the availability of the bus

The memory bus is a shared resource, which means that any access to it by a given task may be deferred because of concurrent accesses from other tasks. To estimate the overall delay that can be incurred by a task due to the contention for a shared bus, a basic approach could be to first derive an upper bound on the delay that a *single* access may incur. This upper bound is computed by constructing a worst-case

scenario in which every competing task gathers all its accesses to the bus within the shortest possible time window. This creates a burst of accesses all concentrated in time and occurring exactly when the request from the analyzed task is released, thereby inducing the maximum delay for this request. Then, the overall delay that a *sequence* of requests may suffer is computed by assuming that each access to the shared bus incurs this precomputed maximum delay. This assumption will lead to conservative estimates, since the other tasks keep progressing in their execution, alternating between computation and memory fetch phases, and do not congest the memory bus at all times.

We propose an alternative approach which bases its computation on a new modeling framework. Instead of computing a worst-case scenario for a single access to the shared bus and then considering that scenario for each and every request of the analyzed task, we model the overall availability of the bus to the analyzed task. Then, as the next step, we leverage this new model to derive an upper bound on the *cumulative* delay that a *sequence* of requests may incur.

Our model captures the best-case and worst-case availability of the shared bus. It is based on the arbiter and coarse-grained memory access information provided by the task-region profiles. Specifically, for a given task τ_i under analysis and any positive integer j, we compute the two functions $\mathcal{T}_i^{\min}(j)$ and $\mathcal{T}_i^{\max}(j)$ that give the *earliest* and *latest* instants of the jth free bus slot of τ_i. If τ_i is run in isolation, there are no competing requests for a work conserving bus, and the bus is always available to τ_i. In such a case $\mathcal{T}_i^{\min}(j) = \mathcal{T}_i^{\max}(j) = (j-1) \cdot \mathrm{TR}$ for all $j > 0$. In other cases, when the task is in contention or the bus arbiter uses reservation of slots as in TDM, we have $\mathcal{T}_i^{\min}(j) < \mathcal{T}_i^{\max}(j)$. These two functions form what we call the *bus availability model* $\mathbb{B}_i = \langle \mathcal{T}_i^{\min}(), \mathcal{T}_i^{\max}() \rangle$ of task τ_i. This model can be computed for any predictable resource and a wide range of arbitration policies. Next, we demonstrate the computation of this bus model for two distinct cases: a non-work-conserving TDM arbiter and a work-conserving fixed-priority arbiter.

4.1 Non-work-conserving TDM arbitration

A TDM arbiter works by periodically repeating a schedule, or frame, with fixed size, f. Each core π_p is allocated a number of slots ϕ_p in the frame at design time, such that $\sum_{\pi_p} \phi_p \leq f$. There are different policies for distributing the slots allocated to a core within the TDM frame, but here we consider the case where slots are assigned contiguously for simplicity. An example of a TDM frame, a contiguous allocation, and some of the associated terminology is illustrated in Fig. 4.

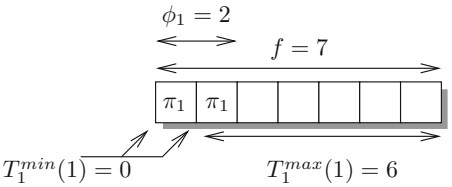


Fig. 4 TDM frame with 7 slots using a contiguous slot allocation

We consider a non-work-conserving instance of the TDM arbiter, which means that requests from a core are only scheduled during bus slots allocated to that core. Empty slots or slots allocated to other cores without pending requests are hence not utilized. This type of policy makes the timing behavior of memory requests of tasks scheduled on different cores completely independent. As a result, only the configuration of the arbiter has to be considered when determining $\mathcal{T}^{\min}()$ and $\mathcal{T}^{\max}()$. For non-work-conserving TDM arbitration with a contiguous slot allocation, $\mathcal{T}^{\min}()$ and $\mathcal{T}^{\max}()$, for task τ_i assigned to core π_p are derived according to Eqs. (1) and (2), respectively.

$$\mathcal{T}_i^{\min}(j) = \left(\left\lfloor \frac{j-1}{\phi_p} \right\rfloor \times f + ((j-1) \mod \phi_p) \right) \times \mathrm{TR} \tag{1}$$

$$\mathcal{T}_i^{\max}(j) = \mathcal{T}_i^{\min}(j) + (f - \phi_p + 1) \times \mathrm{TR} \tag{2}$$

The first term in the computation of $\mathcal{T}^{\min}()$ in Eq. (1) corresponds to the minimum required number of full iterations of the TDM frame to produce j free slots for τ_i and the second term corresponds to the remaining number of required slots after these iterations. The computation of $\mathcal{T}^{\max}()$ is similar, except that it adds an additional $f - \phi_p + 1$ slots to account for releases with maximum misalignment with respect to the set of contiguous slots allocated to the core in the TDM frame (including just missing its own last slot, i.e. the "+1"). Note that these equations also cover non-work-conserving round-robin arbitration, since it is just a special case of TDM where f equals the number of cores sharing the bus and $\forall \pi_p \ \phi_p = 1$. Work-conserving versions of both these arbitration policies can be derived by additionally considering the task-region profiles, although this is omitted for brevity. Figure 4 graphically illustrates the arrival times and waiting times corresponding to $\mathcal{T}_1^{\min}(1)$ and $\mathcal{T}_1^{\max}(1)$. As seen in the figure, the $\mathcal{T}_1^{\min}(1) = 0$, is achieved for a request that arrives just at the beginning of any of the two slots allocated to its corresponding core and $\mathcal{T}_1^{\max}(1) = 6$ for a request arriving just after the last slot allocated to its core has been left idle. For this particular arbitration policy, the best-case and worst-case arrival with respect to the TDM frame is the same for any value of j, although this does not hold in general.

4.2 Work-conserving fixed-priority arbitration

In the context of bus arbitration policies, one of the challenges with currently existing COTS-based multi-core systems is that the memory bus does not recognize/respect task priorities. This is because the bus is generally designed with the aim of enhancing the average-case performance and is not tailored for real-time systems. This can lead to a scenario similar to priority inversion in which requests from higher priority tasks are delayed by requests from lower-priority tasks on the bus. Although the scheduler enforces these priorities while allocating cores to tasks, these priorities are not passed over to the shared hardware resources like the memory bus, which have their own scheduling policies. This problem has been addressed in research by enabling priorities in priority-driven arbiters to be software programmable directly (Akesson et al. 2009) or indirectly by tagging each request with its priority (Zhou et al. 2011). We assume

in this section that the memory bus is designed according to any of these strategies. Based on this, we design a bus-availability model for a fixed-priority arbiter.

Assume that the analyzed task τ_i is scheduled on core π_p. Despite the uncertainty of the arrival patterns of the requests, it is important to determine a lower and upper bound on the cumulative number of requests that tasks with higher priority than τ_i and scheduled on the interfering cores $\pi_q \neq \pi_p$ may inject into the bus. These bounds are denoted by the *per-core request profiles* $\text{PCRP}_q^{min}(i,t)$ and $\text{PCRP}_q^{max}(i,t)$ functions, respectively, which can be computed as shown in Dasari and Nelis (2012) with a pseudo-polynomial time complexity of $O(C_{\max}^2)$, where $C_{\max}$ denotes the maximum WCET among tasks deployed on the given core.

The computation of these functions, in summary is akin to the bin packing problem, in which we need to pack the maximum (or minimum) number of (interfering) requests in a given interval of duration t during which the analyzed task τ_i executes. To do so, the tasks of higher priority than τ_i that run on the interfering cores are ordered by request densities and then packed within an interval of time t in such a manner that the minimum (or maximum) number of requests is derived, while respecting the task arrival rates. With this information, we derive the corresponding earliest and latest times at which free slots are available to the analyzed task τ_i assigned to core π_p according to Eqs. (3) and (4), respectively. Just like the computation of $\mathcal{T}^{\max}()$ for TDM in Eqs. (2), (4) adds an extra TR to capture the possible situation where the task just misses the last free slot before experiencing its worst-case interference pattern.

$$\mathcal{T}_i^{\min}(j) = \min_{t \geq 0} \left\{ t | t - \left(\sum_{\pi_q \neq \pi_p} \text{PCRP}_q^{\min}(i,t) \times \text{TR} \right) = (j-1) \times \text{TR} \right\} \tag{3}$$

$$\mathcal{T}_i^{\max}(j) = \min_{t \geq 0} \left\{ t | t - \left(\sum_{\pi_q \neq \pi_p} \text{PCRP}_q^{\max}(i,t) \times \text{TR} \right) = (j-1) \times \text{TR} \right\} + \text{TR} \tag{4}$$

From the perspective of the analyzed task τ_i executing on core π_p, the memory bus can be viewed as a resource with two alternating phases: a busy phase, in which it serves the requests from the other cores and an idle phase, in which it is available to τ_i. Equation (4) can be interpreted as follows: The other cores will issue $\sum_{\pi_q \neq \pi_p} \text{PCRP}_q^{\max}(i,t)$ requests and utilize the corresponding number of bus slots, each of length TR. The analyzed task can only serve its jth request after $\sum_{\pi_q \neq \pi_p} \text{PCRP}_q^{\max}(i,t)$ requests of the interfering tasks are served and then a free slot is available. In the worst case, the request will be released just after the beginning of the free slot. Hence the next slot will be available for the request of the analyzed task after a time TR. Given this, when in isolation, $\sum_{\pi_q \neq \pi_p} \text{PCRP}_q^{\min}(i,t) = 0$ and $\sum_{\pi_q \neq \pi_p} \text{PCRP}_q^{\max}(i,t) = 0$ and hence $\mathcal{T}_i^{\min}(1) = 0$ and $\mathcal{T}_i^{\max}(1) = \text{TR}$. We pre-compute and store values of $\mathcal{T}_i^{\max}()$ and $\mathcal{T}_i^{\min}()$ for all j while the resulting $t \leq D_i$.

Just like our TDM model can be used to capture the special case of round-robin arbitration, the unspecified work-conserving arbiter presented in Schliecker et al. (2010), Andersson et al. (2010) is just a special case of the fixed-priority arbiter presented in this

section. The unspecified work-conserving arbiter was defined in the context of COTS systems, where the arbitration mechanism is not always specified. However, it is still possible to analyze the system if it can be assumed that the arbiter is work-conserving, which is reasonable for example in the context of commercially-available memory controllers that are designed to optimize average performance. Capturing this arbiter with this model only requires a slight modification to the PCRP functions to make every task believe that it has the lowest priority in the system, which is the same way it was captured in the original publications. Although this is likely to result in a very pessimistic estimation of the worst-case delay, it enables conservative analysis of the system.

As seen in this section, the $\mathcal{T}_i^{\min}(j)$ and $\mathcal{T}_i^{\max}(j)$ functions are arbitration dependent and can be computed for different arbiters (TDM, round-robin, fixed-priority and the unspecified work-conserving arbiter). These functions serve as an input to the next steps of the proposed framework that compute the increased execution time based on the model. In contrast, the methods described in the following sections are independent of the arbitration mechanism.

5 Finding the maximum delay for a request-set mapping

We have presented a model that captures the availability of the memory bus to a given task and demonstrated its use for two very different arbitration mechanisms and highlighted additional arbiters that are supported as special cases of these. This section continues by first describing a method to compute the maximum waiting time of a request $\text{req}_{i,k}$, given a request-to-slot assignment $\sigma_i(k)$ for that request, and the bus availability model $\mathbb{B}_i$. The same rationale is then extended to compute the cumulative waiting time for a sequence of requests of a given task τ_i.

5.1 Maximum delay for a single request

For a given request $\text{req}_{i,k}$ and its request-to-slot assignment $\sigma_i(k)$, the key idea to maximize its waiting time is to release that request as early as possible and delay its service time as much as possible. In other words, for a given request $\text{req}_{i,k}$ and a request-to-slot assignment $\sigma_i(k)$, we need to determine a *lower* bound on its release time and an *upper* bound on its service time and then compute the resulting waiting time. This is done in Lemmas 1 and 2, respectively.

Lemma 1 (A lower bound on the release time of a request) *For any task $\tau_i \in \tau$ and for all $k > 1$, let $\text{req}_{i,k-1}$ and $\text{req}_{i,k}$ be two consecutive requests generated by τ_i. For a given request-to-slot assignment $\sigma_i(k-1)$ and $\sigma_i(k)$, if request $\text{req}_{i,k-1}$ has been served at time $\text{srv}_{i,k-1}$ in the $\sigma_i(k-1)$'th free bus slot then it holds that the release time $\text{rel}_{i,k}$ of $\text{req}_{i,k}$ is such that*

$$\text{rel}_{i,k} \geq \max\left(\mathcal{T}_i^{\min}(\sigma_i(k)-1)+1,\ \text{srv}_{i,k-1}+(\sigma_i(k)-\sigma_i(k-1))\times \text{TR}\right) \quad (5)$$

Proof The lemma is based on two simple observations, corresponding to the two terms in Eq. (5).

1. If it is given that $\text{req}_{i,k}$ is served in the $\sigma_i(k)$'th free bus slot of τ_i then its earliest release time is *immediately after* the earliest time-instant at which the bus can be free for the $(\sigma_i(k) - 1)$'th time. Otherwise, the request would have been served in the previous available free slot, $(\sigma_i(k) - 1)$. Formally, this implies $\text{rel}_{i,k} \geq \mathcal{T}_i^{\min}(\sigma_i(k) - 1) + 1$.
2. Since we assume that a core stalls while its requests are being served, it follows that a request can only be released after the previous request from the same task has been served, i.e. $\text{rel}_{i,k} \geq \text{srv}_{i,k-1}$. In addition, for request $\text{req}_{i,k}$ to be served in the $\sigma_i(k)$'th free bus slot of τ_i, it must hold that $\text{req}_{i,k}$ has missed all the intermediate free bus slots between the $\sigma_i(k-1)$'th and the $\sigma_i(k)$'th, i.e. $\text{rel}_{i,k} \geq \text{srv}_{i,k-1} + (\sigma_i(k) - \sigma_i(k-1)) \times \text{TR}$.

In order to satisfy both conditions, the maximum of the resulting values is considered. □

Lemma 2 (An upper bound on the service time of a request) *For any task $\tau_i \in \tau$ and for all $k > 1$, if request $\text{req}_{i,k}$ is served at time $\text{srv}_{i,k}$ in the $\sigma_i(k)$'th free bus slot then it holds that*

$$\text{srv}_{i,k} \leq \min\left(\mathcal{T}_i^{\max}(\sigma_i(k)), \text{rel}_{i,k} + \mathcal{T}_i^{\max}(1)\right) \tag{6}$$

Proof The latest time at which request $\text{req}_{i,k}$ assigned to slot $\sigma_i(k)$ is served is $\mathcal{T}_i^{\max}(\sigma_i(k))$ (by definition). However, since $\mathcal{T}_i^{\max}(1)$ is defined as the maximum delay that a request may suffer, the value of $\text{srv}_{i,k}$ cannot be greater than $\text{rel}_{i,k} + \mathcal{T}_i^{\max}(1)$. Equation (6) upholds these two conditions by considering the minimum of the respective values. □

The maximum delay for servicing the given request $\text{req}_{i,k}$ in slot $\sigma_i(k)$ is then given by the difference between the upper bound on its service time and the lower bound on its release time.

5.2 Maximum cumulative delay for a request-set mapping

In the previous section, we established a method to compute an upper bound on the delay of a single request assigned to a given free bus slot. Now, we extend this result to maximize the cumulative delay of a *sequence* of $\eta_{(i)}$ requests, given a request-set mapping $\mathbb{M}_i = \{\sigma_i(1), \ldots, \sigma_i(\eta_{(i)})\}$ for that sequence. To maximize the cumulative delay for the mapping $\mathbb{M}_i$, we compute the individual maximum delay for each request by applying Lemmas 1 and 2. Since the release time (and thus the delay) of a given request $\text{req}_{i,k}$ depends on the service time $\text{srv}_{i,k-1}$ of the previous one (see Eq. (5)), we start by computing the maximum delay of the first request $\text{req}_{i,1}$ and iterate up to request $\text{req}_{i,\eta_{(i)}}$. We show in Lemma 3 that this iterative process leads to a worst-case cumulative delay. The lemma is proven by induction and case enumeration and is found in the Appendix. The main benefit of the lemma is that it establishes that the maximum cumulative delay of a request-set mapping can be computed in an iterative manner. We exploit this in the next section as we present an algorithm to find the worst-case request-set mapping.

Lemma 3 (Worst-case cumulative delay) *Let* $\mathbb{M}_i = \{\sigma_i(1), \ldots, \sigma_i(\eta_{(i)})\}$ *refer to a request-set mapping for the* $\eta_{(i)}$ *requests of task* τ_i *and* $\mathcal{D}_i(k)$ *the maximum cumulative delay for the first k requests* $\{\text{req}_{i,1}, \text{req}_{i,2}, \ldots \text{req}_{i,k}\}$*, given this mapping* $\mathbb{M}_i$*. The cumulative delay* $\mathcal{D}_i(\eta_{(i)}) = \sum_{k=1}^{\eta_{(i)}} (\text{srv}_{i,k} - \text{rel}_{i,k})$ *of the* $\eta_{(i)}$ *requests of* τ_i *is maximized for:*

$$\text{rel}_{i,k} = \begin{cases} \mathcal{T}_i^{\min}(\sigma_i(k) - 1) + 1, & \text{if } k = 1 \\ \max(\mathcal{T}_i^{\min}(\sigma_i(k) - 1) + 1, \text{srv}_{i,k-1} + \Delta_k), & \text{otherwise} \end{cases} \tag{7}$$

$$\text{srv}_{i,k} = \min\left(\mathcal{T}_i^{\max}(\sigma_i(k)), \text{rel}_{i,k} + \mathcal{T}_i^{\max}(1)\right) \tag{8}$$

where $\Delta_k = (\sigma_i(k) - \sigma_i(k-1)) \times \text{TR}$.

A detailed proof is presented in the Appendix and the interested reader may please refer the same.

6 Finding the worst-case request-set mapping

We have presented a bus availability model and shown how to leverage it to compute the maximum cumulative delay for *a given sequence of requests and a given request-set mapping*. This section proceeds by presenting how to efficiently determine a request-set mapping for which the maximum cumulative delay is the highest among all possible request-set mappings, i.e. a worst-case request-set mapping. First, we present the basic algorithm to find this worst-case mapping. Then, we proceed by presenting how to eliminate, at an early stage of the computation, many intermediate mappings considered by the algorithm so that the computation time and the memory requirements are reduced.

6.1 Algorithm description

This section proposes an algorithm to find the request-set mapping that maximizes the cumulative delay. In order to eliminate unfeasible mappings that will *provably* not contribute to the global maximum, we start by presenting an important corollary of Lemma 3 followed by a relevant observation, which eventually forms the basis of the algorithm.

Corollary 1 (Dependency between worst-case cumulative delays) *Let us assume a sequence of k requests* $\{\text{req}_{i,1}, \text{req}_{i,2}, \ldots, \text{req}_{i,k}\}$ *from task* τ_i *and a given request-set mapping* $\mathbb{M}_i = \{\sigma_i(1), \sigma_i(2), \ldots, \sigma_i(k)\}$ *for these requests. Let us denote by* $\mathcal{D}_i(k)$ *the* maximum *cumulative delay for these k requests* (*computed using Lemma 3*)*. Now, suppose that we extend the sequence with an extra request with index* $(k+1)$ *assigned to slot h, i.e.* $\sigma_i(k+1) = h$ *such that* $h > \sigma_i(k)$*. The maximum cumulative delay* $\mathcal{D}_i(k+1)$ *for the* $k+1$ *requests can be obtained simply by adding to* $\mathcal{D}_i(k)$ *the maximum delay for that last request* $\text{req}_{i,k+1}$*. This maximum delay for* $\text{req}_{i,k+1}$ *can be obtained by using Eqs.* (7) *and* (8)*, where* $\text{srv}_{i,k}$ *is the service time of the kth request that was obtained during the computation of* $\mathcal{D}_i(k)$*.*

Proof The corollary is a direct consequence of Eqs. (7) and (8). When applying the method of computation of Lemma 3 to the set of $(k+1)$ requests $\{\text{req}_{i,1}, \text{req}_{i,2}, \ldots, \text{req}_{i,k+1}\}$, the resulting cumulative delay $\mathcal{D}_i(k)$ after the kth iteration is the same as the delay $\mathcal{D}_i(k)$ obtained when applying this method to the set of k requests $\{\text{req}_{i,1}, \text{req}_{i,2}, \ldots, \text{req}_{i,k}\}$. In other words, the computation of the maximum cumulative delay for the first k requests is independent of whether or not there is a $(k+1)$'th request in the input sequence. □

Observation 1 *If a sequence of* $(k+1)$ *consecutive requests of a task* τ_i *are served within the first and the h'th slot available to* τ_i*, i.e. within the range* $[1, h]$ *of free bus slots of* τ_i*, then the maximum cumulative delay for these* $(k+1)$ *requests is the maximum between the largest delay computed in the following two scenarios:*

1. *The* $(k+1)$ *requests are* all *served within the range* $[1, h-1]$ *of free bus slots.*
2. *The first k requests are served within the slots* $[1, h-1]$ *and the* $(k+1)$*'th request is served in slot h.*

This observation holds true as these two cases are mutually exclusive and jointly exhaustive*, which implies that any feasible assignment falls either in Cases 1 or 2 and taking the maximum among the resulting delays is trivially safe.*

Based on this corollary and observation, we construct an algorithm to compute $\mathcal{D}_i(k)$ from $\mathcal{D}_i(k-1), \forall k$, which ultimately yields $\mathcal{D}_i(\eta_{(i)})$. The proposed algorithm is *safe-by-construction* as it computes $\mathcal{D}_i(\eta_{(i)})$ by investigating all possible assignments of these $\eta_{(i)}$ requests to the free bus slots and only discards assignments that are proven unfeasible. The algorithm is shown in Algorithm 1 and we proceed by discussing it in detail.

The request-set mappings are captured in a two-dimensional array with $\eta_{(i)}$ rows and UBslot_i columns. The input to the algorithm is the number $\eta_{(i)}$ of requests of the analyzed task τ_i, and an upper bound on the available slots in which the $\eta_{(i)}$ requests may be served. Note that the variables k and j are used to refer to requests and slots, respectively. Each cell(k, j) of this array holds a list of tuples $e_{k,j} = \langle \mathcal{D}_i(k), \sigma_i(k), \text{srv}_{i,k} \rangle$, where each tuple $e_{k,j}$ in that list reflects a feasible request-set mapping of the first k requests to k free bus slots within the range $[1, j]$ of slots available to τ_i. The members of this tuple denote:

- The maximum delay $\mathcal{D}_i(k)$ that can be obtained with the corresponding request-set mapping,
- The free bus slot in which the kth request has been served to reach that maximum delay $\mathcal{D}_i(k)$, i.e. $\sigma_i(k) \in [k, j]$, and
- The corresponding time $\text{srv}_{i,k}$ at which that kth request has been served in that slot to obtain the delay $\mathcal{D}_i(k)$.

The algorithm proceeds in a row-wise manner: it assigns the first request $\text{req}_{i,1}$ to all feasible free bus slots and computes the maximum cumulative delay for each such assignment. Then, it proceeds to analyze the second request (next row of the array) and so on. For the first request and first free bus slot, the algorithm computes *the* worst-case delay when the first request is assigned to that slot (Lines 7, 9, 10). To do so, it uses Lemma 3 and adds the corresponding tuple $e_{1,1}$ to the list of cell(1, 1) in Line 11. In

Algorithm 1: MaxRegDelay($\eta_{(i)}$, UBslot$_i$)

input : $\eta_{(i)}$: no. of requests, UBslot$_i$: last available slot
output: $\mathcal{D}_i(\eta_{(i)})$: maximum cumulative delay incurred by τ_i.

1 Create a 2D array of $\eta_{(i)}$ rows and UBslot$_i$ columns, where each cell(k, j) at row k and column j is a list of tuples $e_{k,j}$, as explained in the description. ;
Set every cell of this array to an empty list $\emptyset$;
2 **for** $k \leftarrow 1$ **to** $\eta_{(i)}$ **do**
3 **for** $j \leftarrow k$ **to** UBslot$_i - (\eta_{(i)} - k)$ **do**
4 **if** $k = 1$ **then**
5 **if** $j > 1$ **then** cell(k, j) $\leftarrow$ cell(k, $j-1$);
6 $\text{rel}_{i,k} \leftarrow \mathcal{T}_i^{\min}(j-1) + 1$;
 // we assume $\mathcal{T}_i^{\min}(0) = -1$
7 **if** $\text{rel}_{i,k} < C_i$ **then**
8 $\text{srv}_{i,k} \leftarrow \min(\mathcal{T}_i^{\max}(j), \text{rel}_{i,k} + \mathcal{T}_i^{\max}(1))$;
9 $\mathcal{D}_i(k) \leftarrow \text{srv}_{i,k} - \text{rel}_{i,k}$;
10 cell(k, j).add($\langle \mathcal{D}_i(k), j, \text{srv}_{i,k} \rangle$);
11 **end**
12 **else**
13 cell(k, j) $\leftarrow$ cell(k, $j-1$);
 // cell(k, $j-1$) $= \emptyset$ if $j = k$
14 **foreach** $e_{k-1,j-1} \in$ cell($k-1$, $j-1$) **do**
 // $e_{k-1,j-1} = \langle \mathcal{D}_i(k-1), \sigma_i(k-1), \text{srv}_{i,k-1} \rangle$
15 $\text{rel}_{i,k} \leftarrow \max(\mathcal{T}_i^{\min}(j-1) + 1, \text{srv}_{i,k-1} + (j - \sigma_i(k-1)) \times \text{TR})$;
16 **if** $\text{rel}_{i,k} < \text{srv}_{i,k-1} + C_i$ **and** $\text{rel}_{i,k} < C_i + \mathcal{D}_i(k-1)$ **then**
17 $\text{srv}_{i,k} \leftarrow \min(\mathcal{T}_i^{\max}(j), \text{rel}_{i,k} + \mathcal{T}_i^{\max}(1))$;
 $\mathcal{D}_i(k) \leftarrow \mathcal{D}_i(k-1) + \text{srv}_{i,k} - \text{rel}_{i,k}$;
 cell(k, j).add($\langle \mathcal{D}_i(k), j, \text{srv}_{i,k} \rangle$);
18 **end**
19 **end**
20 **end**
21 **end**
22 **end**
 // Return the max value of the Delay among the list of tuples stored in the topmost right corner cell, i.e. cell($\eta_{(i)}$, UBslot$_i$)
23 **forall the** $e_{\eta_{(i)},\text{UBslot}_i} \in$ cell($\eta_{(i)}$, UBslot$_i$) **do**
 // $e_{\eta_{(i)},\text{UBslot}_i} = \langle \mathcal{D}_i(\eta_{(i)}), \text{UBslot}_i, \text{srv}_{i,\eta_{(i)}} \rangle$
24 maxDelay $\leftarrow \max(\text{maxDelay}, \mathcal{D}_i(\eta_{(i)}))$;
25 **end**
26 **return** maxDelay;

this case, we have $\text{rel}_{i,1} = 0$, $\text{srv}_{i,k} = \mathcal{T}_i^{\max}(1)$, and $e_{1,1} = \langle \mathcal{T}_i^{\max}(1), 1, \mathcal{T}_i^{\max}(1) \rangle$. The list contains only this tuple. The if-statement of Line 8 aims at reducing the computation time of the algorithm by discarding all the request-to-slot assignments which impose on the first request $\text{req}_{i,1}$ to be released after the task has run for C_i time units, which is impossible.

For $k = 1$ and $j > 1$, the algorithm computes *all* the maximum delays by considering every assignment of the first request, $\text{req}_{i,1}$, to free bus slots $\leq j$. First, the list of the current cell(1, j) is initialized to the list of the previous cell(1, $j-1$) (Line 6), thereby carrying on all the possible worst-case delays that were obtained when this first

request was assigned to a previous free bus slot $< j$. Then, the algorithm addresses the case where the first request is assigned to the jth bus slot by making use of the equations of Lemma 3 to compute $\text{rel}_{i,1}$ and $\text{srv}_{i,1}$ and appends the corresponding tuple $e_{1,j}$ to the list of cell$(1, j)$ (Lines 7, 9, 10, and 11 again).

When $k > 1$ and $j \geq k$, the algorithm computes *all* worst-case delays that can be obtained when the first k requests of τ_i are assigned to any free bus slots within $[k, j]$. On Line 14, the algorithm initializes the list of cell(k, j) to the list of results obtained for the cell$(k, j-1)$. Informally, this reflects Case 1 in Observation 1, which states that the worst-case cumulative delay of the first k requests may be found in the set of maximum delays obtained when these k requests are *all* served *before* the jth free bus slot. Then on Line 15, the algorithm inspects every maximum delay that has been obtained assuming that the first $k-1$ requests were served *before* the jth free bus slot. For each of these delays $\mathcal{D}_i(k-1)$, assuming that the kth request is now served in the jth free bus slot, Lines 16 and 18 compute the release and service time of that request $\text{req}_{i,k}$ using the equations of Lemma 3, by referring to the corresponding request-to-slot assignment $\sigma_i(k-1)$ of the $(k-1)$'th request, as well as its service time $\text{srv}_{i,k-1}$ in this free bus slot $\sigma_i(k-1)$. This reflects Case 2 in Observation 1 as the maximum delay $\mathcal{D}_i(k)$ for the first k requests is computed assuming that request $\text{req}_{i,k}$ is assigned to the jth free slot and the previous $k-1$ requests are served in the earlier bus slots. Note that the computation of the resulting maximum cumulative delay $\mathcal{D}_i(k)$ in Line 19 is safe as explained in Corollary 1.

Similarly to Line 8, the condition of Line 17 is used to filter out a host of unfeasible solutions. In short, the time interval between the release of the currently considered request $\text{req}_{i,k}$ and the service time of the previous one cannot exceed the total execution requirement of the task and the current request cannot be released later than the maximum execution requirement of the task plus the maximum delay $\mathcal{D}_i(k-1)$ that τ_i may incur till there due to interference with the first $(k-1)$ requests.

Note that k spans from 1 to $\eta_{(i)}$, while j takes all values within the range $[k, \text{UBslot}_i - (\eta_{(i)} - k)]$. The reason for limiting the range of j is because the kth request of τ_i cannot possibly be served in a free bus slot $< k$ (leading to a lower bound $j \geq k$) and the next $(\eta_{(i)} - k)$ requests following $\text{req}_{i,k}$ require at least $(\eta_{(i)} - k)$ slots in order to be served (leading to the upper bound $j \leq \text{UBslot}_i - (\eta_{(i)} - k)$).

6.2 Elimination of unfeasible request-set mappings

Having proposed an algorithm to determine the worst-case request-set mapping, we proceed by improving its efficiency. Algorithm 1 carries on all possible request-set mappings and their associated maximum delays, finally returning the one leading to the maximum cumulative delay. This section presents two methods to identify which request-set mappings cannot lead to the worst-case cumulative delay and discard them at an early stage of the computation. By pruning the solution space at each iteration, the set of candidate solutions is substantially reduced, thereby improving the scalability of the algorithm with respect to the number of requests and potential free bus slots. Lemmas 4 and 5 present the theoretical foundation for our pruning mechanisms. They establish two relations between a pair of request-set mappings which, if satisfied,

allows one of the mappings be pruned without risk of discarding the mapping leading to the worst-case cumulative delay. The proofs of the two lemmas are similar and are both based on case enumeration. However, for completeness, they are both provided in the appendix.

Lemma 4 *Let* $\mathbb{M}_i = \{\sigma_i(1), \ldots, \sigma_i(k)\}$ *refer to a request-set mapping for the first k requests of task* τ_i. *Let* $\mathcal{D}_i(k)$ *be the maximum cumulative delay for these k requests considering this mapping* $\mathbb{M}_i$, *and let* $\text{srv}_{i,k}$ *be the* absolute *time at which the kth request is served in a scenario leading to this delay* $\mathcal{D}_i(k)$. *Similarly, let* $\mathbb{M}'_i = \{\sigma'_i(1), \ldots, \sigma'_i(k)\}$ *denote another request-set mapping for the first k requests of task* τ_i. *Let* $\mathcal{D}'_i(k)$ *be the maximum cumulative delay considering this mapping* $\mathbb{M}'_i$, *and let* $\text{srv}'_{i,k}$ *be the* absolute *time at which the kth request is served in a scenario leading to this delay* $\mathcal{D}'_i(k)$. *If it holds that*

$$\sigma_i(k) \leq \sigma'_i(k) \tag{9}$$

$$\textit{and } \mathcal{D}_i(k) \leq \mathcal{D}'_i(k) \tag{10}$$

$$\textit{and } \text{srv}_{i,k} + (\sigma'_i(k) - \sigma_i(k)) \times \text{TR} \geq \text{srv}'_{i,k} \tag{11}$$

then for all $h > \sigma'_i(k)$, *assigning an extra request* $\text{req}_{i,k+1}$ *to the h'th free bus slot in both mappings* $\mathbb{M}_i$ *and* $\mathbb{M}'_i$, *i.e.,* $\sigma_i(k+1) = \sigma'_i(k+1) = h$, *leads to*

$$\sigma_i(k+1) = \sigma'_i(k+1) \tag{12}$$

$$\textit{and } \mathcal{D}_i(k+1) \leq \mathcal{D}'_i(k+1) \tag{13}$$

$$\textit{and } \text{srv}_{i,k+1} + (\sigma'_i(k+1) - \sigma_i(k+1)) \times \text{TR} \geq \text{srv}'_{i,k+1} \tag{14}$$

A detailed proof is presented in the Appendix and the interested reader may please refer the same.

Lemma 5 *Under the same conditions as in Lemma 4, if it holds that*

$$\sigma_i(k) \leq \sigma'_i(k) \tag{15}$$

$$\textit{and } \mathcal{D}_i(k) + (\text{srv}'_{i,k} - \text{srv}_{i,k}) \leq \mathcal{D}'_i(k) \tag{16}$$

$$\textit{and } \text{srv}_{i,k} + (\sigma'_i(k) - \sigma_i(k)) \times \text{TR} \leq \text{srv}'_{i,k} \tag{17}$$

then for all $h > \sigma'_i(k)$, *assigning an extra request* $\text{req}_{i,k+1}$ *to the h'th free bus slot in both mappings* $\mathbb{M}_i$ *and* $\mathbb{M}'_i$, *i.e.,* $\sigma_i(k+1) = \sigma'_i(k+1) = h$, *leads to*

$$\sigma_i(k+1) \leq \sigma'_i(k+1) \tag{18}$$

$$\textit{and } \mathcal{D}_i(k+1) + (\text{srv}'_{i,k+1} - \text{srv}_{i,k+1}) \leq \mathcal{D}'_i(k+1) \tag{19}$$

$$\textit{and } \text{srv}_{i,k+1} + (\sigma'_i(k+1) - \sigma_i(k+1)) \times \text{TR} \leq \text{srv}'_{i,k+1} \tag{20}$$

A detailed proof is presented in the Appendix and the interested reader may please refer the same.

The vital inference from the expressions in Lemmas 4 and 5 is that the maximum cumulative delay for the first $(k+1)$ requests of τ_i is higher by using the mapping $\mathbb{M}'_i$

for the first k requests instead of the mapping $\mathbb{M}_i$. Then, since Conditions (12), (13), and (14) are the same as Conditions (9), (10), and (11) (and the corresponding relation holds for the conditions in Lemma 5), the lemmas continue to hold for all subsequent requests $>k+1$ until the last request of τ_i. This means that $\mathbb{M}_i$ can be safely omitted during the computation of Algorithm 1 as it cannot lead to the maximum cumulative delay.

In order to leverage the result of Lemmas 4 and 5, we implement a function "ListReduce(cell(k, j))" at the end of the first inner loop, i.e., "for $j \leftarrow k$ to $\text{UBslot}_i - (\eta_{(i)} - k)$" in Algorithm 1. This function makes sure that $\nexists$ two distinct tuples $e_{k,j}$ and $e'_{k,j}$ in the list of cell(k, j) such that the conditions in Lemmas 4 and 5 hold. Each time such a pair of tuples is found, only the one with the highest cumulative delay is kept while the other is discarded. This is a key addition to the algorithm that *significantly* reduces the number of tuples in cell(k, j).

7 Region-based analysis

As seen in Sect. 2.3, we can obtain more information on the distribution of the requests by dividing the execution of each task into a sequence of sampling regions. For each region, we can derive an upper bound on the number of requests that can be issued by the task within that region. However, Algorithm 1 did not leverage this region-specific information and used only coarse-grained information about the number of requests in the *entire task*, represented by $\eta_{(i)}$. In other words, Algorithm 1 views the input task τ_i as a single region that can issue up to $\eta_{(i)}$ requests, which may result in a pessimistic upper bound.

In contrast, finer-grained regions with a bounded number of requests and duration allows the analysis to narrow down the range of slots to which the requests can be mapped. This region-based analysis has the advantage of limiting the number of possible candidate slots that must be explored, which decreases the computation time and tightens the analysis. We proceed by elaborating on the theoretical foundations of the analysis, followed by a detailed description of the algorithm.

7.1 Theoretical foundation

When a task is divided into regions and runs in conjunction with other tasks, the time at which each of its regions starts executing depends on the delays incurred by the requests issued in its previous regions. This raises questions about what the worst-case starting time of a region is. Lemma 6 below expresses a relation that exists between the starting time of a region and the maximum delay that it can incur. In essence, it shows that any region that incurs the maximum delay by starting at a time t_1 cannot finish later than if it had started at its maximum starting time t_2. This property enables a fine-tuned WCET analysis in which the distribution of requests across regions is exploited to obtain region-accurate estimates.

Lemma 6 *Let g be a region of a task τ_i that starts at time t and finishes at time f after incurring its maximum blocking delay d. It holds that any earlier starting time $t' < t$ for region g results in a maximum finishing time $f' \leq f$.*

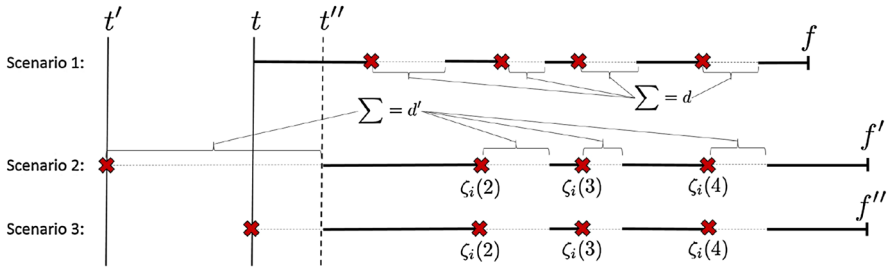


Fig. 5 Illustration of Scenarios 1, 2, and 3 used in the proof of Lemma 6

Proof The proof is obtained by contradiction. Let us assume two execution scenarios for region g. In Scenario 1, region g starts executing at time t and finishes at time f after incurring its maximum blocking delay d whereas in Scenario 2, it starts at time $t' < t$ and finishes at time $f' > f$ (it thus incurs a delay $d' > d$). That is, in Scenario 2 region g starts its execution earlier and finishes later than in Scenario 1. We show by contradiction that Scenario 2 is impossible. To do so, two cases must be explored: In Scenario 2,

Case 1 region g releases its first request **at or after** time t.

Case 2 region g releases its first request **before** time t.

Case 1 In this case, the request-to-slot assignments that led to the blocking delay d' in Scenario 2 can also be used in Scenario 1, since the available free slots are the same in both scenarios. This would result in a delay d equal to d' in Scenario 1 and since $t > t'$ we get $f > f'$, which contradicts the initial assumption that $f < f'$.

Case 2 In this case, region g releases its first request before time t. Figure 5 illustrates the two scenarios in such a situation. An "X" represents the release of a request, a continuous line represents the execution of the region, and a dashed line is an interval of time during which the task stalls, waiting for its last request to be served. It is assumed in this illustration that region g generates a maximum of $\eta_{i,g} = 4$ requests.

Suppose that in Scenario 2, region g incurs the maximum delay of $(t - t')$ in the time-interval $[t', t]$, by releasing a single request at the very beginning of its execution. The delay incurred by this single request can even extend until time $t'' > t$, as depicted in Scenario 2 in Fig. 5. This situation can easily be shown to be a worst case for Scenario 2 (with respect to its finishing time), as it generates the maximum delay with the fewest requests and it delays the actual workload of $L_i^{\text{reg-size}}$ units of execution as much as possible.

Now, let us denote by $\{\sigma_i(2), \ldots, \sigma_i(\eta_{i,g})\}$ the request-set mapping of the $(\eta_{i,g} - 1)$ last requests of region g in Scenario 2 (note that, unlike what is depicted in Fig. 5, the mapping of these requests may be the same as in Scenario 1). We can create a third scenario, in which region g starts its execution at time t (as in Scenario 1) and such that its first request is released at the beginning of its execution, thereby incurring the same delay between $[t, t'']$ as in Scenario 2, and all the subsequent requests follow the same request-to-slot assignments as in Scenario 2, thereby incurring again the same delay as in Scenario 2. In this new Scenario 3, it thus holds that region g starts at time t and finishes at time $f'' = f' > f$, which contradicts our initial assumption defining d as the maximum delay that region g can incur when starting at time t.

In short, we showed in this proof that for any scenario in which a given region g starts before a time t, releases requests before that time t, and finishes at a time f' (like Scenario 2 in our proof), we can create a corresponding scenario (like Scenario 3 here) in which region g starts at time t and finishes at time f' as well. Therefore, if we compute the maximum delay for a given region and a given starting time, it gives us a maximum finishing time for that region that cannot be earlier than in a scenario where the region starts earlier, which proves the lemma. □

The important inference from Lemma 6 is that the WCET of a task (considering contention) can be determined by computing the worst-case finishing time f_1 of its first region, and then iterating over the subsequent regions, assuming for each region g, a starting time of f_{g-1}. The WCET of the entire task is then given by the worst-case finishing time of its last region. This is exploited in our algorithm for region-based analysis, presented next.

7.2 Algorithm for region-based analysis

With Algorithm 2, we propose an arbiter-independent method to determine the worst-case cumulative delay. It is basically an extension of Algorithm 1 that augments it with region-based information. Since the inputs to this algorithm are the $\mathcal{T}_i^{\min}()$, $\mathcal{T}_i^{\max}()$ functions and the details of the analyzed task, any arbiter for which these values can be determined can leverage this algorithm.

Algorithm 2: ComputeTaskWCET(τ_i)

input : τ_i
output: WCET of τ_i (considering contention)

1 $w_i = \frac{C_i}{L_i^{\text{reg-size}}}$;
2 **for** *region g in task* τ_i *from* 1 *to* w_i **do**
3 $\quad \eta_{i,g} \leftarrow$ No of requests in region g;
$\quad \text{UBTime}_{i,g} \leftarrow f_{i,g-1} + L_i^{\text{reg-size}} + \eta_{i,g} \cdot \mathcal{T}_i^{\max}(1)$;
$\quad$ `// with` $f_{i,0} = 0$
$\quad$ `// Find the earliest slot for which` $\mathcal{T}^{\max}()$ `is greater that the finishing time of the previous region`
4 $\quad \text{LBslot}_{i,g} \leftarrow \min_{x>0}\{x \mid \mathcal{T}_i^{\max}(x) \geq f_{i,g-1}\}$;
$\quad$ `// Find the earliest slot for which` $\mathcal{T}^{\min}()$ `is greater that the coarse upper bound of the current region`
5 $\quad \text{UBslot}_{i,g} \leftarrow \min_{x>0}\{x \mid \mathcal{T}_i^{\min}(x) \geq \text{UBTime}_{i,g}\}$;
6 $\quad \delta_{i,g} = \text{MaxRegDelay}(\eta_{i,g}, \text{LBslot}_{i,g}, \text{UBslot}_{i,g})$;
7 $\quad f_{i,g} = f_{i,g-1} + L_i^{\text{reg-size}} + \delta_{i,g}$;
8 **end**
9 **return** f_{i,w_i};

The algorithm commences by computing the number w_i of regions (Line 1) and then considers each region g successively (Line 2), which was shown to be safe by Lemma 6. Next, given the number $\eta_{i,g}$ of requests in the analyzed region g, it finds

a *coarse* upper bound on its increased execution time $\text{UBTime}_{i,g}$ assuming that each request in region g may incur a delay of $T_i^{\max}(1)$. Then, it computes the range of the free bus slots that the requests of region g may occupy (Lines 5–6), assuming on Line 5 a starting time of $f_{i,g-1}$.

To compute the worst-case delay of each region, the algorithm invokes a slightly modified version of Algorithm 1 in which:

1. j now spans from $k + \text{LBslot}_{i,g}$ to $\text{UBslot}_{i,g} - (\eta_{i,g} - k)$, assuming that $\text{LBslot}_{i,g}$ is passed to Algorithm 1 as an additional input parameter and k is the slot index in the algorithm on Line 4,
2. the 2D array contains $\text{UBslot}_{i,g} - \text{LBslot}_{i,g}$ columns,
3. all the references to a cell(k, j) are replaced with a reference to cell$(k, j - \text{LBslot}_{i,g})$, and
4. references to C_i are substituted for references to $L_i^{\text{reg-size}}$.

Note that a task modeled as a single region is a special case in which $\text{LBslot}_{i,1} = 1$, the region size $L_i^{\text{reg-size}}$ is C_i, and the maximum number of requests is $\eta_{(i)}$. The delay of the currently analyzed region $\delta_{i,g}$ is computed on Line 7 and is then accounted for in the worst-case finishing time $f_{i,g}$ computed on Line 8. The process is repeated for all the regions and the finishing time of the last region gives the WCET of the task including the maximum cumulative delay for accesses to shared resources.

It can be seen that for two consecutive regions g and $(g + 1)$, the ranges of candidate free bus slots $\left[\text{LBslot}_{i,g}, \text{UBslot}_{i,g}\right]$ and $\left[\text{LBslot}_{i,g+1}, \text{UBslot}_{i,g+1}\right]$ computed at Lines 5 and 6 may overlap. As a result, the finishing time f_{i,w_i} that is returned by the algorithm may sometimes consider a request-set mapping of all the requests in which two requests from two different regions of the task are assigned to a same free bus slot. Even though it may lead to pessimistic (i.e. over-approximated) results, it is safe and *sometimes necessary* for our region-based analysis technique to work with this assumption, because the aggregation of local maximum delays (local to each region) does not always lead to a global maximum delay for the entire task. To illustrate that claim, let us consider an intuitive example: consider a task τ_i with only 2 regions where each one can generate up to 2 requests. As depicted in Fig. 6, their respective ranges of candidate

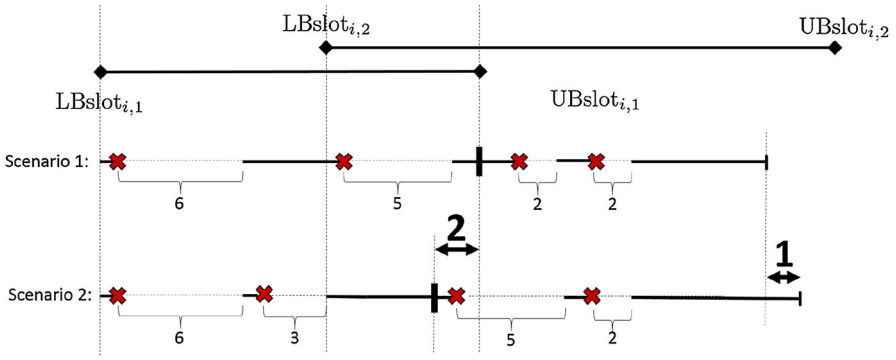


Fig. 6 Example of overlapping regions

free bus slots overlap. Since we do not assume any specific shape for the functions $\mathcal{T}_i^{\min}()$ and $\mathcal{T}_i^{\max}()$, let us assume that these two functions are defined such that:

1. there is a free bus slot within $\left[\text{LBslot}_{i,1}, \text{LBslot}_{i,2}\right]$ that can generate a maximum delay of 6 and all the other slots in that interval generate a delay no greater than 3.
2. all the free bus slots that are available to both regions, i.e. within $\left[\text{LBslot}_{i,2}, \text{UBslot}_{i,1}\right]$, generate a maximum delay of 5.
3. all the free bus slots within $\left[\text{UBslot}_{i,1}, \text{UBslot}_{i,2}\right]$, generate a maximum delay of 2.

With these assumptions, let us create two different request-set mappings for the 4 requests of τ_i. We call these two mappings: Scenarios 1 and 2 (see Fig. 6). In Scenario 1, the *local* maximum delay for the first region is 11 and it is obtained by assigning its first request within $\left[\text{LBslot}_{i,1}, \text{LBslot}_{i,2}\right]$ to get the maximum delay of 6 and then assign its second request to one of the free bus slots within $\left[\text{LBslot}_{i,2}, \text{UBslot}_{i,1}\right]$ to get a delay of 5. By doing so, the second region that starts after the completion of the first one is only able to get a total delay of 4, which leads to an overall delay of $6+5+2+2=15$ for Scenario 1. In contrast, Scenario 2 assigns the second request of the first region to another time slot within $\left[\text{LBslot}_{i,1}, \text{LBslot}_{i,2}\right]$, which leads to a delay of 9 time units for that first region. Even though a delay of 9 for the first region is not a local maximum, it enables the second region to start its execution earlier and benefit from a first delay of 5 time units, by assigning its first request to a slot within $\left[\text{LBslot}_{i,2}, \text{UBslot}_{i,1}\right]$, and a second delay of 2 by assigning its second request to a free bus slot within $\left[\text{UBslot}_{i,1}, \text{UBslot}_{i,2}\right]$. In this second scenario, the overall delay is 16, which is higher than in the first scenario. This is why any region-based analysis technique that is based on analyzing each task region separately must be aware that the global maximum delay cannot be obtained by simply adding up the maximum delays local to each region. In order to make the aggregation of local maxima possible (i.e. safe and correct), we allow the ranges of candidate free bus slot slots of each region to overlap. It can be shown that by doing so, the resulting maximum delay obtained for the entire task is pessimistic but it is safe.

7.3 Reducing time-complexity

Algorithm 1 computes the maximum delay that a given task τ_i may incur with a non-polynomial time-complexity. This non-polynomiality is due to the exponential growth in the number of request-set mappings that each cell(k, j) holds when k and j increase. Specifically, the number of request-set mappings listed in $c(k, j)$ is equal to the number of mappings in $c(k, j-1)$, see Line 14, plus the number of mappings in $c(k-1, j-1)$; For each mapping of $c(k-1, j-1)$ considered at Line 15, one mapping is indeed added to $c(k, j)$ in Line 20. Although Lemmas 4 and 5 provide two mechanisms that potentially reduce the number of mappings carried on from one iteration of the algorithm to the next, they do not provide any guarantee on the number of mappings that they will be able to discard and hence, they do not reduce the theoretical time-complexity. It is important to highlight that the algorithm proceeds in phases: Once $\mathcal{T}^{\min}()$ and $\mathcal{T}^{\max}()$ values are pre-computed for the given arbiter,

they are accessed in constant time in the algorithm and therefore the complexity of Algorithm 1 is agnostic to the underlying arbiter.

A simple way to reduce the complexity is to set up an upper limit on the number of mappings that each cell can hold. Let us denote this limit by L. At run time, at the end of each iteration in the inner loop (between Lines 23 and 24), we add a simple test that counts the number X of mappings of the current cell cell(k, j). If this number X is lower than our pre-set limit L then the algorithm proceeds with the next cell. Otherwise, the algorithm takes all the mappings of $c(k, j)$ and collapses them all into a single "dummy" mapping that contains the *maximum* cumulative delay, the *minimum* service time, and the *latest* request-to-slot assignment among the delays, service times, and request-to-slot assignments of all the collapsed mappings. It can be easily showed by looking at Algorithm 1 that this choice of parameters for the dummy mapping are the worst, in the sense that those parameters will lead to the maximum resulting delay incurred by the analyzed task τ_i. With this technique, the time-complexity of Algorithm 1 is reduced to $O(L \times \eta_{(i)} \times \text{UBslot}_i)$ at the cost of adding pessimism in the computation, since the dummy mappings retain only the worst parameters that may come from different mappings. This trick hence represents a programmable trade-off between computation time and accuracy of the proposed analysis.

Our experimental evaluation uses an implementation of Algorithm 1 that integrates the two optimization mechanisms provided by Lemmas 4 and 5 as well as the region-based analysis detailed in Sect. 7. Our implementation currently does not use this method, as it successfully ran and analyzed all the benchmark programs used in the experiments in a reasonable time. Therefore, we only present this method here in order to show that there exist solutions to a (theoretical) complexity issue that may potentially arise when running the analysis, but we did not investigate these solutions further as we have not experienced such problems in our simulations. Also we have not yet evaluated the increased pessimism that the 'dummy mappings' could introduce at this point and will consider it for future work.

8 Related work

Several frameworks, such as Real-Time Calculus (Thiele et al. 2000) and Network Calculus (Cruz 1991) have been proposed for general delay analysis of shared resources, such as tasks executing on processors, network packets and memory requests. These frameworks typically compute delays as the maximum difference between worst-case supply and demand functions. Since the frameworks are general, it is up to the user to derive appropriate supply and demand functions for a particular problem. A key challenge addressed by our approach that is not covered by existing literature on Real-Time Calculus or Network Calculus is that the worst-case demand function is not exactly known, as we only have coarse-grained information about the number of requests from the sampling regions. A main innovation of our work can hence be described as determining what the worst-case demand function actually looks like, given these sampling regions and information about the arbiter. This in turn enables us to compute the maximum cumulative delay.

The specific topic of bus contention analysis has received considerable attention in recent years and these efforts can be classified into two classes: (1) approaches that modify the hardware or the software of the system to enable or improve analysis, and (2) approaches that analyze a given system without assuming any modification of the hardware and/or software. We proceed by discussing each of these in turn.

On the hardware side, a number of memory controllers have been designed specifically for real-time systems and proposed together with corresponding analyses that bound the WCRT of memory requests (Akesson and Goossens 2011; Reineke et al. 2011; Paolieri et al. 2009; Shah et al. 2012; Wu et al. 2013; Li et al. 2014). These analyses benefit from full knowledge of the internals of the memory controller, such as page policies, transaction scheduler and the DRAM command scheduler, and exploit this information to produce tight bounds. On the software side, servers with memory budgets, built into the operating system, have been proposed to limit the memory interference (Nowotsch et al. 2014; Yun et al. 2012; Behnam et al. 2013; Yun et al. 2013) from tasks executing on other cores, enabling it to be managed based on enforcement rather than characterization. Our work contrasts to these efforts in the sense that it considers both the software and hardware to be given with no interface or any other means to modify/re-configure it.

Several approaches have been proposed for bus contention analysis in given COTS platforms. Similarly to our work, most analyses consider multi-core systems with a bus providing access to a shared memory with a single port (Schliecker et al. 2010; Schliecker and Ernst 2011; Pellizzoni et al. 2010; Dasari et al. 2011; Dasari and Nelis 2012; Chattopadhyay et al. 2014; Rodrigues et al. 2013). However, these works are quite different with respect to the considered task models and scheduling policies for both the tasks themselves and their memory requests.

Applications are typically modeled as independent periodic/sporadic task sets or acyclic task graphs (Rosén et al. 2007; Chattopadhyay et al.2010), and the scheduling is often based on fixed-priorities (Dasari et al. 2011; Schliecker and Ernst2011), while tasks in task graphs are *statically scheduled* using techniques that respect precedence constraints, e.g. list scheduling. The approaches support different task preemption models, ranging from fully preemptive (Schliecker et al. 2010; Schliecker and Ernst 2011) to non-preemptive (Dasari et al. 2011; Dasari and Nelis 2012; Rosén et al. 2007; Chattopadhyay et al. 2010), and with limited-preemption at the granularity of TDM time slots as a compromise in between (Pellizzoni et al. 2010). Most of these works consider analysis of shared resources as a separate analysis, while (Chattopadhyay et al. 2014; Rodrigues et al. 2013) integrate it into the WCET estimation tool to exploit information about the execution of the application, such as when memory requests are issued.

A problem with most of the previously mentioned analysis approaches is that they only support a single bus arbiter, such as an unspecified work-conserving arbiter (Dasari et al. 2011; Schliecker et al. 2010), fixed-priority arbitration, round robin (Dasari and Nelis 2012), TDM (Rosén et al. 2007; Chattopadhyay et al. 2010, 2014; Schranzhofer et al. 2010, 2011) or first-come first-served (FIFO). This does not address the diversity of memory arbiters in contemporary platforms, making them point-solutions exclusive to a single platform rather than a reusable framework that applies more generally.

Of particular interest is the work in Schranzhofer et al. (2010), which computes the worst-case completion time for tasks accessing a resource shared by a TDM arbiter. Unlike our approach, the arbiter grants access to the resource in a coarse-grained manner, where each bus slot has a fixed longer duration that typically fits many requests. Similarly to the naive approach of using a constant worst-case delay for every memory request, the proposed assumes that each request is issued at the worst-case time just at the end of the allocated TDM slot, maximizing the delay. However, it outperforms a naive analysis by considering the maximum number of allocated TDM slots during which the application can execute without delay when accessing memory. In case this number is less than the number of memory requests, the remaining number of memory requests can be efficiently fetched one after the other during the following allocated TDM slots, resulting in less pessimistic bounds. The approach hence capitalizes on the fact that the execution requirements of the task may prevent the worst-case situation from happening to all memory requests. We compare our generic method with this approach in the next section.

9 Experimental evaluation

This section experimentally evaluates the proposed framework by simulating a multi-core system running real application traces. First, the experimental setup is explained, followed by three experiments. The first experiment demonstrates the generality of our approach by executing the applications with three different arbiters and evaluating the computation time of the proposed analysis. The second experiment evaluates the impact of different region sizes and shows how finer-grained task region-profiles improve the accuracy and increase the efficiency of the analysis. Lastly, the final experiment compares our framework to a state-of-the-art analysis approach for TDM arbitration, i.e. the approach proposed in Schranzhofer et al. (2010).

9.1 Experimental setup

The experiments consider a multi-core platform, where the processors are simulated by the SimpleScalar 3.0 processor simulator (Austin et al. 2002) with separate data and instruction caches, each with a size of 16 KB. The L2 cache is a private unified 128 KB cache with 64 B cache lines and an associativity of 4. The processor core is assumed to run at a frequency of 1.6 GHz. The memory is a 64-bit DDR3-1600 DIMM (JEDEC 2012) running at a frequency of 800 MHz, meaning that one memory cycle equals two processor cycles. The memory access time is $TR = 80$ processor cycles, corresponding to an in-order dynamically scheduled DRAM controller with a close-page policy (Li et al. 2014). The experiments consider a platform instance with 4 cores, each core running an application from the MediaBench test suite (Lee et al. 1997) as a single independent task. For each application in the benchmark, memory-trace files were generated by running it on the experimental platform. The traces were then post-processed according to the sampling regions used in the experiments to compute the region-profiles of the task. The experiments were executed on a computer equipped with Intel Core I5 processor (2.0 GHz, 4 cores) and 4 GB memory.

Table 1 Benchmark characterization

Benchmark	Exec. time [Kcycles]	Requests	Request density
unepic	15,775	67,664	4.290
jpeg-encode	46,160	92,905	2.013
epic	62,540	96,984	1.551
jpeg-decode	21,417	22,121	1.033
h263-encode	566,845	418,808	0.738
h263-decode	8462	5456	0.645
mpeg2encode	823,274	319,306	0.388
gsmdecode	43,012	10,104	0.235
mpeg2decode	100,454	28,744	0.286
adpcmdecode	4193	575	0.137
adpcmencode	6358	581	0.091
g721-decode	172,563	9792	0.057
g721-encode	152,829	7439	0.049

The essential characteristics of the benchmark applications, which we use in the evaluation set are shown in Table 1. Note that the table does not contain all applications from the suite, as some of them would not compile with the SimpleScalar toolchain and others would not provide functionally correct output upon verification. Instead of changing the code of these applications and thereby defeating the purpose of standard benchmarks, we opted to exclude these applications. The table shows the execution time of the used applications on the SimpleScalar processor, the total number of memory requests during its execution, and lastly the request density (requests per cycle) as a measure of its memory intensity. It is clear from the table that the chosen benchmark applications are not trivial as the execution times are typically several million cycles during which thousands of requests are issued. *This highlights the scalability of our approach and contrasts to previous work that use much smaller applications from the CHStone* (Hara et al. 2008) *and Malardalen WCET benchmarks* (Gustafsson et al. 2010).

9.2 Application to different arbitration mechanisms

The objective of this experiment is to demonstrate the generality of our approach by applying it to three commonly-used arbiters, being fixed-priority, an unspecified work-conserving arbiter, and TDM, respectively. For each task, we determine the interference from other tasks and compute the increase in WCET for each of the three arbiters using a region size of 20 Kcycles. Other region sizes are evaluated in the following experiment. We also examine the computation time of the proposed analysis for the different arbiters. To get a representative sample of applications for the WCET benchmark, we chose 4 applications considering the 2 highest and the 2 lowest densities as shown in Table 1.

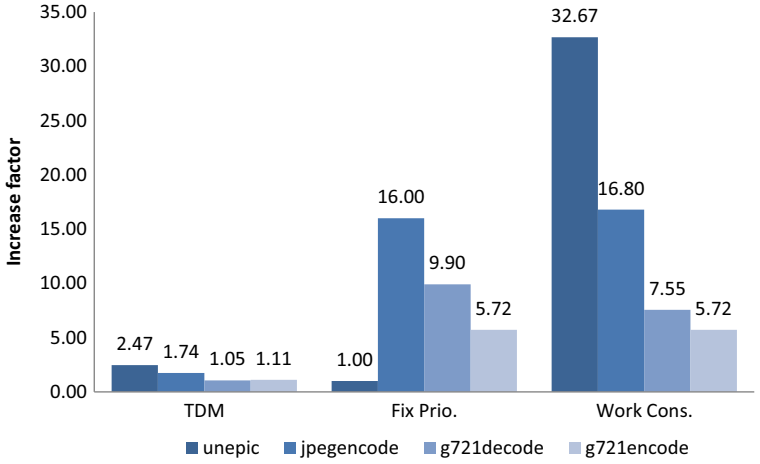


Fig. 7 Increase in WCET for different arbitration mechanisms

The results of the experiment are shown in Fig. 7, where tasks are arranged in descending order of priorities (*unepic* has the highest priority) for the case of fixed-priority arbitration. As expected, for the fixed-priority scheduler the task with the highest priority experiences no interference (an increase factor of 1x) from the other tasks. We observe a counter-intuitive effect in that *jpegencode* (priority 2) experiences a larger increase in WCET than the lower priority tasks. This is because *jpegencode* has higher request density than the two lower priority tasks, implying that it is more memory intensive. Despite having lower delay per memory access due to the higher priority, this results in higher impact of the cumulative delay on the increase factor.

For the unspecified work-conserving arbiter, the requests of a given task may be blocked by all requests from all concurrently executing tasks. During this time, more requests can be injected by other tasks, thereby further blocking the requests of the analyzed task. Hence, $\mathcal{T}_i^{\max}(1)$ is very high for a task blocked by a memory-intensive task. This increases the possible number of slots in which requests of the analyzed task may fit and many of these requests may incur a delay of $\mathcal{T}_i^{\max}(1)$, leading to a high WCET estimate, as seen in the figure.

Note that this arbitration mechanism is equivalent to fixed-priority arbitration where every task is assumed to have the lowest priority. This can be seen in Fig. 7, where the lowest priority task, *g721encode*, has the same WCET with fixed-priority arbitration and the unspecified work-conserving arbiter. It is interesting to note that for *jpegencode*, there is only a minor difference in the increase factor in the two arbitration mechanisms. This is because the other two low-density tasks affect its performance very marginally and the major interference is still from the task *unepic*.

Unlike the previous two arbiters, TDM is neither priority-based, nor work conserving. Here, it is configured with a frame size of 24 and each of the four cores is allocated 6 slots. Other configurations are evaluated in the final experiment. We note from the results that TDM arbitration performs remarkably well compared to fixed-priority arbitration, as only the highest priority task has a smaller increase factor using

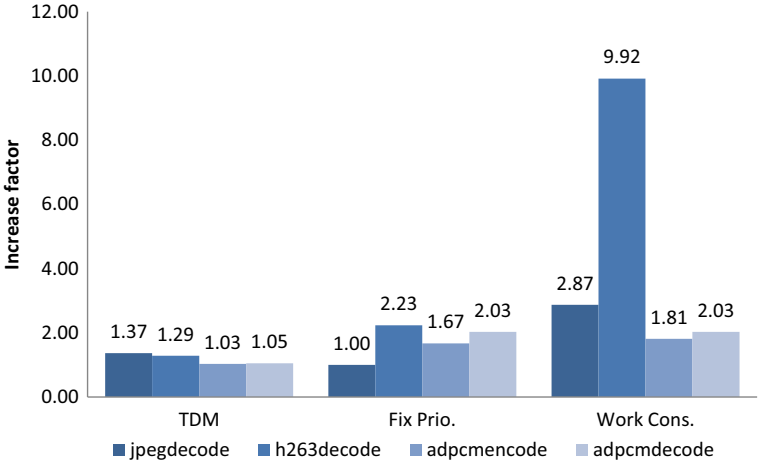


Fig. 8 Increase in WCET for different arbitration mechanisms

fixed-priorities. An explanation for this is that the worst-case per-core request-profile (PCRP) computation for the interfering cores becomes quite pessimistic as it determines the worst possible release time of all higher-priority requests in the region. On the other hand, the worst-case for TDM is independent of the release times of other tasks and the only uncertainty comes from the possible misalignment between the release of the requests and the TDM table, which is bounded by the frame size.

Similar trends are seen when the three arbitration mechanisms are applied to another set of benchmarks as shown in Fig. 8.

Considering the computation time of the analysis, the fixed-priority arbitration took 8 h to complete for the entire task set. The tasks with higher priorities complete faster than the slower ones, since they are less impacted by interference, resulting in fewer possible request-set mappings. This is reflected in the analysis of the unspecified work-conserving arbiter, where all tasks can suffer interference from all other tasks, increasing the analysis time to around 15 h for the entire task set. In contrast, the TDM arbiter is non-work-conserving and thereby completely independent of other tasks, enabling the computation of $\mathcal{T}_i^{\min}()$ and $\mathcal{T}_i^{\max}()$ in constant time. Furthermore, small TDM frame sizes provide relatively few possible request-set mappings, reducing the total analysis time to less than 45 min.

9.3 Impact of the region size

We also evaluated the impact of the region sizes. To this end, we re-ran the previous experiments using both smaller and larger region sizes. Three different sizes are used: 30, 40 and 50 Kcycles, respectively, where larger region sizes imply fewer regions and coarser-grained request profiles for each region. We choose two benchmarks with a high request density, *epic* and *mpeg2-decode*, and two benchmarks that have low request density, *g721encode* and *g721decode*. The results of the experiment using the

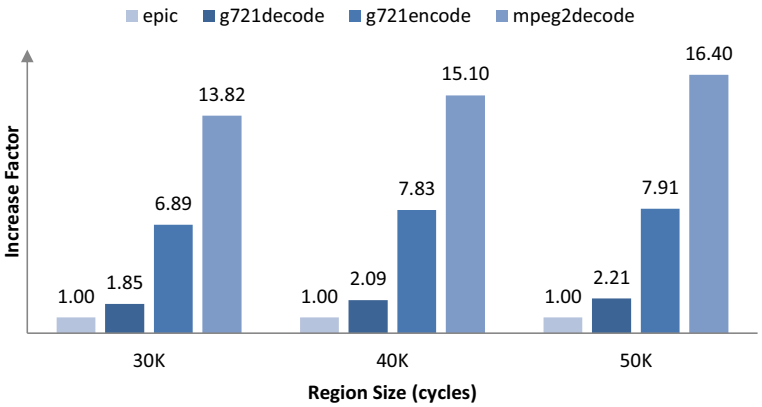


Fig. 9 Increase in WCET for different region sizes (in cycles) using fixed-priority arbitration. The highest priority is given to the task *epic* followed by *g721decode*, etc

fixed-priority arbiter are shown in Fig. 9. Note that the highest priority task, *epic*, experiences no interference across all region sizes. For the other tasks, the results generally follow the intuition that smaller regions result in tighter WCET. This is because finer-grained task-region profiles provide more information about the actual request distribution, eliminating unfeasible request releases in the PCRP computation (see Dasari and Nelis 2012 for details about the PCRP computation).

Results similar to those in Fig. 9 were also observed for the unspecified work-conserving arbiter, whose analysis is very similar and also relies on the PCRP computation. However, the TDM arbiter is largely insensitive to changes in the region size. The explanation for this is that it is independent of the behavior of the other clients and does not benefit from finer-grained information about their behavior.

9.4 Comparison against the state-of-the-art

The final experiment compares our approach against the state-of-the-art. Given that no other unified framework exists for comparison, we compared our approach against the approach in Schranzhofer et al. (2010), specifically targeting TDM arbiters. The results are presented in Table 2 and show the percentage increase in WCET considering the cumulative delay to serve all the memory requests in the available slots. The comparison was done for different TDM configurations, where different number of consecutive slots, ϕ_p, are allocated to four tasks executing on the different cores. The frame size in this experiment is hence equal to $f = 4 \times \phi_p$.

The results indicate that our approach provides tighter worst-case estimates than the previous approach for most of the configurations. The cases in which the SOA performs better, are highlighted in bold, and it can be seen that the difference is marginal. Both approaches perform better with a smaller number of allocated slots, since this reduces the worst-case misalignment between a request release and the next slot allocated to the processor running the task.

Table 2 Comparison of the increase factor between our unified framework (UF) and the state-of-the-art (SOA) (Schranzhofer et al. 2010).

Benchmark	$\phi_p = 1$		$\phi_p = 5$		$\phi_p = 10$	
	UF	SOA	UF	SOA	UF	SOA
unepic	**2.37**	**2.36**	2.45	4.33	2.53	4.59
jpeg-encode	1.64	1.64	1.73	3.35	1.82	4.35
epic	**1.50**	**1.49**	1.57	2.59	1.65	3.27
adpcm-decode	1.04	1.04	1.05	1.08	1.05	1.12
adpcm-encode	1.03	1.03	1.03	1.06	1.03	1.08
g721-decode	1.01	1.02	1.04	1.07	1.08	1.15
g721encode	1.01	1.01	1.05	1.07	1.06	1.11
gsmdecode	**1.08**	**1.07**	1.11	1.29	1.15	1.52
h263decode	**1.21**	**1.20**	1.27	1.69	1.34	2.02
h263encode	1.29	1.29	1.34	1.99	1.40	2.38
jpegdecode	1.33	1.33	1.36	2.11	1.39	2.15
mpeg2decode	1.12	1.12	1.15	1.45	1.19	1.73
mpeg2encode	1.12	1.12	1.14	1.44	1.18	1.58

10 Conclusions

The necessity of deriving tight upper bounds on the contention delay due to the shared memory bus is an indispensable prerequisite in order to efficiently compute the worst-case execution time (WCET) of real-time tasks. To maximize the applicability of the analysis, this must furthermore be done in a general way that can easily be applied to the diversity of arbiters in modern systems.

This work proposed a general framework to address this problem. A key novelty of this framework is a general bus availability model that seamlessly allows different arbiters to be analyzed using a simple interface. We demonstrated how to use this interface to characterize a fixed-priority arbiter, time-division multiplexing (TDM), and explained how these characterizations are modified to also cover round robin and an unspecified work-conserving arbiter. The bus availability model was then leveraged by an arbiter-independent analysis to compute the WCET of a task when co-scheduled with other tasks contending on the bus. A key feature of this analysis is that it allows information about memory requests to be provided in multiple smaller regions to speed up the analysis while improving its accuracy.

We experimentally demonstrated that the approach addresses the diversity problem by applying it to three different arbiters using applications from the MediaBench suite. The scalability of the analysis was shown as the analysis completed in 45 min to 15 h depending on the arbiter when considering four concurrently executing applications with execution times of several million clock cycles during which they issue thousands of requests. We also evaluated the impact of region size for fixed-priority arbitration and showed how finer-grained information about requests improve the accuracy of the analysis. Lastly, we evaluated the quality of the analysis by comparing it with a state-of-

the-art approach to TDM analysis and showed that our approach consistently resulted in lower WCET of the analyzed applications for the considered TDM configurations.

Acknowledgments This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within project UID/CEC/04234/2013 (CISTER Research Centre); by FCT/MEC and the EU ARTEMIS JU within project ARTEMIS/0001/2013—JU grant nr. 621429 (EMC2); by the North Portugal Regional Operational Programme (ON.2—O Novo Norte) under the National Strategic Reference Framework (NSRF), through ERDF, and by National Funds through FCT/MEC, within project NORTE-07-0124-FEDER-000063 (BEST-CASE, New Frontiers); and by the European Union under the Seventh Framework Programme (FP7/2007-2013), grant agreement n° 611016 (P-SOCRATES) and by the European social fund within the framework of realizing the project "Support of inter-sectoral mobility and quality enhancement of research teams at Czech Technical University in Prague", CZ.1.07/2.3.00/30.0034.

Appendix

This appendix contains the proofs of Lemmas 3, 4, and 5. These proofs are all quite long as they are all based on case enumeration. Although many of the cases in the proof are similar, some even identical, the presented proofs cover all cases exhaustively for completeness.

Proof of Lemma 3

Proof We prove the lemma by induction. First, we show in the basic step that the claim is true considering only the first request $\text{req}_{i,1}$ and its slot assignment $\sigma_i(1)$. That is, we show that the release and service times given by Eqs. (7) and (8) result in a maximum cumulative delay $\mathcal{D}_i(1) = \text{srv}_{i,1} - \text{rel}_{i,1}$. Then, in the inductive step, we show that if the claim is true considering the set of the first k requests, $k \geq 1$ (induction hypothesis), then the property holds for the first $(k+1)$ requests as well. In other words, assuming that Eqs. (7) and (8) assign a release and service time to the k first requests that result in a maximum cumulative delay $\mathcal{D}_i(k)$, then the same equations provide a maximum cumulative delay $\mathcal{D}_i(k+1)$ when applied to the first $(k+1)$ requests. Both the basic and inductive steps are proven by showing that any other choice of release and service time, for any of the requests in the considered set of requests, results in a lower cumulative delay.

Basic step By considering only the first request $\text{req}_{i,1}$, it is easy to see that any release time $\text{rel}_{i,1}$ different from that given by Eq. (7) leads to $\text{rel}_{i,1} > \mathcal{T}_i^{\min}(\sigma_i(1)-1)+1$. This follows from the fact that having $\text{rel}_{i,1} < \mathcal{T}_i^{\min}(\sigma_i(1)-1)+1$ is not possible, as shown in Lemma 1. Besides, choosing any other release time $\text{rel}_{i,1} > \mathcal{T}_i^{\min}(\sigma_i(1)-1)+1$ would have as sole impact, a decrease in the difference $(\text{srv}_{i,1} - \text{rel}_{i,1})$, and subsequently a lower delay $\mathcal{D}_i(1)$ incurred by request $\text{req}_{i,1}$. In short, since $\mathcal{T}_i^{\min}(\sigma_i(1)-1)+1$ is a lower bound on the release time of request $\text{req}_{i,1}$ (from Lemma 1), choosing $\text{rel}_{i,1} = \mathcal{T}_i^{\min}(\sigma_i(1)-1)+1$ is the best choice to guarantee a maximum delay for the first request. Similarly, since $\min(\mathcal{T}_i^{\max}(\sigma_i(k)), \text{rel}_{i,k} + \mathcal{T}_i^{\max}(k))$ was shown to be an upper bound on the service time of request $\text{req}_{i,k}$, $\forall k$ (see Lemma 2), it is easy to see that the choice of $\text{srv}_{i,1}$ by Eq. (8) also guarantees a maximum delay for this first request. In conclusion, we showed that $\mathcal{D}_i(1) = \text{srv}_{i,1} - \text{rel}_{i,1}$ is maximum when $\text{rel}_{i,1}$ and $\text{srv}_{i,1}$ are given by the equations of Lemma 3.

Inductive step Assuming that Eqs. (7) and (8) define a release and a service time for the first k requests of τ_i such that their cumulative delay $\mathcal{D}_i(k)$ is maximized, we will show that defining $\text{rel}_{i,k+1}$ and $\text{srv}_{i,k+1}$ using the equations of Lemma 3 maximizes $\mathcal{D}_i(k+1)$. By applying the same reasoning as in the basic step, it is evident that choosing any other value of $\text{rel}_{i,k+1}$ greater than its lower bound (given in Lemma 1 and Eq. (7)) and/or any other service time $\text{srv}_{i,k+1}$ lower than its upper bound (given in Lemma 2 and Eq. (8)) induces a lower delay for request $\text{req}_{i,k+1}$, and thus a lower cumulative delay $\mathcal{D}_i(k+1)$.

However, it may be noted from the release-time equation (Eq. 7) that the choice of service time $\text{srv}_{i,k}$ of the previous request $\text{req}_{i,k}$ influences the lower bound on $\text{rel}_{i,k+1}$, and subsequently an upper bound on $\text{srv}_{i,k+1}$ (see Eq. (8)). One should therefore investigate the following question: although choosing $\text{srv}_{i,k} = \min(\mathcal{T}_i^{\max}(\sigma_i(k)), \text{rel}_{i,k} + \mathcal{T}_i^{\max}(1))$ guarantees a maximum cumulative delay $\mathcal{D}_i(k)$ for the first k requests (from the induction hypothesis), doing so might define a range of possible values for $\text{rel}_{i,k+1}$ that discards those leading to a maximum cumulative delay $\mathcal{D}_i(k+1)$. The remainder of this proof consists of showing that any value of $\text{srv}_{i,k}$ different from that given by Eq. (8) results in a lower cumulative delay $\mathcal{D}_i(k+1)$.

To figure out how $\text{srv}_{i,k}$ affects the range of possible values for $\text{rel}_{i,k+1}$ and $\text{srv}_{i,k+1}$, let us consider different values X and Y for $\text{srv}_{i,k}$, where $X = \min(\mathcal{T}_i^{\max}(\sigma_i(k)), \text{rel}_{i,k} + \mathcal{T}_i^{\max}(1))$ (as given by Expression (8)) and Y is any positive number $< X$. We show in the following that $\mathcal{D}_i(k+1)$ is always maximum for $\text{srv}_{i,k} = X$.

We first introduce two symbols for compaction and readability:

$$\mathcal{K}_{k+1}^{\min} \stackrel{\text{def}}{=} \mathcal{T}_i^{\min}(\sigma_i(k+1) - 1) + 1 \text{ (first term in Eq. (5))}$$

$$\Delta_{k+1} \stackrel{\text{def}}{=} (\sigma_i(k+1) - \sigma_i(k)) \times \text{TR} \text{ (parts of second term in Eq. (5))}$$

We know from Lemma 1 that $\text{rel}_{i,k+1} \geq \max(\mathcal{K}_{k+1}^{\min}, \text{srv}_{i,k} + \Delta_{k+1})$ and thus three cases may arise depending on the request-to-slot assignment $\sigma_i(k+1)$ of request $\text{req}_{i,k+1}$ (these three cases are a simple enumeration of all possible "dominance" relations between the three considered terms):

1. $\mathcal{K}_{k+1}^{\min} \leq Y + \Delta_{k+1} < X + \Delta_{k+1}$
2. $Y + \Delta_{k+1} < \mathcal{K}_{k+1}^{\min} \leq X + \Delta_{k+1}$
3. $Y + \Delta_{k+1} \leq X + \Delta_{k+1} \leq \mathcal{K}_{k+1}^{\min}$

We proceed by proving each of these cases.

Case 1 $\mathcal{K}_{k+1}^{\min} \leq Y + \Delta_{k+1} < X + \Delta_{k+1}$

In this case, choosing $\text{srv}_{i,k} = Y$ leads to $\text{rel}_{i,k+1} \geq Y + \Delta_{k+1}$ (from Lemma 1). By setting $\text{rel}_{i,k+1}$ to $Y + \Delta_{k+1}$, we get

$$\begin{aligned}\mathcal{D}_i(k+1) &= \sum_{\ell=1}^{k+1} (\text{srv}_{i,\ell} - \text{rel}_{i,\ell}) \\ &= \sum_{\ell=1}^{k-1} (\text{srv}_{i,\ell} - \text{rel}_{i,\ell}) + (\text{srv}_{i,k} - \text{rel}_{i,k}) + (\text{srv}_{i,k+1} - \text{rel}_{i,k+1})\end{aligned}$$

$$= \sum_{\ell=1}^{k-1} (\text{srv}_{i,\ell} - \text{rel}_{i,\ell}) + Y - \text{rel}_{i,k} + \text{srv}_{i,k+1} - (Y + \Delta_{k+1})$$
$$= \sum_{\ell=1}^{k-1} (\text{srv}_{i,\ell} - \text{rel}_{i,\ell}) + \text{srv}_{i,k+1} - \text{rel}_{i,k} - \Delta_{k+1} \quad (21)$$

On the other hand, choosing $\text{srv}_{i,k} = X$ leads to $\text{rel}_{i,k+1} \geq X + \Delta_{k+1}$ (from Lemma 1). Then, if we set $\text{rel}_{i,k+1} = X + \Delta_{k+1}$ (i.e., the earliest possible release time) then applying the same reasoning as above leads to the same equality, i.e.,

$$\mathcal{D}_i(k+1) = \sum_{\ell=1}^{k-1} (\text{srv}_{i,\ell} - \text{rel}_{i,\ell}) + \text{srv}_{i,k+1} - \text{rel}_{i,k} - \Delta_{k+1} \quad (22)$$

Since (21) = (22), it is correct to claim that choosing $\text{srv}_{i,k} = X$ leads to a worst-case cumulative delay $\mathcal{D}_i(k+1)$.

Case 2 $Y + \Delta_{k+1} < \mathcal{K}_{k+1}^{\min} \leq X + \Delta_{k+1}$

In this case, choosing $\text{srv}_{i,k} = Y$ leads to $\text{rel}_{i,k+1} \geq \mathcal{T}^{\min}(\sigma_i(k+1) - 1)$ (from Lemma 1). Let $\text{rel}_{i,k+1} = \mathcal{T}_i^{\min}(\sigma_i(k+1) - 1)$ (i.e., the earliest possible release time-instant), from a reasoning similar to that above it holds that

$$\mathcal{D}_i(k+1) = \sum_{\ell=1}^{k+1} (\text{srv}_{i,\ell} - \text{rel}_{i,\ell})$$
$$= \sum_{\ell=1}^{k-1} (\text{srv}_{i,\ell} - \text{rel}_{i,\ell}) + Y - \text{rel}_{i,k} + \text{srv}_{i,k+1} - \mathcal{K}_{k+1}^{\min}$$
$$< \sum_{\ell=1}^{k-1} (\text{srv}_{i,\ell} - \text{rel}_{i,\ell}) + Y - \text{rel}_{i,k} + \text{srv}_{i,k+1} - (Y + \Delta_{k+1})$$
$$< \sum_{\ell=1}^{k-1} (\text{srv}_{i,\ell} - \text{rel}_{i,\ell}) - \text{rel}_{i,k} + \text{srv}_{i,k+1} - \Delta_{k+1} \quad (23)$$

On the other hand, choosing $\text{srv}_{i,k} = X$ leads to $\text{rel}_{i,k+1} \geq X + \Delta_{k+1}$ (from Lemma 1). If $\text{rel}_{i,k+1} = X + \Delta_{k+1}$, then the cumulative delay $\mathcal{D}_i(k+1)$ of requests $\text{req}_{,1}, \text{req}_{,2}, \ldots, \text{req}_{,k+1}$ is given by

$$\mathcal{D}_i(k+1) = \sum_{\ell=1}^{k+1} (\text{srv}_{i,\ell} - \text{rel}_{i,\ell})$$
$$= \sum_{\ell=1}^{k-1} (\text{srv}_{i,\ell} - \text{rel}_{i,\ell}) + X - \text{rel}_{i,k} + \text{srv}_{i,k+1} - \mathcal{K}_{k+1}^{\min}$$

$$\geq \sum_{\ell=1}^{k-1}(\mathrm{srv}_{i,\ell} - \mathrm{rel}_{i,\ell}) + X - \mathrm{rel}_{i,k} + \mathrm{srv}_{i,k+1} - (X + \Delta_{k+1})$$

$$\geq \sum_{\ell=1}^{k-1}(\mathrm{srv}_{i,\ell} - \mathrm{rel}_{i,\ell}) - \mathrm{rel}_{i,k} + \mathrm{srv}_{i,k+1} - \Delta_{k+1} \tag{24}$$

Since (24) > (23), we can conclude that the cumulative delay is higher for $\mathrm{srv}_{i,k} = X$.

Case 3 $Y + \Delta_{k+1} \leq X + \Delta_{k+1} \leq \mathcal{K}_{k+1}^{\min}$

In this case, choosing either $\mathrm{srv}_{i,k} = Y$ or $\mathrm{srv}_{i,k} = X$ leads to $\mathrm{rel}_{i,k+1} \geq \mathcal{T}_i^{\min}(\mathrm{srv}_{i,k+1} - 1)$ (from Lemma 1). Therefore, the range of possible values for $\mathrm{rel}_{i,k+1}$ is not affected by the choice of $\mathrm{srv}_{i,k}$ and the maximum cumulative delay is obviously obtained for $\mathrm{srv}_{i,k} = X$. □

10.1 Proof of Lemma 4

Proof The proof must show that given Conditions (9), (10), and (11), Eqs. (12), (13), and (14) hold. From the claim itself, Eq. (12) trivially holds. We stated this equality only for completeness in order to show that the situation after assigning the $(k+1)$'th request is identical to the situation before assigning it. Let us start the proof by introducing some symbols to improve readability:

$$\mathcal{K}_{k+1}^{\min} \stackrel{\text{def}}{=} \mathcal{T}_i^{\min}(\sigma_i(k+1) - 1) + 1$$
$$\mathcal{K}_{k+1}^{\max} \stackrel{\text{def}}{=} \mathcal{T}_i^{\max}(\sigma_i(k+1))$$
$$\Delta_{k+1} \stackrel{\text{def}}{=} (\sigma_i(k+1) - \sigma_i(k)) \times \mathrm{TR}$$
$$\Delta'_{k+1} \stackrel{\text{def}}{=} (\sigma'_i(k+1) - \sigma'_i(k)) \times \mathrm{TR}$$

According to these new symbols and the equations of Lemma 3, the four quantities $\mathrm{srv}_{i,k+1}$, $\mathrm{rel}_{i,k+1}$, $\mathrm{srv}'_{i,k+1}$, and $\mathrm{rel}'_{i,k+1}$ can be re-written as

$$\mathrm{rel}_{i,k+1} = \max\left(\mathcal{K}_{k+1}^{\min}, \mathrm{srv}_{i,k} + \Delta_{k+1}\right) \tag{25}$$
$$\mathrm{srv}_{i,k+1} = \min\left(\mathcal{K}_{k+1}^{\max}, \mathrm{rel}_{i,k+1} + \mathcal{T}_i^{\max}(1)\right) \tag{26}$$
$$\mathrm{rel}'_{i,k+1} = \max\left(\mathcal{K}_{k+1}^{\min}, \mathrm{srv}'_{i,k} + \Delta'_{k+1}\right) \tag{27}$$
$$\mathrm{srv}'_{i,k+1} = \min\left(\mathcal{K}_{k+1}^{\max}, \mathrm{rel}'_{i,k+1} + \mathcal{T}_i^{\max}(1)\right) \tag{28}$$

From (11), it holds that

$$\mathrm{srv}'_{i,k} - \sigma'_i(k) \times \mathrm{TR} \leq \mathrm{srv}_{i,k} - \sigma_i(k) \times \mathrm{TR}$$

By adding $h \times \mathrm{TR}$ to both sides of this inequality, we get

$$\mathrm{srv}'_{i,k} + (h - \sigma'_i(k)) \times \mathrm{TR} \leq \mathrm{srv}_{i,k} + (h - \sigma_i(k)) \times \mathrm{TR}$$

and the symbols Δ_{k+1} and Δ'_{k+1} can now be used to simplify this result:

$$\text{srv}'_{i,k} + \Delta'_{k+1} \leq \text{srv}_{i,k} + \Delta_{k+1} \tag{29}$$

In order to prove that Inequalities (13) and (14) always hold true, we must investigate three cases. These three cases simply come from an enumeration of all possible "dominance" relations between the three terms $\mathcal{K}^{\min}_{k+1}$, $\text{srv}_{i,k} + \Delta_{k+1}$, and $\text{srv}'_{i,k} + \Delta'_{k+1}$:

- Case 1 $\text{srv}'_{i,k} + \Delta'_{k+1} \leq \text{srv}_{i,k} + \Delta_{k+1} \leq \mathcal{K}^{\min}_{k+1}$
- Case 2 $\text{srv}'_{i,k} + \Delta'_{k+1} \leq \mathcal{K}^{\min}_{k+1} \leq \text{srv}_{i,k} + \Delta_{k+1}$
- Case 3 $\mathcal{K}^{\min}_{k+1} \leq \text{srv}'_{i,k} + \Delta'_{k+1} \leq \text{srv}_{i,k} + \Delta_{k+1}$

Case 1 $\text{srv}'_{i,k} + \Delta'_{k+1} \leq \text{srv}_{i,k} + \Delta_{k+1} \leq \mathcal{K}^{\min}_{k+1}$

In this case, we have

$$\begin{aligned} \text{rel}_{i,k+1} = \text{rel}'_{i,k+1} = \mathcal{K}^{\min}_{k+1} \qquad & \text{from (25) and (27)} \\ \text{and thus } \text{srv}_{i,k+1} = \text{srv}'_{i,k+1} \qquad & \text{from (26) and (28)} \end{aligned}$$

These service times trivially satisfy Condition (14) since $\sigma_i(k+1) = \sigma'_i(k+1) = h$. Then, using $\text{rel}_{i,k+1} = \text{rel}'_{i,k+1}$, $\text{srv}_{i,k+1} = \text{srv}'_{i,k+1}$, and $\mathcal{D}_i(k) \leq \mathcal{D}'_i(k)$ from (10), we get

$$\mathcal{D}_i(k) + \text{srv}_{i,k+1} - \text{rel}_{i,k+1} \leq \mathcal{D}'_i(k) + \text{srv}'_{i,k+1} - \text{rel}'_{i,k+1}$$

This inequality can be re-written as

$$\mathcal{D}_i(k+1) \leq \mathcal{D}'_i(k+1)$$

which satisfies Condition (13).

Case 2 $\text{srv}'_{i,k} + \Delta'_{k+1} \leq \mathcal{K}^{\min}_{k+1} \leq \text{srv}_{i,k} + \Delta_{k+1}$

In this case, from (25) and (27) we have the following relation between the release time-instants of the $(k+1)$'th request in the mappings $\mathbb{M}_i$ and $\mathbb{M}'_i$:

$$\text{rel}_{i,k+1} = \text{srv}_{i,k} + \Delta_{k+1} \geq \text{rel}'_{i,k+1} = \mathcal{K}^{\min}_{k+1} \tag{30}$$

Next, we need to handle the relation between the service times $\text{srv}_{i,k+1}$ and $\text{srv}'_{i,k+1}$ of this last request and we must explore three more sub-cases. These three sub-cases simply come from an enumeration of all possible "dominance" relations between the three terms $\mathcal{K}^{\max}_{k+1}$, $\text{rel}_{i,k+1} + \mathcal{T}^{\max}_i(1)$, and $\text{rel}'_{i,k+1} + \mathcal{T}^{\max}_i(1)$:

- Case 2.1 $\text{rel}'_{i,k+1} + \mathcal{T}^{\max}_i(1) \leq \text{rel}_{i,k+1} + \mathcal{T}^{\max}_i(1) \leq \mathcal{K}^{\max}_{k+1}$
- Case 2.2 $\text{rel}'_{i,k+1} + \mathcal{T}^{\max}_i(1) \leq \mathcal{K}^{\max}_{k+1} \leq \text{rel}_{i,k+1} + \mathcal{T}^{\max}_i(1)$
- Case 2.3 $\mathcal{K}^{\max}_{k+1} \leq \text{rel}'_{i,k+1} + \mathcal{T}^{\max}_i(1) \leq \text{rel}_{i,k+1} + \mathcal{T}^{\max}_i(1)$

Case 2.1 $\text{rel}'_{i,k+1} + \mathcal{T}^{\max}_i(1) \leq \text{rel}_{i,k+1} + \mathcal{T}^{\max}_i(1) \leq \mathcal{K}^{\max}_{k+1}$

In this particular sub-case, it holds from (26) and (28) that

$$\text{srv}_{i,k+1} = \text{rel}_{i,k+1} + \mathcal{T}_i^{\max}(1) \tag{31}$$
$$\text{srv}'_{i,k+1} = \text{rel}'_{i,k+1} + \mathcal{T}_i^{\max}(1) \tag{32}$$

and it immediately follows from (30), (31) and (32) that $\text{srv}_{i,k+1} \geq \text{srv}'_{i,k+1}$, which satisfies Condition (14) since $\sigma_i(k+1) = \sigma'_i(k+1) = h$. Also from (31) and (32), it holds that $\text{srv}_{i,k+1} - \text{rel}_{i,k+1} = \mathcal{T}_i^{\max}(1) = \text{srv}'_{i,k+1} - \text{rel}'_{i,k+1}$ and by using this equality together with $\mathcal{D}_i(k) \leq \mathcal{D}'_i(k)$ from (10), we obtain

$$\mathcal{D}_i(k) + \mathcal{T}_i^{\max}(1) \leq \mathcal{D}'_i(k) + \mathcal{T}_i^{\max}(1)$$

and thus

$$\mathcal{D}_i(k) + \text{srv}_{i,k+1} - \text{rel}_{i,k+1} \leq \mathcal{D}'_i(k) + \text{srv}'_{i,k+1} - \text{rel}'_{i,k+1}$$

By re-writing this inequality we get

$$\mathcal{D}_i(k+1) \leq \mathcal{D}'_i(k+1)$$

which satisfies Condition (13).

Case 2.2 $\text{rel}'_{i,k+1} + \mathcal{T}_i^{\max}(1) \leq \mathcal{K}_{k+1}^{\max} \leq \text{rel}_{i,k+1} + \mathcal{T}_i^{\max}(1)$

In this case, we have

$$\text{srv}_{i,k+1} = \mathcal{K}_{k+1}^{\max} \qquad \text{from (26)}$$
$$\text{and } \text{srv}'_{i,k+1} = \text{rel}'_{i,k+1} + \mathcal{T}_i^{\max}(1) \qquad \text{from (28)}$$

It thus holds from Case 2.2 that $\text{srv}_{i,k+1} \geq \text{srv}'_{i,k+1}$ and these service times trivially satisfy Condition (14) since $\sigma_i(k+1) = \sigma'_i(k+1) = h$. Then, assuming *by contradiction* that Condition (13) is *not* satisfied, we must have:

$$\mathcal{D}_i(k+1) > \mathcal{D}'_i(k+1)$$

which can be re-written as

$$\mathcal{D}_i(k) + \text{srv}_{i,k+1} - \text{rel}_{i,k+1} > \mathcal{D}'_i(k) + \text{srv}'_{i,k+1} - \text{rel}'_{i,k+1}$$

By replacing $\text{srv}_{i,k+1}$ and $\text{srv}'_{i,k+1}$ with their values, we get

$$\mathcal{D}_i(k) + \mathcal{K}_{k+1}^{\max} - \text{rel}_{i,k+1} > \mathcal{D}'_i(k) + \mathcal{T}_i^{\max}(1)$$

and then,

$$\mathcal{D}_i(k) > \mathcal{D}'_i(k) + \mathcal{T}_i^{\max}(1) - (\mathcal{K}_{k+1}^{\max} - \text{rel}_{i,k+1})$$

and since from Case 2.2 $\mathcal{K}_{k+1}^{\max} - \text{rel}_{i,k+1} \leq \mathcal{T}_i^{\max}(1)$, it follows from the above inequality that

$$\mathcal{D}_i(k) > \mathcal{D}'_i(k)$$

which contradicts Condition (10). This contradiction implies that Condition (13) *is* satisfied.

Case 2.3 $\mathcal{K}_{k+1}^{\max} \leq \text{rel}'_{i,k+1} + \mathcal{T}_i^{\max}(1) \leq \text{rel}_{i,k+1} + \mathcal{T}_i^{\max}(1)$

In this case it holds from (26) and (28) that

$$\text{srv}_{i,k+1} = \text{srv}'_{i,k+1} = \mathcal{K}_{k+1}^{\max}$$

and it immediately follows that $\text{srv}_{i,k+1} \geq \text{srv}'_{i,k+1}$, which satisfies Condition (14) since $\sigma_i(k+1) = \sigma'_i(k+1) = h$. Then, assuming *by contradiction* that Condition (13) is *not* satisfied, we must have:

$$\mathcal{D}_i(k+1) > \mathcal{D}'_i(k+1)$$

which can be re-written as

$$\mathcal{D}_i(k) + \text{srv}_{i,k+1} - \text{rel}_{i,k+1} > \mathcal{D}'_i(k) + \text{srv}'_{i,k+1} - \text{rel}'_{i,k+1}$$

By replacing $\text{srv}_{i,k+1}$ and $\text{srv}'_{i,k+1}$ with their values, we get

$$\mathcal{D}_i(k) + \mathcal{K}_{k+1}^{\max} - \text{rel}_{i,k+1} > \mathcal{D}'_i(k) + \mathcal{K}_{k+1}^{\max} - \text{rel}'_{i,k+1}$$

and then,

$$\mathcal{D}_i(k) > \mathcal{D}'_i(k) + \text{rel}_{i,k+1} - \text{rel}'_{i,k+1}$$

From Eq. (30), a case condition of Case 2,we have $\text{rel}_{i,k+1} \geq \text{rel}'_{i,k+1}$ and it follows from the above inequality that

$$\mathcal{D}_i(k) > \mathcal{D}'_i(k)$$

which contradicts Condition (10). This contradiction implies that Condition (13) *is* satisfied.

Case 3 $\mathcal{K}_{k+1}^{\min} \leq \text{srv}'_{i,k} + \Delta'_{k+1} \leq \text{srv}_{i,k} + \Delta_{k+1}$

In this case, from (25) and (27) we have the following relation between the release time-instants of the $(k+1)$'th request in the mappings $\mathbb{M}_i$ and $\mathbb{M}'_i$:

$$\text{rel}_{i,k+1} = \text{srv}_{i,k} + \Delta_{k+1}$$
$$\text{rel}'_{i,k+1} = \text{srv}'_{i,k} + \Delta'_{k+1}$$

Next, we need to handle the relation between the service times $\text{srv}_{i,k+1}$ and $\text{srv}'_{i,k+1}$ of this last request and we hence have the same three sub-cases to explore as in Case 2, but with the slightly different case conditions of Case 3.

- Case 3.1 $\text{rel}'_{i,k+1} + \mathcal{T}_i^{\max}(1) \leq \text{rel}_{i,k+1} + \mathcal{T}_i^{\max}(1) \leq \mathcal{K}_{k+1}^{\max}$
- Case 3.2 $\text{rel}'_{i,k+1} + \mathcal{T}_i^{\max}(1) \leq \mathcal{K}_{k+1}^{\max} \leq \text{rel}_{i,k+1} + \mathcal{T}_i^{\max}(1)$
- Case 3.3 $\mathcal{K}_{k+1}^{\max} \leq \text{rel}'_{i,k+1} + \mathcal{T}_i^{\max}(1) \leq \text{rel}_{i,k+1} + \mathcal{T}_i^{\max}(1)$

Case 3.1 $\text{rel}'_{i,k+1} + \mathcal{T}_i^{\max}(1) \leq \text{rel}_{i,k+1} + \mathcal{T}_i^{\max}(1) \leq \mathcal{K}_{k+1}^{\max}$
From (26) and (28), we get

$$\text{srv}_{i,k+1} = \text{rel}_{i,k+1} + \mathcal{T}_i^{\max}(1) \tag{33}$$

$$\text{and } \text{srv}'_{i,k+1} = \text{rel}'_{i,k+1} + \mathcal{T}_i^{\max}(1) \tag{34}$$

and it immediately follows from (33) and (34) that $\text{srv}_{i,k+1} \geq \text{srv}'_{i,k+1}$, which satisfies Condition (14) since $\sigma_i(k+1) = \sigma'_i(k+1) = h$. Also from (33) and (34), it holds that $\text{srv}_{i,k+1} - \text{rel}_{i,k+1} = \mathcal{T}_i^{\max}(1) = \text{srv}'_{i,k+1} - \text{rel}'_{i,k+1}$. Similarly to Case 2.1, by using this equality together with $\mathcal{D}_i(k) \leq \mathcal{D}'_i(k)$ from (10), we obtain

$$\mathcal{D}_i(k) + \mathcal{T}_i^{\max}(1) \leq \mathcal{D}'_i(k) + \mathcal{T}_i^{\max}(1)$$

and thus

$$\mathcal{D}_i(k) + \text{srv}_{i,k+1} - \text{rel}_{i,k+1} \leq \mathcal{D}'_i(k) + \text{srv}'_{i,k+1} - \text{rel}'_{i,k+1}$$

By re-writing this inequality we get

$$\mathcal{D}_i(k+1) \leq \mathcal{D}'_i(k+1)$$

which satisfies Condition (13).

Case 3.2 $\text{rel}'_{i,k+1} + \mathcal{T}_i^{\max}(1) \leq \mathcal{K}_{k+1}^{\max} \leq \text{rel}_{i,k+1} + \mathcal{T}_i^{\max}(1)$
In this case, we have

$$\text{srv}_{i,k+1} = \mathcal{K}_{k+1}^{\max} \text{ from (26)}$$
$$\text{srv}'_{i,k+1} = \text{rel}'_{i,k+1} + \mathcal{T}_i^{\max}(1) \text{ from (28)}$$

It thus holds from Case 3.2 that $\text{srv}_{i,k+1} \geq \text{srv}'_{i,k+1}$ and these service times trivially satisfy Condition (14) since $\sigma_i(k+1) = \sigma'_i(k+1) = h$. Then, assuming *by contradiction* that Condition (13) is *not* satisfied, we must have:

$$\mathcal{D}_i(k+1) > \mathcal{D}'_i(k+1)$$

which can be re-written as

$$\mathcal{D}_i(k) + \text{srv}_{i,k+1} - \text{rel}_{i,k+1} > \mathcal{D}'_i(k) + \text{srv}'_{i,k+1} - \text{rel}'_{i,k+1}$$

By replacing $\text{srv}_{i,k+1}$ and $\text{srv}'_{i,k+1}$ with their values, we get

$$\mathcal{D}_i(k) + \mathcal{K}_{k+1}^{\max} - \text{rel}_{i,k+1} > \mathcal{D}'_i(k) + \mathcal{T}_i^{\max}(1)$$

and then,

$$\mathcal{D}_i(k) > \mathcal{D}'_i(k) + \mathcal{T}_i^{\max}(1) - (\mathcal{K}_{k+1}^{\max} - \text{rel}_{i,k+1})$$

and since from Case 3.2 $\mathcal{K}_{k+1}^{\max} - \text{rel}_{i,k+1} \leq \mathcal{T}_i^{\max}(1)$, it follows from the above inequality that

$$\mathcal{D}_i(k) > \mathcal{D}'_i(k)$$

which contradicts Condition (10). This contradiction implies that Condition (13) *is* satisfied.

Case 3.3 $\mathcal{K}_{k+1}^{\max} \leq \text{rel}'_{i,k+1} + \mathcal{T}_i^{\max}(1) \leq \text{rel}_{i,k+1} + \mathcal{T}_i^{\max}(1)$

In this case, it holds from (26) and (28) that

$$\text{srv}_{i,k+1} = \text{srv}'_{i,k+1} = \mathcal{K}_{k+1}^{\max}$$

and it immediately follows that $\text{srv}_{i,k+1} \geq \text{srv}'_{i,k+1}$, which satisfies Condition (14) since $\sigma_i(k+1) = \sigma'_i(k+1) = h$. Then, assuming *by contradiction* that Condition (13) is *not* satisfied, we must have:

$$\mathcal{D}_i(k+1) > \mathcal{D}'_i(k+1)$$

which can be re-written as

$$\mathcal{D}_i(k) + \text{srv}_{i,k+1} - \text{rel}_{i,k+1} > \mathcal{D}'_i(k) + \text{srv}'_{i,k+1} - \text{rel}'_{i,k+1}$$

By replacing $\text{srv}_{i,k+1}$ and $\text{srv}'_{i,k+1}$ with their values, we get

$$\mathcal{D}_i(k) + \mathcal{K}_{k+1}^{\max} - \text{rel}_{i,k+1} > \mathcal{D}'_i(k) + \mathcal{K}_{k+1}^{\max} - \text{rel}'_{i,k+1}$$

and then,

$$\mathcal{D}_i(k) > \mathcal{D}'_i(k) + \text{rel}_{i,k+1} - \text{rel}'_{i,k+1}$$

From Case 3.3, we have $\text{rel}_{i,k+1} \geq \text{rel}'_{i,k+1}$ and it follows from the above inequality that

$$\mathcal{D}_i(k) > \mathcal{D}'_i(k)$$

which contradicts Condition (10). This contradiction implies that Condition (13) *is* satisfied. □

10.2 Proof of Lemma 5

Proof The proof must show that given Conditions (15), (16), and (17), Eqs. (18), (19), and (20) hold. From the claim itself, Eq. (18) trivially holds since $\sigma_i(k+1) = \sigma'_i(k+1) = h$. We stated this equality only for completeness in order to show that the situation after assigning the $(k+1)$'th request is same as the situation before assigning it. Let us start the proof by introducing some symbols to improve readability:

$$\mathcal{K}^{\min}_{k+1} \stackrel{\text{def}}{=} \mathcal{T}^{\min}_i(h-1)+1 \text{ and } \mathcal{K}^{\max}_{k+1} \stackrel{\text{def}}{=} \mathcal{T}^{\max}_i(h)$$
$$\text{and } \Delta_{k+1} \stackrel{\text{def}}{=} (h-\sigma_i(k)) \times \text{TR} \text{ and } \Delta'_{k+1} \stackrel{\text{def}}{=} (h-\sigma'_i(k)) \times \text{TR}$$

According to these new symbols and from the equations of Lemma 3, the four quantities $\text{srv}_{i,k+1}$, $\text{rel}_{i,k+1}$, $\text{srv}'_{i,k+1}$, and $\text{rel}'_{i,k+1}$ can be re-written as

$$\text{rel}_{i,k+1} = \max\left(\mathcal{K}^{\min}_{k+1}, \text{srv}_{i,k} + \Delta_{k+1}\right) \tag{35}$$
$$\text{srv}_{i,k+1} = \min\left(\mathcal{K}^{\max}_{k+1}, \text{rel}_{i,k+1} + \mathcal{T}^{\max}_i(1)\right) \tag{36}$$
$$\text{rel}'_{i,k+1} = \max\left(\mathcal{K}^{\min}_{k+1}, \text{srv}'_{i,k} + \Delta'_{k+1}\right) \tag{37}$$
$$\text{srv}'_{i,k+1} = \min\left(\mathcal{K}^{\max}_{k+1}, \text{rel}'_{i,k+1} + \mathcal{T}^{\max}_i(1)\right) \tag{38}$$

According to (17), we have

$$\text{srv}_{i,k} - \sigma_i(k) \times \text{TR} \leq \text{srv}'_{i,k} - \sigma'_i(k) \times \text{TR}$$

and by adding "$h \times$ TR" to both sides we get

$$\text{srv}_{i,k} + (h-\sigma_i(k)) \times \text{TR} \leq \text{srv}'_{i,k} + (h-\sigma'_i(k)) \times \text{TR}$$

which gives, by definition of Δ_{k+1} and Δ'_{k+1},

$$\text{srv}_{i,k} + \Delta_{k+1} \leq \text{srv}'_{i,k} + \Delta'_{k+1} \tag{39}$$

With the help of Inequality (39), we will now prove that Inequalities (19) and (20) always hold true (remember that Inequality (18) is always satisfied). Note that both Inequalities (19) and (20) are indirectly based on the release and service time of the $(k+1)$'th request in both mappings $\mathbb{M}_i$ and $\mathbb{M}'_i$, i.e. they are based on the four quantities $\text{srv}_{i,k+1}$, $\text{rel}_{i,k+1}$, $\text{srv}'_{i,k+1}$, and $\text{rel}'_{i,k+1}$. Therefore, if we first focus on the relation between the release times $\text{rel}_{i,k+1}$ and $\text{rel}'_{i,k+1}$ in the two mappings $\mathbb{M}_i$ and $\mathbb{M}'_i$ then it holds from (39), (35), and (37) that only three cases must be investigated:

- Case 1 $\text{srv}_{i,k} + \Delta_{k+1} \leq \text{srv}'_{i,k} + \Delta'_{k+1} \leq \mathcal{K}^{\min}_{k+1}$
- Case 2 $\text{srv}_{i,k} + \Delta_{k+1} \leq \mathcal{K}^{\min}_{k+1} \leq \text{srv}'_{i,k} + \Delta'_{k+1}$
- Case 3 $\mathcal{K}^{\min}_{k+1} \leq \text{srv}_{i,k} + \Delta_{k+1} \leq \text{srv}'_{i,k} + \Delta'_{k+1}$

Case 1 $\text{srv}_{i,k} + \Delta_{k+1} \leq \text{srv}'_{i,k} + \Delta'_{k+1} \leq \mathcal{K}^{\min}_{k+1}$

Proof of (20) In this case, we have from (35) and (37), $\text{rel}_{i,k+1} = \text{rel}'_{i,k+1} = \mathcal{K}^{\min}_{k+1}$ and from (36) and (38), $\text{srv}_{i,k+1} = \text{srv}'_{i,k+1}$, which satisfies (20) since $\sigma_i(k+1) = \sigma'_i(k+1) = h$.

Proof of (19) By combining (15) with (17) we get $\text{srv}_{i,k} \leq \text{srv}'_{i,k}$ and thus it holds from Inequality (16) that $\mathcal{D}_i(k) \leq \mathcal{D}'_i(k)$. Therefore, since $\text{rel}_{i,k+1} = \text{rel}'_{i,k+1}$ and $\text{srv}_{i,k+1} = \text{srv}'_{i,k+1}$ it also holds that

$$\mathcal{D}_i(k) + \text{srv}_{i,k+1} - \text{rel}_{i,k+1} \leq \mathcal{D}'_i(k) + \text{srv}'_{i,k+1} - \text{rel}'_{i,k+1}$$

and thus,

$$\mathcal{D}_i(k+1) \leq \mathcal{D}'_i(k+1)$$

and since $\text{srv}_{i,k+1} = \text{srv}'_{i,k+1}$ in this case, we can write

$$\mathcal{D}_i(k+1) + (\text{srv}'_{i,k+1} - \text{srv}_{i,k+1}) \leq \mathcal{D}'_i(k+1)$$

which satisfies (19).

Case 2 $\text{srv}_{i,k} + \Delta_{k+1} \leq \mathcal{K}^{\min}_{k+1} \leq \text{srv}'_{i,k} + \Delta'_{k+1}$

In this case, we get from (35) and (37),

$$\text{rel}'_{i,k+1} = \text{srv}'_{i,k} + \Delta'_{k+1} \geq \text{rel}_{i,k+1} = \mathcal{K}^{\min}_{k+1} \tag{40}$$

Next, we need to handle the relation between the service times $\text{srv}_{i,k+1}$ and $\text{srv}'_{i,k+1}$ in the two mappings $\mathbb{M}_i$ and $\mathbb{M}'_i$ and it holds from (36) and (38) that we have three more sub-cases to explore:

- Case 2.1 $\text{rel}_{i,k+1} + \mathcal{T}^{\max}_i(1) \leq \text{rel}'_{i,k+1} + \mathcal{T}^{\max}_i(1) \leq \mathcal{K}^{\max}_{k+1}$
- Case 2.2 $\text{rel}_{i,k+1} + \mathcal{T}^{\max}_i(1) \leq \mathcal{K}^{\max}_{k+1} \leq \text{rel}'_{i,k+1} + \mathcal{T}^{\max}_i(1)$
- Case 2.3 $\mathcal{K}^{\max}_{k+1} \leq \text{rel}_{i,k+1} + \mathcal{T}^{\max}_i(1) \leq \text{rel}'_{i,k+1} + \mathcal{T}^{\max}_i(1)$

Case 2.1 $\text{rel}_{i,k+1} + \mathcal{T}^{\max}_i(1) \leq \text{rel}'_{i,k+1} + \mathcal{T}^{\max}_i(1) \leq \mathcal{K}^{\max}_{k+1}$

Proof of (20) From (36) and (38) we get

$$\text{srv}_{i,k+1} = \text{rel}_{i,k+1} + \mathcal{T}^{\max}_i(1) \tag{41}$$

$$\text{srv}'_{i,k+1} = \text{rel}'_{i,k+1} + \mathcal{T}^{\max}_i(1) \tag{42}$$

From (40), (41) and (42), it immediately follows that $\text{srv}_{i,k+1} \leq \text{srv}'_{i,k+1}$, which satisfies (20) since $\sigma_i(k+1) = \sigma'_i(k+1) = h$.

Proof of (19) Also from (41) and (42), it holds that $\text{srv}_{i,k+1} - \text{rel}_{i,k+1} = \mathcal{T}^{\max}_i(1) = \text{srv}'_{i,k+1} - \text{rel}'_{i,k+1}$. Using $\mathcal{D}_i(k) + (\text{srv}'_{i,k} - \text{srv}_{i,k}) \leq \mathcal{D}'_i(k)$ from (16), we get

$$\mathcal{D}_i(k) + \left(\text{srv}'_{i,k} - \text{srv}_{i,k}\right) + \mathcal{T}^{\max}_i(1) \leq \mathcal{D}'_i(k) + \mathcal{T}^{\max}_i(1)$$

and thus

$$\mathcal{D}_i(k) + \left(\mathrm{srv}'_{i,k} - \mathrm{srv}_{i,k}\right) + \mathrm{srv}_{i,k+1} - \mathrm{rel}_{i,k+1} \leq \mathcal{D}'_i(k) + \mathrm{srv}'_{i,k+1} - \mathrm{rel}'_{i,k+1}$$

which implies

$$\mathcal{D}_i(k+1) + \left(\mathrm{srv}'_{i,k} - \mathrm{srv}_{i,k}\right) \leq \mathcal{D}'_i(k+1) \tag{43}$$

Now, we have:

$$\begin{aligned}
\mathrm{srv}'_{i,k+1} - \mathrm{srv}_{i,k+1} &= \mathrm{rel}'_{i,k+1} + \mathcal{T}_i^{\max}(1) - (\mathrm{rel}_{i,k+1} + \mathcal{T}_i^{\max}(1)) \\
&\quad \text{from (41) and (42)} \\
&= \mathrm{rel}'_{i,k+1} - \mathrm{rel}_{i,k+1} \\
&= \mathrm{srv}'_{i,k} + \Delta'_{k+1} - \mathcal{K}_{k+1}^{\min} \\
&\quad \text{from (40)} \\
&\leq \mathrm{srv}'_{i,k} + \Delta'_{k+1} - (\mathrm{srv}_{i,k} + \Delta_{k+1}) \\
&\quad \text{from Case 2} \\
&\leq \mathrm{srv}'_{i,k} - \mathrm{srv}_{i,k} + \left(\Delta'_{k+1} - \Delta_{k+1}\right) \\
&\leq \mathrm{srv}'_{i,k} - \mathrm{srv}_{i,k} + \left(h - \sigma'_i(k)\right) \times \mathrm{TR} - (h - \sigma_i(k)) \times \mathrm{TR} \\
&\quad \text{from the definition of } \Delta_{k+1} \text{ and } \Delta'_{k+1} \\
&\leq \mathrm{srv}'_{i,k} - \mathrm{srv}_{i,k} + \left(\sigma_i(k) - \sigma'_i(k)\right) \times \mathrm{TR} \\
&\leq \mathrm{srv}'_{i,k} - \mathrm{srv}_{i,k} \\
&\quad \text{since} \sigma_i(k) - \sigma'_i(k) \leq 0 \text{ from (15)}
\end{aligned}$$

Therefore it holds from the above inequality and from (43) that:

$$\mathcal{D}_i(k+1) + \left(\mathrm{srv}'_{i,k+1} - \mathrm{srv}_{i,k+1}\right) \leq \mathcal{D}'_i(k+1)$$

which satisfies (19).

Case 2.2 $\mathrm{rel}_{i,k+1} + \mathcal{T}_i^{\max}(1) \leq \mathcal{K}_{k+1}^{\max} \leq \mathrm{rel}'_{i,k+1} + \mathcal{T}_i^{\max}(1)$

Proof of (20) From (36) and (38), we get $\mathrm{srv}_{i,k+1} = \mathrm{rel}_{i,k+1} + \mathcal{T}_i^{\max}(1)$ and $\mathrm{srv}'_{i,k+1} = \mathcal{K}_{k+1}^{\max}$. We thus get $\mathrm{srv}_{i,k+1} \leq \mathrm{srv}'_{i,k+1}$, which satisfies (20) since $\sigma_i(k+1) = \sigma'_i(k+1) = h$.

Proof of (19) We use proof by contradiction. Suppose that Inequality (19) is *not* satisfied, we must have:

$$\mathcal{D}_i(k+1) + \left(\mathrm{srv}'_{i,k+1} - \mathrm{srv}_{i,k+1}\right) > \mathcal{D}'_i(k+1)$$

and thus,

$$\mathcal{D}_i(k) + \left(\mathrm{srv}'_{i,k+1} - \mathrm{srv}_{i,k+1}\right) + \mathrm{srv}_{i,k+1} - \mathrm{rel}_{i,k+1} > \mathcal{D}'_i(k) + \mathrm{srv}'_{i,k+1} - \mathrm{rel}'_{i,k+1}$$

which can be re-written as

$$\mathcal{D}_i(k) - \mathrm{rel}_{i,k+1} > \mathcal{D}'_i(k) - \mathrm{rel}'_{i,k+1}$$

and since it holds from (40) that $\mathrm{rel}'_{i,k+1} = \mathrm{srv}'_{i,k+1} + \Delta'_{k+1} \geq \mathrm{rel}_{i,k+1} = \mathcal{K}^{\min}_{k+1}$ and $\mathcal{K}^{\min}_{k+1} \geq \mathrm{srv}_{i,k} + \Delta_{k+1}$ from Case 2, we get

$$\mathcal{D}_i(k) - (\mathrm{srv}_{i,k} + \Delta_{k+1}) > \mathcal{D}'_i(k) - \left(\mathrm{srv}'_{i,k+1} + \Delta'_{k+1}\right)$$

and thus

$$\mathcal{D}_i(k) + \left(\mathrm{srv}'_{i,k+1} - \mathrm{srv}_{i,k}\right) + \left(\Delta'_{k+1} - \Delta_{k+1}\right) > \mathcal{D}'_i(k)$$

As seen at the end of Case 2.1, we have $\Delta'_{k+1} - \Delta_{k+1} \leq 0$ from (15) and from the definitions of Δ_{k+1} and Δ'_{k+1}, and thus it holds that

$$\mathcal{D}_i(k) + \left(\mathrm{srv}'_{i,k+1} - \mathrm{srv}_{i,k}\right) > \mathcal{D}'_i(k)$$

which contradicts (16). This contradiction implies that Condition (19) is always satisfied.

Case 2.3 $\mathcal{K}^{\max}_{k+1} \leq \mathrm{rel}_{i,k+1} + \mathcal{T}^{\max}_i(1) \leq \mathrm{rel}'_{i,k+1} + \mathcal{T}^{\max}_i(1)$

Proof of (20) From (36) and (38), we get $\mathrm{srv}_{i,k+1} = \mathrm{srv}'_{i,k+1} = \mathcal{K}^{\max}_{k+1}$ and it immediately follows that $\mathrm{srv}_{i,k+1} \leq \mathrm{srv}'_{i,k+1}$, which satisfies (20) since $\sigma_i(k+1) = \sigma'_i(k+1) = h$.

Proof of (19) The proof is identical to the proof of (19) in Case 2.2 and is repeated here only for completeness. We use proof by contradiction. Suppose that Inequality (19) is *not* satisfied, we must have:

$$\mathcal{D}_i(k+1) + \left(\mathrm{srv}'_{i,k+1} - \mathrm{srv}_{i,k+1}\right) > \mathcal{D}'_i(k+1)$$

and thus,

$$\mathcal{D}_i(k) + \left(\mathrm{srv}'_{i,k+1} - \mathrm{srv}_{i,k+1}\right) + \mathrm{srv}_{i,k+1} - \mathrm{rel}_{i,k+1} > \mathcal{D}'_i(k) + \mathrm{srv}'_{i,k+1} - \mathrm{rel}'_{i,k+1}$$

which can be re-written as

$$\mathcal{D}_i(k) - \mathrm{rel}_{i,k+1} > \mathcal{D}'_i(k) - \mathrm{rel}'_{i,k+1}$$

and since it holds from (40) that $\mathrm{rel}'_{i,k+1} = \mathrm{srv}'_{i,k+1} + \Delta'_{k+1} \geq \mathrm{rel}_{i,k+1} = \mathcal{K}^{\min}_{k+1}$ and $\mathcal{K}^{\min}_{k+1} \geq \mathrm{srv}_{i,k} + \Delta_{k+1}$ from the Case 2, we get

$$\mathcal{D}_i(k) - (\mathrm{srv}_{i,k} + \Delta_{k+1}) > \mathcal{D}'_i(k) - \left(\mathrm{srv}'_{i,k+1} + \Delta'_{k+1}\right)$$

and thus

$$\mathcal{D}_i(k) + \left(\text{srv}'_{i,k+1} - \text{srv}_{i,k}\right) + \left(\Delta'_{k+1} - \Delta_{k+1}\right) > \mathcal{D}'_i(k)$$

As seen at the end of Case 2.1, we have $\Delta'_{k+1} - \Delta_{k+1} \leq 0$ from (15) and from the definitions of Δ_{k+1} and Δ'_{k+1}, and thus it holds that

$$\mathcal{D}_i(k) + \left(\text{srv}'_{i,k+1} - \text{srv}_{i,k}\right) > \mathcal{D}'_i(k)$$

which contradicts (16). This contradiction implies that Condition (19) is always satisfied.

Case 3 $\mathcal{K}^{\min}_{k+1} \leq \text{srv}_{i,k} + \Delta_{k+1} \leq \text{srv}'_{i,k} + \Delta'_{k+1}$

In this case, we get from (35) and (37), $\text{rel}_{i,k+1} = \text{srv}_{i,k} + \Delta_{k+1}$ and $\text{rel}'_{i,k+1} = \text{srv}'_{i,k} + \Delta'_{k+1}$ and thus, according to (39), it holds that

$$\text{rel}_{i,k+1} \leq \text{rel}'_{i,k+1} \tag{44}$$

Again, we need to handle the relation between the service times $\text{srv}_{i,k+1}$ and $\text{srv}'_{i,k+1}$ in the two mappings $\mathbb{M}_i$ and $\mathbb{M}'_i$ and it holds from (36) and (38) that we have three more sub-cases to explore:

- Case 3.1 $\text{rel}_{i,k+1} + \mathcal{T}^{\max}_i(1) \leq \text{rel}'_{i,k+1} + \mathcal{T}^{\max}_i(1) \leq \mathcal{K}^{\max}_{k+1}$
- Case 3.2 $\text{rel}_{i,k+1} + \mathcal{T}^{\max}_i(1) \leq \mathcal{K}^{\max}_{k+1} \leq \text{rel}'_{i,k+1} + \mathcal{T}^{\max}_i(1)$
- Case 3.3 $\mathcal{K}^{\max}_{k+1} \leq \text{rel}_{i,k+1} + \mathcal{T}^{\max}_i(1) \leq \text{rel}'_{i,k+1} + \mathcal{T}^{\max}_i(1)$

Case 3.1 $\text{rel}_{i,k+1} + \mathcal{T}^{\max}_i(1) \leq \text{rel}'_{i,k+1} + \mathcal{T}^{\max}_i(1) \leq \mathcal{K}^{\max}_{k+1}$

Proof of (20) From (36) and (38), we get

$$\text{srv}_{i,k+1} = \text{rel}_{i,k+1} + \mathcal{T}^{\max}_i(1) \tag{45}$$
$$\text{srv}'_{i,k+1} = \text{rel}'_{i,k+1} + \mathcal{T}^{\max}_i(1) \tag{46}$$

From (44), (45) and (46), it immediately follows that $\text{srv}_{i,k+1} \leq \text{srv}'_{i,k+1}$, which satisfies (20) since $\sigma_i(k+1) = \sigma'_i(k+1) = h$.

Proof of (19) From (45) and (46), it holds that $\text{srv}_{i,k+1} - \text{rel}_{i,k+1} = \mathcal{T}^{\max}_i(1) = \text{srv}'_{i,k+1} - \text{rel}'_{i,k+1}$. Using $\mathcal{D}_i(k) + (\text{srv}'_{i,k} - \text{srv}_{i,k}) \leq \mathcal{D}'_i(k)$ from (16), we get

$$\mathcal{D}_i(k) + \left(\text{srv}'_{i,k} - \text{srv}_{i,k}\right) + \mathcal{T}^{\max}_i(1) \leq \mathcal{D}'_i(k) + \mathcal{T}^{\max}_i(1)$$

and then

$$\mathcal{D}_i(k) + \left(\text{srv}'_{i,k} - \text{srv}_{i,k}\right) + \text{srv}_{i,k+1} - \text{rel}_{i,k+1} \leq \mathcal{D}'_i(k) + \text{srv}'_{i,k+1} - \text{rel}'_{i,k+1}$$

which implies

$$\mathcal{D}_i(k+1) + \left(\text{srv}'_{i,k} - \text{srv}_{i,k}\right) \leq \mathcal{D}'_i(k+1) \tag{47}$$

Now, we have:

$$\begin{aligned}
\mathrm{srv}'_{i,k+1} - \mathrm{srv}_{i,k+1} &= \mathrm{rel}'_{i,k+1} + \mathcal{T}_i^{\max}(1) - (\mathrm{rel}_{i,k+1} + \mathcal{T}_i^{\max}(1)) \\
&\quad \text{from (45) and (46)} \\
&= \mathrm{rel}'_{i,k+1} - \mathrm{rel}_{i,k+1} \\
&\quad \text{From (35) and (37) and Case 3} \\
&= \mathrm{srv}'_{i,k} + \Delta'_{k+1} - (\mathrm{srv}_{i,k} + \Delta_{k+1}) \\
&= \mathrm{srv}'_{i,k} - \mathrm{srv}_{i,k} + (\Delta'_{k+1} - \Delta_{k+1}) \\
&\leq \mathrm{srv}'_{i,k} - \mathrm{srv}_{i,k} \\
&\quad \text{because } \Delta'_{k+1} - \Delta_{k+1} \leq 0 \text{ from (15)} \\
&\quad \text{and the definitions of } \Delta_{k+1} \text{ and } \Delta'_{k+1}
\end{aligned}$$

Therefore, it holds from the above inequality and from (47) that:

$$\mathcal{D}_i(k+1) + \left(\mathrm{srv}'_{i,k+1} - \mathrm{srv}_{i,k+1}\right) \leq \mathcal{D}'_i(k+1)$$

which satisfies (19).

Case 3.2 $\mathrm{rel}_{i,k+1} + \mathcal{T}_i^{\max}(1) \leq \mathcal{K}_{k+1}^{\max} \leq \mathrm{rel}'_{i,k+1} + \mathcal{T}_i^{\max}(1)$

Proof of (20) From (36) and (38), we get $\mathrm{srv}_{i,k+1} = \mathrm{rel}_{i,k+1} + \mathcal{T}_i^{\max}(1)$ and $\mathrm{srv}'_{i,k+1} = \mathcal{K}_{k+1}^{\max}$. We thus get $\mathrm{srv}_{i,k+1} \leq \mathrm{srv}'_{i,k+1}$, which satisfies (20) since $\sigma_i(k+1) = \sigma'_i(k+1) = h$.

Proof of (19) The proof is by contradiction. If Inequality (19) is *not* satisfied then we must have:

$$\mathcal{D}_i(k+1) + \left(\mathrm{srv}'_{i,k+1} - \mathrm{srv}_{i,k+1}\right) > \mathcal{D}'_i(k+1)$$

and thus,

$$\left(\mathcal{D}_i(k) + \mathrm{srv}_{i,k+1} - \mathrm{rel}_{i,k+1}\right) + \left(\mathrm{srv}'_{i,k+1} - \mathrm{srv}_{i,k+1}\right) > \mathcal{D}'_i(k) + \mathrm{srv}'_{i,k+1} - \mathrm{rel}'_{i,k+1}$$

which can be re-written as

$$\mathcal{D}_i(k) - \mathrm{rel}_{i,k+1} > \mathcal{D}'_i(k) - \mathrm{rel}'_{i,k+1}$$

By replacing the values of $\mathrm{rel}_{i,k+1}$ and $\mathrm{rel}'_{i,k+1}$ from Case 3, we have

$$\mathcal{D}_i(k) - (\mathrm{srv}_{i,k} + \Delta_{k+1}) > \mathcal{D}'_i(k) - \left(\mathrm{srv}'_{i,k} + \Delta'_{k+1}\right)$$

and thus,

$$\mathcal{D}_i(k) + \left(\mathrm{srv}'_{i,k} - \mathrm{srv}_{i,k}\right) + \left(\Delta'_{k+1} - \Delta_{k+1}\right) > \mathcal{D}'_i(k)$$

and since $\Delta'_{k+1} - \Delta_{k+1} \leq 0$ from (15) and the def. of Δ_{k+1} and Δ'_{k+1}, it holds that

$$\mathcal{D}_i(k) + \mathrm{srv}'_{i,k} - \mathrm{srv}_{i,k} > \mathcal{D}'_i(k)$$

which contradicts (16). This contradiction implies that Eq. (19) is satisfied.

Case 3.3 $\mathcal{K}_{k+1}^{\max} \leq \text{rel}_{i,k+1} + \mathcal{T}_i^{\max}(1) \leq \text{rel}'_{i,k+1} + \mathcal{T}_i^{\max}(1)$

Proof of (20) From (36) and (38), we get $\text{srv}_{i,k+1} = \text{srv}'_{i,k+1} = \mathcal{K}_{k+1}^{\max}$ and it immediately follows that $\text{srv}_{i,k+1} \leq \text{srv}'_{i,k+1}$, which satisfies (20) since $\sigma_i(k+1) = \sigma'_i(k+1) = h$.

Proof of (19) The proof is by contradiction. If Inequality (19) is *not* satisfied, we must have:

$$\mathcal{D}_i(k+1) + \left(\text{srv}'_{i,k+1} - \text{srv}_{i,k+1}\right) > \mathcal{D}'_i(k+1)$$

and thus,

$$\mathcal{D}_i(k) + \left(\text{srv}'_{i,k+1} - \text{srv}_{i,k+1}\right) + \text{srv}_{i,k+1} - \text{rel}_{i,k+1} > \mathcal{D}'_i(k) + \text{srv}'_{i,k+1} - \text{rel}'_{i,k+1}$$

By replacing $\text{srv}_{i,k+1}$ and $\text{srv}'_{i,k+1}$ with their values, we get

$$\mathcal{D}_i(k) + \left(\mathcal{K}_{k+1}^{\max} - \mathcal{K}_{k+1}^{\max}\right) + \mathcal{K}_{k+1}^{\max} - \text{rel}_{i,k+1} > \mathcal{D}'_i(k) + \mathcal{K}_{k+1}^{\max} - \text{rel}'_{i,k+1}$$

and thus,

$$\mathcal{D}_i(k) + \text{rel}'_{i,k+1} - \text{rel}_{i,k+1} > \mathcal{D}'_i(k)$$

By replacing the values of $\text{rel}_{i,k+1}$ and $\text{rel}'_{i,k+1}$ from Case 3, we have

$$\mathcal{D}_i(k) + \left(\text{srv}'_{i,k} + \Delta'_{k+1}\right) - \left(\text{srv}_{i,k} + \Delta_{k+1}\right) > \mathcal{D}'_i(k)$$

which gives

$$\mathcal{D}_i(k) + \left(\text{srv}'_{i,k} - \text{srv}_{i,k}\right) + \left(\Delta'_{k+1} - \Delta_{k+1}\right) > \mathcal{D}'_i(k)$$

and since $\Delta'_{k+1} - \Delta_{k+1} \leq 0$ from (15) and the def. of Δ_{k+1} and Δ'_{k+1}, it holds that

$$\mathcal{D}_i(k) + \text{srv}'_{i,k} - \text{srv}_{i,k} > \mathcal{D}'_i(k)$$

which contradicts (16). This contradiction implies that Eq. (19) is satisfied. □

References

Akesson B, Goossens K (2011) Architectures and modeling of predictable memory controllers for improved system integration. In: Design, automation test in Europe conference exhibition (DATE), 2011, pp 1–6

Akesson B, Hansson A, Goossens K (2009) Composable resource sharing based on latency-rate servers. In: 12th Euromicro conference on digital system design, architectures, methods and tools, 2009, DSD'09. IEEE Computer Society, Washington, DC, pp 547–555

Andersson B, Easwaran A, Lee J (2010) Finding an upper bound on the increase in execution time due to contention on the memory bus in COTS-based multicore systems. ACM Sigbed Rev 7(1):4

Austin T, Larson E, Ernst D (2002) SimpleScalar: an infrastructure for computer system modeling. Computer 35(2):59–67

Behnam M, Inam R, Nolte T, Sjödin M (2013) Multi-core composability in the face of memory-bus contention. SIGBED Rev 10(3):35–42

Benini, L., Flamand, E., Fuin, D., Melpignano, D.: P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator. In: Proc. of Design, Automation and Test in Europe Conference, pp. 983–987 (2012)

Chattopadhyay S, Chong LK, Roychoudhury A, Kelter T, Marwedel P, Falk H (2014) A unified WCET analysis framework for multicore platforms. ACM Trans Embed Comput Syst 13(4s):124:1–124:29

Chattopadhyay S, Roychoudhury A, Mitra T (2010) Modeling shared cache and bus in multi-cores for timing analysis. In: Proceedings of the 13th international workshop on software & compilers for embedded systems, pp 6:1–6:10

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

Dasari D, Akesson B, Nelis V, Awan MAA, Petters SM (2013) Identifying the sources of unpredictability in COTS-based multicore systems. In: 8th IEEE international symposium on industrial embedded systems (SIES)

Dasari D, Andersson B, Nelis V, Petters SM, Easwaran A, Lee J (2011) Response time analysis of COTS-based multicores considering the contention on the shared memory bus. In: IEEE 10th international conference on trust, security and privacy in computing and communications, pp 1068–1075

Dasari D, Nelis V (2012) An analysis of the impact of bus contention on the WCET in multicores. In: IEEE 9th international conference on embedded software and systems (HPCC-ICESS), pp 1450–1457

Gustafsson J, Betts A, Ermedahl A, Lisper B (2010) The Mälardalen WCET benchmarks—past, present and future. OCG, Brussels, pp 137–147

Hara Y, Tomiyama H, Honda S, Takada H, Ishii K (2008) CHStone: a benchmark program suite for practical c-based high-level synthesis. In: IEEE international symposium on circuits and systems, 2008, ISCAS 2008, pp 1192–1195

IEC 61508 (2010) Functional safety of electrical/electronic/programmable electronic safety-related systems

JEDEC Solid State Technology Association (2012) DDR3 SDRAM specification, JESD79-3F edn

Kelter T, Falk H, Marwedel P, Chattopadhyay S, Roychoudhury A (2011) Bus-aware multicore WCET analysis through TDMA offset bounds. In: Proceedings of the 2011 Euromicro conference on real-time systems, pp 3–12

Kollig P, Osborne C, Henriksson T (2009) Heterogeneous multi-core platform for consumer multimedia applications. In: Proceedings of design, automation and test in Europe conference

Lee C, Potkonjak M, Mangione-Smith W (1997) Mediabench: a tool for evaluating and synthesizing multimedia and communicatons systems. In: Proceedings of ACM/IEEE international symposium on microarchitecture, pp 330–335

Li Y, Akesson B, Goossens K (2014) Dynamic command scheduling for real-time memory controllers. In: 26th Euromicro conference on real-time systems (ECRTS), pp 3–14

Nowotsch, J., Paulitsch, M.: Leveraging multi-core computing architectures in avionics. In: 2012 Ninth European dependable computing conference (EDCC), pp 132–143. IEEE, Washington, DC (2012)

Nowotsch J, Paulitsch M, Henrichsen A, Pongratz W, Schacht A (2014) Monitoring and wcet analysis in cots multi-core-soc-based mixed-criticality systems. In: Design, automation and test in Europe conference and exhibition (DATE), pp 1–5

Paolieri M, Quinones E, Cazorla F, Valero M (2009) An analyzable memory controller for hard real-time CMPs. IEEE Embe Syst Lett 1(4):86–90

Pellizzoni R, Schranzhofer A, Chen JJ, Caccamo M, Thiele L (2010) Worst case delay analysis for memory interference in multicore systems. In: Conference on design, automation and test in Europe, pp 741–746

Reineke J, Liu I, Patel H, Kim S, Lee EA (2011) PRET DRAM controller: bank privatization for predictability and temporal isolation. In: CODES+ISSS '11: proceedings of the IEEE/ACM international conference on Hardware/software codesign and system synthesis, pp 99–108

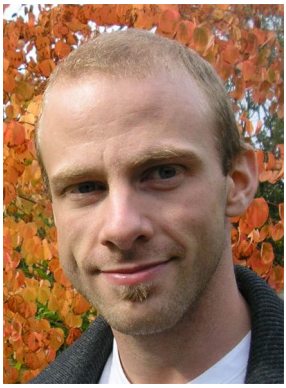
Rodrigues V, Akesson B, Melo de Sousa S, Florido M (2013) A declarative compositional timing analysis for multicores using the latency-rate abstraction. Practical aspects of declarative languages. Lecture Notes in Computer Science, vol 7752. Springer, Berlin Heidelberg, pp 43–59

Rosén J, Andrei A, Eles P, Peng Z (2007) Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip. In: Proceedings of the real-time systems symposium, pp 49–60

Schliecker S, Ernst R (2011) Real-time performance analysis of multiprocessor systems with shared memory. ACM Trans Embed Comput Syst 10:22:1–22:27
Schliecker S, Negrean M, Ernst R (2010) Bounding the shared resource load for the performance analysis of multiprocessor systems. In: Proceedings of the conference on design, automation and test in Europe, pp 759–764
Schranzhofer A, Chen JJ, Thiele L (2010) Timing analysis for TDMA arbitration in resource sharing systems. In: 16th IEEE real-time and embedded technology and applications symposium, pp 215–224
Schranzhofer A, Pellizzoni R, Chen JJ, Thiele L, Caccamo M (2010) Worst-case response time analysis of resource access models in multi-core systems. In: Proceedings of the 47th design automation conference. DAC'10. ACM, New York, pp 332–337
Schranzhofer A, Pellizzoni R, Chen JJ, Thiele L, Caccamo M (2011) Timing analysis for resource access interference on adaptive resource arbiters. In: Real-time and embedded technology and applications symposium
Shah H, Raabe A, Knoll A (2012) Bounding WCET of applications using SDRAM with priority based budget scheduling in MPSoCs. In: Design, automation test in Europe conference exhibition (DATE), pp 665–670
Thiele L, Chakraborty S, Naedele M (2000) Real-time calculus for scheduling hard real-time systems. In: The 2000 IEEE international symposium on circuits and systems, 2000. ISCAS 2000, Geneva, vol 4. IEEE Computer Society Press, Washington, DC, pp 101–104
van Berkel C (2009) Multi-core for mobile phones. In: Proceedings of design, automation and test in Europe conference, pp 1260–1265
Wenzel I, Kirner R, Rieder B, Puschner P (2009) Measurement-based timing analysis. Leveraging applications of formal methods verification and validation. Springer, Berlin
Wilhelm R, Engblom J, Ermedahl A, Holsti N, Thesing S, Whalley D, Bernat G, Ferdinand C, Heckmann R, Mitra T, Mueller F, Puaut I, Puschner P, Staschulat J, Stenström P (2008) The worst-case execution-time problem - overview of methods and survey of tools. ACM Trans Embed Comput Syst 7:36:1–36:53
Wilhelm R, Grund D, Reineke J, Schlickling M, Pister M, Ferdinand C (2009) Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems. Trans Comput-Aided Des Integ Circ Syst 28(7):966–978
Wu ZP, Krish Y, Pellizzoni R (2013) Worst case analysis of DRAM latency in multi-requestor systems. In: Proceedings of IEEE real-time systems symposium
Yun H, Yao G, Pellizzoni R, Caccamo M, Sha L (2012) Memory access control in multiprocessor for real-time systems with mixed criticality. In: 24th Euromicro conference on real-time systems (ECRTS), pp 299–308
Yun H, Yao G, Pellizzoni R, Caccamo M, Sha L (2013) Memguard: memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In: IEEE 19th real-time and embedded technology and applications symposium (RTAS). IEEE Computer Society Press, Washington, DC, pp 55–64
Zhou M, Bock S, Ferreira A, Childers B, Melhem R, Mosse D (2011) Real-time scheduling for phase change main memory systems. In: TrustCom, ICESS11, pp 991–998

Dakshina Dasari received her Masters Degree in 2004 from National Institute of Technology, Surathkal (NITK), India and Ph.D from the University of Porto in 2014. Prior to her Ph.D she has worked in the networking domain at Sun Microsystems, Citrix Systems. Post Ph.D, she has been working as an architect at the Robert Bosch Research and Technology Centre, India. Her research interests include computer architectures, real-time embedded systems with a focus on timing analysis and architectures for embedded automotive systems.

Vincent Nelis received his Ph.D. degree in Computer Science at the University of Brussels (ULB) in 2010. Since then, he has been working as a Research Associate at CISTER-ISEP Research Unit in Porto, Portugal. His research interests include real-time scheduling theory, with a focus on multiprocessor/multi-core systems, and in execution time and interference analysis.



Benny Akesson received his M.Sc. degree at Lund Institute of Technology, Sweden in 2005 and a Ph.D. from Eindhoven University of Technology, the Netherlands in 2010. Since then, he has been employed as a Postdoctoral Researcher at Eindhoven University of Technology, CISTER-ISEP Research Unit, and Czech Technical University in Prague. His research interests include design and analysis of multi-core real-time systems with shared resources.