An Efficient Configuration Methodology for Time-Division Multiplexed Single Resources

Benny Akesson[1], Anna Minaeva[1], Přemysl Šůcha[1], Andrew Nelson[2] and Zdeněk Hanzálek[1]
[1]Czech Technical University in Prague, [2]Eindhoven University of Technology

***Abstract*—Complex contemporary systems contain multiple applications, some which have firm real-time requirements while others do not. These applications are deployed on multi-core platforms with shared resources, such as processors, interconnect, and memories. However, resource sharing causes contention between sharing applications that must be resolved by a resource arbiter. Time-Division Multiplexing (TDM) is a commonly used arbiter, but it is challenging to configure such that the bandwidth and latency requirements of the real-time resource clients are satisfied, while minimizing their total allocation to improve the performance of non-real-time clients.**

This work addresses this problem by presenting an efficient TDM configuration methodology. The five main contributions are: 1) An analysis to derive a bandwidth and latency guarantee for a TDM schedule with arbitrary slot assignment, 2) A formulation of the TDM configuration problem and a proof that it is NP-hard, 3) An integer-linear programming model that optimally solves the configuration problem by exhaustively evaluating all possible TDM schedule sizes, 4) A heuristic method to choose candidate schedule sizes that substantially reduces computation time with only a slight decrease in efficiency, 5) An experimental evaluation of the methodology that examines its scalability and quantifies the trade-off between computation time and total allocation for the optimal and the heuristic algorithms. The approach is also demonstrated on a case study of a HD video and graphics processing system, where a memory controller is shared by a number of processing elements.

I. Introduction

Modern consumer-electronics systems feature an increasing number of applications. Some of these applications have *firm real-time requirements* and must always satisfy their deadlines, while other non-real-time applications are more concerned about average performance to feel responsive to the user. To realize the application requirements with low cost and energy consumption, platforms are implemented with heterogeneous processing cores and hardware accelerators sharing resources, such as memories, buses, and peripherals [1], [2]. The processing cores and accelerators are hence *clients* competing for shared resources, resulting in *contention* that must be resolved by a resource arbiter. Time-Division Multiplexing (TDM) is a commonly used arbiter for many types of resources ranging from processing elements via buses and networks-on-chips to memory controllers and peripherals. The reasons for its popularity is that it is conceptually easy to understand and analyze and has efficient implementations both in hardware and software. Additionally, it provides temporal isolation between clients when used in a non-work-conserving manner [3]. Example platforms relying extensively on TDM for a variety of resources are PRET [4] and CompSOC [3].

An important problem with TDM arbitration is to find a schedule that efficiently satisfies the *bandwidth and latency requirements* of the clients, while minimizing their resource utilization (maximizing slack capacity) to improve performance of their non-real-time counterparts. Such a configuration requires both choosing a suitable schedule length (frame size) and assigning time slots to the clients. This configuration process must be efficient in terms of minimizing utilization to reduce cost and must finish within reasonable time, even for complex systems, to avoid negatively impacting design time.

The five main contributions of this paper are: 1) We present a novel approach to derive a latency-rate ($\mathcal{LR}$) [5] service guarantee on bandwidth and latency for a single resource shared by a TDM arbiter with arbitrary slot assignments and prove its correctness. 2) We formulate our TDM configuration problem and prove that it is NP-hard, 3) We present an optimal integer-linear programming (ILP) model that assumes the TDM frame size to be given, requiring many iterations to synthesize the best frame size, 4) We propose a heuristic algorithm for choosing a limited set of frame sizes to evaluate with the ILP model, providing a trade-off between the computation time of the solver and efficiency that is useful when non-optimal solutions are acceptable. 5) We experimentally evaluate the scalability of the approach and quantify the trade-off between efficiency and computation time of the optimal and heuristic solutions. The methodology is also demonstrated on a case study of a HD video and graphics processing system, where 7 memory clients share a memory.

The rest of this paper is organized as follows. Related work is discussed in Section II. Section III then proceeds by presenting necessary background about the $\mathcal{LR}$ server framework and its application to TDM arbitration. The analysis that extends the $\mathcal{LR}$ framework to cover arbitrary slot assignments is presented in Section IV, before the configuration problem is formalized and its complexity determined in Section V. The ILP model is then introduced in Section VI, followed by the heuristic filtering algorithm in Section VII. The experimental evaluation and case study are presented in Section VIII, before concluding the paper in Section IX.

II. Related work

There are many examples of work relying on TDM arbitration in the context of timing analysis of platform resources, such as memory controllers and buses. These can be divided into two classes; those using *fine-grained* [6]–[9] and *coarse-grained* [10]–[15] TDM arbitration, respectively. A slot in a fine-grained TDM schedule corresponds to a single resource access, while coarse-grained TDM arbitration considers slots with fixed or variable duration that are typically much longer. A similarity between all the above-mentioned works is that they assume the TDM schedule to be given and hence require an efficient configuration methodology to provide good results, showing the relevance of research in this area.

A TDM configuration methodology has been proposed in [10] that heuristically determines and optimizes bus schedules to satisfy the response time requirements of task graphs.

Key differences with our work is that it primarily targets coarse-grained schedules and relies on application information about when requests are issued. In contrast, our work considers fine-grained TDM arbitration and provides a service guarantee that is independent of the application model.

Other methodologies to configure fine-grained TDM arbiters have been proposed in the context of off-chip and on-chip networks. An approach for synthesizing TDM schedules for TTEthernet with the goal of satisfying deadlines for time-triggered traffic, while minimizing the latency for rate-controlled traffic is proposed in [16]. The methodologies in [17], [18] furthermore consider slot allocation in contention-free TDM networks-on-chips. Just like [10], the approaches in [17], [18] are heuristic and their efficiencies have not been quantified compared to optimal solutions. Furthermore, the problem of scheduling networks is different from ours as it considers *multiple resources* (network links) and is dependent on the problem of determining paths through the network.

Our work is *different from the aforementioned methodologies in that it provides a latency-rate guarantee, while optimally minimizing the total allocated rate for single resources shared by a fine-grained TDM arbiter*. The only previous solution to this problem is the configuration methodology for multi-channel memory controllers in [19]. Just like our work, this approach to TDM configuration is based on integer-linear programming, although the formulation is limited to continuous slot assignment and assumes the frame size to be given. These restrictions result in significant over-allocation that often prevents a given set of requirements from being satisfied. In contrast, *our methodology supports arbitrary slot assignment and synthesizes the frame size*. As we experimentally show in Section VIII, this causes our methodology to consistently outperform the work in [19].

III. Background

This section presents relevant background information to understand the work in this paper. First, we present the concept of latency-rate servers, which is an abstraction of the service provided to a client by a shared resource. We then proceed by discussing how TDM arbitration fits with this abstraction.

A. Latency-Rate Servers

Latency-rate ($\mathcal{LR}$) [5] servers is a shared resource abstraction that guarantees a client c_i sharing a resource a minimum allocated rate (bandwidth), ρ_i, after a maximum service latency (interference), Θ_i, as shown in Figure 1. The figure illustrates a client requesting service from a shared resource over time (upper solid line) and the resource providing service (lower solid line). The $\mathcal{LR}$ service guarantee, the dashed line indicated as service bound in the figure, provides a lower bound on the amount of data that can be transferred to a client during any interval of time. This makes a $\mathcal{LR}$ server suitable for performance analysis of streaming applications concerned with the time to serve *sequences of requests* rather than just single requests. Examples of such applications are audio and video encoders/decoders [9], [20], [21], and wireless radios [9], [22]. The values of Θ_i and ρ_i of each client depend on the particular choice of arbiter and how it is configured. Examples of arbiters that belong to the class of $\mathcal{LR}$ servers are TDM, several varieties of the Round-Robin and Fair-Queuing algorithms [5], as well as priority-based

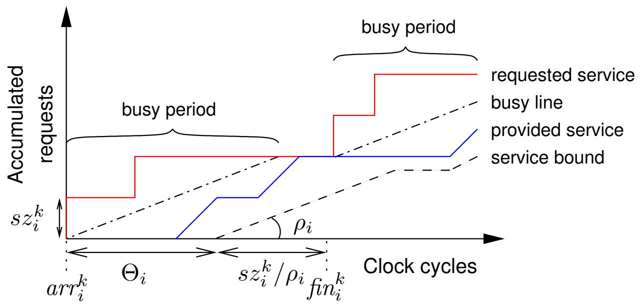


Fig. 1. A $\mathcal{LR}$ server and associated concepts for a client sharing a resource.

arbiters like Credit-Controlled Static-Priority [23]. The main benefit of the $\mathcal{LR}$ abstraction is that it enables performance analysis of systems with shared resources in a unified manner, irrespective of the chosen arbiter and configuration, using well-known performance analysis frameworks, such as network calculus [24] and data-flow analysis [25].

The $\mathcal{LR}$ service guarantee is conditional and only applies if the client produces enough requests to keep the server busy. This is captured by the concept of *busy periods*, which intuitively are periods in which a client requests at least as much service as it has been allocated (ρ_i) on average. This is illustrated in Figure 1, where the client is in a busy period when the requested service curve is above the dash-dotted reference line with slope ρ_i that we informally refer to as the *busy line*. The figure also shows how the service bound is shifted when the client is not in a busy period. A client is casually referred to as a busy client during a busy period. We now have all the necessary ingredients to provide a formal definition of a $\mathcal{LR}$ server in Definition 1.

Definition 1 ($\mathcal{LR}$ server): A server is a $\mathcal{LR}$ server if and only if a non-negative service latency Θ_i can be found such that the provided service, w_i^j, of a client c_i is bounded by Equation (1) during a busy period with duration j. The minimum non-negative constant Θ_i satisfying the equation is the service latency of the server.

$$w_i^j \geq \max(0, \rho_i \cdot (j - \Theta_i)) \quad (1)$$

It has been shown in [26] that the worst-case finishing time of the k^{th} request from a client i can be bounded using the $\mathcal{LR}$ abstraction according to Equation (2), where sz_i^k is the size of the request in number of required slots, arr_i^k is the arrival time, and fin_i^{k-1} is the worst-case finishing time of the previous request from the client. This bound is visualized for the k^{th} request in Figure 1.

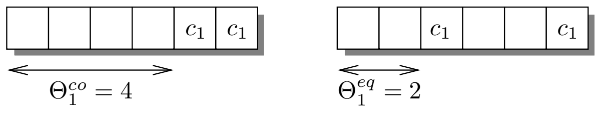
$$fin_i^k = \max(arr_i^k + \Theta_i, fin_i^{k-1}) + sz_i^k/\rho_i \quad (2)$$

B. Time-Division Multiplexing

Having introduced $\mathcal{LR}$ servers as a general abstraction of shared resources, we proceed by showing how the abstraction applies to resources shared using TDM arbitration.

A fine-grained TDM arbiter operates by periodically repeating a schedule, or frame, with a fixed number of slots, f. The schedule comprises a number of slots, each corresponding to a single resource access with bounded execution time. Every client c_i is allocated a number of slots ϕ_i in the schedule at design time.

The rate (bandwidth) allocated to a client, ρ_i, is determined purely by the number of allocated slots in the schedule and is computed according to Equation (3). The service latency, on the other hand, depends on the *slot assignment policy* that determines how the allocated slots are distributed in the schedule. A commonly used slot assignment policy is to use a *continuous allocation* [6]–[9], [19], where slots allocated to a client appear consecutively in the schedule, as shown in Figure 2a. For this policy, the service latency of a client (in slots), Θ^{co}, can simply be computed according to Equation (4). The service latency assumes that the busy period starts just after the last slot allocated to the client to maximize the number of interfering slots. If the busy period starts at any later point in the TDM schedule, the service latency is reduced.



(a) Continuous slot assignment. (b) Equidistant slot assignment.

Fig. 2. TDM schedule with frame size $f = 6$ and $\phi_1 = 2$ allocated slots to client c_1 for two different slot assignment policies. The allocated rate of c_1 is $\rho_1 = 2/6$.

$$\rho_i = \phi_i / f \tag{3}$$

$$\Theta_i^{co} = f - \phi_i = f \cdot (1 - \rho_i) \tag{4}$$

The main benefit of the continuous slot assignment policy is that it is simple to understand and implement, and that both the service latency and the rate are straight-forward to compute. However, it is provably also the policy that results in the longest possible service latency for a given number of allocated slots. In contrast, the shortest service latency is achieved by using a schedule where the allocated slots are placed equidistantly, as shown in Figure 2b. This assignment results in a service latency, Θ^{eq}, according to Equation (5). Although the service latency of an equidistant schedule is minimal, it is not always possible to create such a schedule for all clients, since several of them may require the same slot to make their allocations equidistant. As a result, *the optimal schedule for a given set of clients and requirements is more complex and irregular, and requires a more sophisticated slot assignment policy and service latency analysis.*

$$\Theta_i^{eq} = f/\phi_i - 1 = 1/\rho_i - 1 \tag{5}$$

IV. Latency Computation

This section proposes a method to compute the service latency and rate parameters of a client from an *arbitrary schedule* and proves its correctness. The key idea is to break down a TDM schedule with an arbitrary slot allocation for client c_i into N_i *sub-schedules*, each with a continuous allocation. Sub-schedule $j \in [1, N_i]$ hence comprises $\tilde{\phi}_i^j$ idle slots (slots not allocated to client c_i under analysis), followed by ϕ_i^j allocated slots. In case the first slot in the first sub-schedule is allocated to the client, then $\tilde{\phi}_i^1 = 0$. For each sub-schedule, we can apply a *local analysis*, corresponding to the analysis for continuous TDM allocations previously presented in Equations (3) and (4), to determine the local service latency, Θ_i^j, and rate, ρ_i^j, of client c_i. However, these bounds are not guaranteed to be conservative for busy periods starting and ending in arbitrary sub-schedules. This section hence presents a *global analysis* to determine the global service latency that is valid for arbitrary TDM allocations. In the remainder of this paper, we will omit the index of the client under analysis in the interest of readability unless required to disambiguate.

We start by presenting a motivational example that explains why deriving the service latency of an arbitrary TDM schedule requires a global analysis. Intuitively, it is just a matter of identifying the largest gap between slots allocated to the client in the schedule, i.e. using the local analyses, although this is not correct. The difficulty lies in the definition of service latency; that it is about the maximum time before the allocated rate is guaranteed to be *continuously provided* during a busy period and that this busy period may start at any point in the schedule and last for an arbitrary number of sub-schedules.

The problem is illustrated in Figure 3, which shows a TDM schedule, repeated twice, along with its corresponding service bound. The schedule has a frame size $f = 10$, where five slots are allocated to client c_1 resulting in a global rate of $\rho = 0.5$. The schedule comprises two sub-schedules with continuous allocations, one with a local size of $f^1 = 4$, where c_1 is allocated 1 slot ($\phi^1 = 1$), and one with a local size of $f^2 = 6$ and c_1 has an allocation of $\phi^2 = 4$ slots. Given Equations (3) and (4), the local service latencies and rates are $\Theta^1 = 3$, $\rho^1 = 0.25$, $\Theta^2 = 2$, $\rho^2 = 0.67$, respectively. The provided service line in Figure 3 corresponds to the actual worst-case provided service by the schedule without considering the $\mathcal{LR}$ abstraction. The dotted line starting after the third TDM slot shows that assuming the service latency of the schedule to be equal to the largest local service latency results in a non-conservative service guarantee, as it is intersected by the provided service line. The reason is that although $\Theta^1 = 3$ is the longest gap between allocated slots, there is not enough allocated slots in sub-schedule 1 to maintain the allocated rate during the idle slots in the following sub-schedule. Hence, the global service latency does not only depend on a single sub-schedule, but also on the following ones.

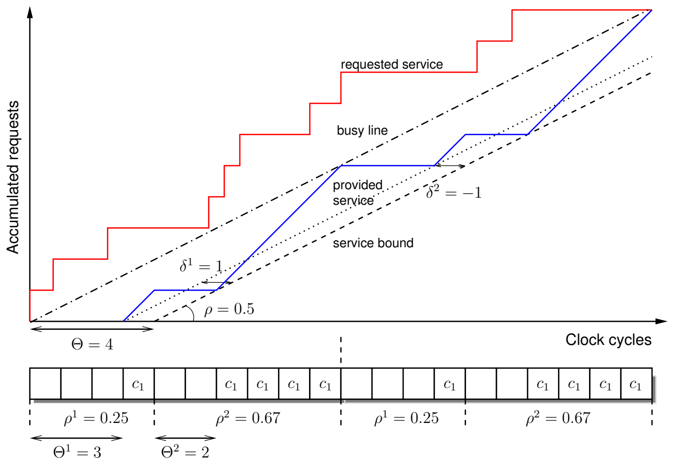


Fig. 3. A TDM schedule with $f = 10$ comprising two sub-schedules and an example of the corresponding $\mathcal{LR}$ server.

The solution to this problem is to use a global analysis that considers the interaction between sub-schedules. For this purpose, Definition 2 defines a parameter called *service latency* *offset*, δ^j, for each sub-schedule j that represents the ability of the allocated slots in the sub-schedule to maintain the global rate guarantee through the idle part of the following sub-schedule. If the offset of sub-schedule j is positive, its value indicates how much the local service latency Θ^j must be increased for the guarantee to be valid also throughout sub-schedule $j+1$. This can intuitively be understood as the analysis shifts the allocated slots in sub-schedule j δ^j steps to the right in the TDM schedule. This shift reduces the number of idle slots in sub-schedule $j+1$ in the analysis, while increasing the number of idle slots in sub-schedule j. As a result, the guaranteed rate is now consistent with the allocated rate at cost of an increase in service latency.

Definition 2 (Service latency offset): The service latency offset of a sub-schedule j for a client is denoted by δ^j and is defined as $\delta^j = \phi^j + \tilde{\phi}^{j+1} - \phi^j \cdot 1/\rho$.

Computing the offset for the example in Figure 3 gives $\delta^1 = 1$, implying that the service latency has to be increased by one for the global rate to be maintained without interruption. This is illustrated by the service bound in Figure 3, where increasing the local service latency by one to $\Theta = 4$ is just enough to make the global rate guarantee conservative during the second sub-schedule.

Lemma 1 shows an important property of the service latency offset, namely that the sum of offsets for all sub-schedules is equal to zero. Intuitively, this means that the combinations of the local rates always converge to the global rate, which is true by definition. This property can also be seen in Figure 3.

Lemma 1 (Convergence of local rates): The sum of service latency offsets of a client for all N sub-schedules equals zero, i.e., $\sum_{j\in N} \delta^j = 0$.

Proof: The proof of the lemma is divided into three small steps, as shown in Equation (6). The first step simply substitutes δ^j for its definition from Definition 2. The second step then uses the fact that the sum of ϕ^j and $\tilde{\phi}^{j+1}$ of the client over all sub-schedules are equal to its global ϕ and $\tilde{\phi}$, respectively. The final step then exploits that $\phi+\tilde{\phi} = f$ and that $\rho = \phi/f$, and then the proof follows by algebraic reduction.

$$\sum_{j\in N} \delta^j = \sum_{j\in N} (\phi^j + \tilde{\phi}^{j+1} - \phi^j \cdot 1/\rho) = \\ \phi + \tilde{\phi} - \phi \cdot 1/\rho = f - \phi \cdot f/\phi = 0 \quad (6)$$

□

The global service latency can finally be computed according to Theorem 1. Equation (7) in the theorem computes the maximum latency for busy periods starting in any sub-schedule (variable j) and ending in any sub-schedule (variable l). Note that it is sufficient to consider N (the number of sub-schedules) possible end-points, since it follows from Lemma 1 that the same offsets reoccur for every sub-schedule for every iteration of the TDM schedule. For clarity, the indices in Theorem 1 do not show wrapping around the N sub-schedules, although this is just a matter of a simple modulo operation.

Theorem 1 (Service latency for arbitrary TDM schedules): The service latency of a client for an arbitrary TDM schedule is determined according to Equation (7).

$$\Theta = \max_{j\in[1,N]} \Big(\Theta^j + \max\big(0, \max_{k\in[1,N]} \sum_{l=j}^{j+k-1} \delta^l\big)\Big) \quad (7)$$

Proof: We prove the theorem by showing that the $\mathcal{LR}$ guarantee in Definition 1 holds during a busy period starting in any sub-schedule j and ending in any sub-schedule $k \geq j$. We will show this assuming the worst-case start and end points within the sub-schedules, which is in the beginning of the starting sub-schedule j and just before the set of consecutive slots allocated to the client in sub-schedule k.

The proof will show that the provided rate during the busy period, ρ^*, is greater than or equal to the allocated rate, i.e. $\rho^* \geq \rho$. The provided rate during the busy period is computed by considering the number of allocated slots in the interval over the total number of slots after waiting for the global service latency, as shown in Equation (8). Here, the number of slots indicated by the sum of offsets are discounted from the total number of slots. The reason is that these slots are added to the global service latency and should hence not be included in the rate component of the service bound, since there is no obligation to deliver on the allocated rate during the initial Θ slots. However, the slots allocated during this time still counts towards the guarantee, even though service is provided early, since it is just a matter of staying above the guaranteed service line of the $\mathcal{LR}$ server.

$$\rho^* = \sum_{m=j}^{k-1} \phi^m / \Big(\phi^m + \tilde{\phi}^{m+1} - \max\big(0, \max_{n\in[0,k-j-1]} \sum_{m=j}^{j+n} \delta^m\big)\Big) \quad (8)$$

We distinguish two mutually exclusive and jointly exhaustive cases considering the starting and ending sub-schedules.

Case 1: $j < k$
This describes the general case where the busy period starts in a sub-schedule j and ends in a later sub-schedule k. From Equation (7), we get that the service latency in this case is expressed by $\Theta^{j,k} = \Theta^j + \max(0, \sum_{m=j}^{k-1} \delta^m) \leq \Theta$.

The provided rate is expressed according to Equation (8). Removing the max expressions and just summing up the offsets from j to $k-1$ results in a conservative minimum rate, shown in Equation (9), since the same number of allocated slots are provided over a potentially larger number of total slots.

$$\rho^* \geq \sum_{m=j}^{k-1} \phi^m / \big(\phi^m + \tilde{\phi}^{m+1} - \sum_{n=j}^{k-1} \delta^n\big) \quad (9)$$

In the final step, we substitute δ^j for its definition from Definition 2 and conclude the proof by algebraic reduction.

$$\rho^* \geq \sum_{m=j}^{k-1} \phi^m / \Big(\phi^m + \tilde{\phi}^{m+1} - \sum_{n=j}^{k-1} (\phi^n + \tilde{\phi}^{n+1} - \phi^n \cdot 1/\rho)\Big) = \rho$$

Case 2: $j = k$
We now consider the special case where the busy period starts and stops in the same sub-schedule j, confining the busy period to a single sub-schedule with continuous slot

assignment. This makes the case identical to the simple case previously discussed in Section III-B and the service latency can hence be computed according to Equation (4). This is covered by Equation (7), for the case where $k = N$, since $\sum_{j \in N} \delta^j = 0$ by Lemma 1.

Assuming the worst-case start and end points in sub-schedule j, $\rho^* = 0$, which satisfies the $\mathcal{LR}$ guarantee since no service is expected to be provided until after the global service latency $\Theta \geq \Theta^j$ according to Definition 1.

Together, the two cases prove that even under worst-case alignment with the TDM schedule, a busy client is guaranteed a minimum rate of ρ after a global service latency Θ, given by Equation (7). The global service latency, Θ, is furthermore minimal, since any smaller value would result in $\rho^* < \rho$ for the case where n is maximal in Equation (8). $\square$

V. Problem Formulation

So far, we have discussed how to derive the $\mathcal{LR}$ guarantees provided to a client from a given TDM schedule with an arbitrary slot assignment. However, we have not yet addressed the problem of finding a frame size and TDM slot allocation that satisfies the requirements of a *set of clients*, while minimizing the rate allocated to real-time clients. Towards this, this section first formulates the problem, which we refer to as the *TDM Configuration Problem/Latency-Rate* (TCP/LR), and then shows that it is NP-hard. Section VI later proposes an integer-linear programming formulation to solve the problem.

An instance of the TCP/LR problem is defined by a tuple of requirements $\langle C, \hat{\Theta}, \hat{\rho} \rangle$, where:

- $C = \{c_1, ..., c_n\}$ is the set of real-time clients that share a resource, where n is the number of clients.
- $\hat{\Theta} = [\hat{\Theta}_1, \hat{\Theta}_2, ..., \hat{\Theta}_n] \in \mathbb{R}^n$ and $\hat{\rho} = [\hat{\rho}_1, \hat{\rho}_2, ..., \hat{\rho}_n] \in \mathbb{R}^n$ are given *service latency* (in number of TDM slots) and *rate (bandwidth)* (required fraction of total available slots) *requirements* of the clients, respectively.

To satisfy the given requirements of a problem instance, we proceed by formalizing a TDM schedule and its associated parameters:

- f denotes the TDM frame size, $f \in \mathbb{Z}^+$. It is bounded according to $\underline{f} \leq f \leq \bar{f}$, where $\underline{f}$ and $\bar{f}$ are lower and upper bounds, respectively. The set $F = \{1, 2, \cdots, f\}$ denotes TDM slots.
- $S = [s_1, s_2, ..., s_f]$ is a schedule we want to find, where $s_i \in \{C \cup \varnothing\}$ indicates the client scheduled in slot i or $\varnothing$ (empty element) if the slot is not allocated.
- $\phi = \{\phi_1, \phi_2, ..., \phi_n\}$ is the number of slots allocated to each client, i.e. $\phi_i = \mid \{s_j\} : s_j = c_i \mid$.
- $\Theta = [\Theta_1, \Theta_2, ..., \Theta_n] \in \mathbb{R}^n$ and $\rho = [\rho_1, \rho_2, ..., \rho_n] \in \mathbb{R}^n$ are the *service latency* and *allocated rate*, respectively, provided by the TDM schedule.

The goal of TCP/LR is to find a schedule S for n clients sharing the resource such that the service latency and rate constraints (Constraints 1 and 2 below) are fulfilled and the objective function, Φ, being the total allocated rate to the real-time clients in C is minimized, as shown in Equation (10). This ensures that all real-time requirements are satisfied while maximizing the unallocated resource capacity available to non-real-time clients, thus maximizing their performance.

Constraint 1: The bandwidth (rate) requirements of all clients must be satisfied, i.e. $\rho_i \geq \hat{\rho}_i,\ c_i \in C$

Constraint 2: The service latency requirements of all clients must be satisfied, i.e. $\Theta_i \leq \hat{\Theta}_i,\ c_i \in C$

$$\textit{Minimize:} \sum_{c_i \in C} \rho_i = \Phi \tag{10}$$

After formulating the TCP/LR problem, Theorem 2 states that TCP/LR is NP-hard, which justifies our ILP-based approach to find optimal solutions. The proof is based on a polynomial transformation from the Periodic Maintenance Scheduling Problem (PMSP) [27] and is found in Appendix A.

Theorem 2 (NP-hardness): TCP/LR is NP-hard.

VI. ILP Model

Having established that TCP/LR is NP-hard, we know that there exist no algorithms with polynomial complexity that solves the problem optimally unless *P=NP*. An approach to find optimal solutions based on integer-linear programming (ILP) is hence justified, as there are available solvers to efficiently explore the vast solution space. In this section, we start by presenting an ILP model of our problem using only four simple constraints. After this, we present five optimizations of the model that significantly reduce the solution space and the computation time of the solver.

A. Basic Model

We now present the basic ILP formulation. This model assumes that the frame size, f, is fixed and that the optimal frame size is found by simply running the model for all possible values of f in $[\underline{f}, \bar{f}]$ and picking the value that minimizes the criterion. Later, we will introduce both exact and heuristic methods for pruning the number of different frame sizes to try. The proposed model is based on the time-indexed scheduling approach [28]. This means that for each client $c_i \in C$, there are exactly f binary variables x_i^j, defined according to:

$$x_i^j = \begin{cases} 1, & \text{if slot } j \text{ is allocated to client } c_i. \\ 0, & \text{otherwise.} \end{cases}$$

Reformulating the minimization criterion from Equation (10) in terms of this variable, results in Equation (11) that aims to minimize the total number of allocated slots in the given frame size f.

$$\textit{Minimize:} \frac{\sum_{c_i \in C} \sum_{j \in F} x_i^j}{f} \tag{11}$$

The solution space is defined by four constraints. The first two constraints consider slot allocation. Constraint 3 simply states that a slot can be allocated to maximally one client. Constraint 4 then dictates that enough slots must be allocated to a client to satisfy its rate requirement, which is computed according to Equation (3).

Constraint 3: Each slot is allocated to at most one client.

$$\sum_{c_i \in C} x_i^j \leq 1, \qquad j \in F.$$

Constraint 4: The number of slots allocated to a client c_i must be greater than or equal to the number of slots required to satisfy its rate requirement, $\hat{\rho}_i$.

$$\sum_{j=1}^{f} x_i^j \geq f \cdot \hat{\rho}_i, \qquad c_i \in C.$$

The following two constraints focus on the worst-case provided service offered by the TDM schedule to a client, $\underline{w}$ ($\underline{w} \leq w$). Constraint 5 states that the worst-case provided service to a client c_i during a busy period of any duration j starting in any slot k cannot be larger than the service provided by its allocated slots. The worst-case provided service corresponds to the solid line labeled 'provided service' in Figure 3. Constraint 6 states that the worst-case provided service of the client, $\underline{w}_i^j$, must satisfy its $\mathcal{LR}$ requirements and is a straight-forward implementation of Definition 1.

Constraint 5: The worst-case provided service of client c_i, $\underline{w}_i^j$, cannot exceed the service provided by the TDM schedule during a busy period with duration j.

$$\underline{w}_i^j \leq \sum_{l=k}^{(k+j) \bmod f} x_i^l, \qquad k \in F,\, c_i \in C,\, j \in F.$$

Constraint 6: The worst-case provided service of a client c_i must satisfy the $\mathcal{LR}$ guarantee corresponding to its required service latency and rate, $\hat{\rho}_i$ and $\hat{\Theta}_i$, for a busy period with duration j.

$$\underline{w}_i^j \geq \hat{\rho}_i \cdot (j - \hat{\Theta}_i), \qquad j \in F,\, c_i \in C.$$

B. Optimization of Computation Time

After introducing the basic ILP model of TCP/LR, we proceed by presenting five optimizations to the formulation that reduce the computation time of the solver, while preserving the optimality of the solution. The key is exploiting more information about the problem to further constrain the solution space, or to remove constraints when they are not useful.

The first optimization exploits that an *increased lower bound on the number of slots allocated to a client*, $\underline{\phi}_i$, may be found by considering the number of slots required to satisfy its service latency requirement instead of solely focusing on the rate requirement. However, the number of slots required to satisfy the latency requirement depends on where the slots are allocated in the frame, which is not yet known. This optimization conservatively addresses this by assuming an equidistant allocation, previously shown in Figure 2b, since this optimal assignment lower bounds the required slots to satisfy the service latency requirement. This optimization effectively prunes the solution space in cases where the required rate is low, but the service latency requirement is tight. Constraint 7, which replaces Constraint 4, captures this optimization.

Constraint 7: The minimum number of slots allocated to a client c_i, $\underline{\phi}_i$, is the minimum number required to satisfy both its service latency requirement and its rate requirement.

$$\sum_{j=1}^{f} x_i^j \geq \underline{\phi}_i = \max(\lceil \hat{\rho}_i \cdot f \rceil, \left\lceil \frac{f}{\hat{\Theta}_i + 1} \right\rceil), \qquad c_i \in C.$$

The second optimization *removes redundant constraints* generated by Constraints 5 and 6, thereby reducing computation time. As one can see, $f^2 \cdot n$ constraints are generated by Constraint 5 and $f \cdot n$ constraints by Constraint 6. However, it is not necessary to generate Constraints 5 and 6 for $j < \hat{\Theta}_i$, since the service bound provided by the $\mathcal{LR}$ guarantee is always zero in this interval by Definition 1. This is clearly seen in Figure 1. Removing these unnecessary constraints is particularly helpful in cases where the service latency requirement is large.

Additional constraints can be removed if more slots are required to satisfy the service latency requirements than the rate requirements, i.e. when the second term in the max-expression in Constraint 7 is dominant. In the remainder of this paper, we refer to clients with this property as *latency-dominated*, as opposed to *bandwidth-dominated*, clients. Lemma 2 shows that the offsets of all sub-schedules must be equal to zero for latency-dominated clients, which means that their service latency becomes equal to the largest gap in the TDM schedule. This case allows Constraints 5 and 6 to only be generated for a single point per client, namely $j = \lfloor \hat{\Theta}_i \rfloor + 1$, $c_i \in C$. This confirms that the client never waits longer than its service latency requirement for an allocated slot and since all offsets are equal to zero, we know that the allocated rate is guaranteed no matter in what sub-schedule the busy period ends.

Lemma 2 (Offsets of latency-dominated use-cases): All offsets of all sub-schedules for latency-dominated clients are equal to zero, i.e. $\hat{\rho}_i \cdot f \leq \frac{f}{\hat{\Theta}_i+1} \rightarrow \delta_i^l = 0, l \in N_i, c_i \in C$

Proof: The proof comprises four simple steps, where the first three are shown in Equation (12). The first step substitutes δ^l for its definition from Definition 2. The second step then exploits that $\hat{\rho}_i \cdot f \leq \frac{f}{\hat{\Theta}_i+1}$ implies $\hat{\Theta}_i + 1 \leq \frac{1}{\hat{\rho}_i}$ and makes a substitution. The third step is an algebraic reduction and realizing that the expression is less than or equal to zero, since the idle slots in each sub-schedule is bounded according to $\tilde{\phi}^{l+1} \leq \hat{\Theta}_i$ and that $\phi_i^j \geq 1$ by the definition of a sub-schedule.

$$\delta^l = \phi_i^l + \tilde{\phi}_i^{l+1} - \phi_i^l \cdot 1/\hat{\rho}_i \leq \phi_i^l + \tilde{\phi}_i^{l+1} - \phi_i^l \cdot (\hat{\Theta}_i + 1) = \tilde{\phi}_i^{l+1} - \phi_i^l \cdot \hat{\Theta}_i \leq 0 \quad (12)$$

Lastly, we know from Lemma 1 that $\sum_{l \in N_i} \delta^l = 0$, which implies that there cannot be a $\delta_i^j < 0$ unless there is a $\delta_i^k \geq 1, j \neq k$. This proves the lemma, since the only remaining possibility is $\forall l \in N_i, \delta_i^l = 0$. □

The third optimization reduces the solution space by *reducing rotational symmetry*. This means that for any given TDM schedule, $f-1$ similar schedules can be generated by rotating the given schedule and wrapping around its end. The problem is that all these schedules have the same criterion value and only one of them needs to be in the considered solution space. Constraint 8 addresses this problem by adding a constraint that fixes the allocation of the first slot to the client with the smallest $\underline{\phi}_i$ (defined in Constraint 7). This particular choice of client has been experimentally determined to significantly reduce the computation time of the solver.

Constraint 8: The first slot in the TDM schedule is allocated to client c_k, who has the smallest lower bound on number of allocated slots, i.e. $x_k^1 = 1, k = argmin_{c_i \in C}\ \underline{\phi}_i$.

Since the considered model works with a given frame size, finding the frame size that minimizes the criterion requires

$\overline{f} - \underline{f} + 1$ invocations of the model. To speed up this iteration, the fourth optimization exploits *value propagation* by passing the minimum criterion value found so far to future iterations. This optimization allows the solver to terminate an instance of a model with a particular frame size as soon as it realizes that it cannot outperform the previously best value, $\overline{\Phi}$. To maximize the benefit, frame sizes are considered in increasing order with the aim of quickly finding reasonable values to propagate from smaller problem instances. This optimization is stated in Constraint 9.

Constraint 9: The criterion for the considered frame size must be less than or equal to the lowest criterion found using any previously tested frame size.

$$\frac{\sum_{c_i \in C} \sum_{j \in F} x_i^j}{f} \leq \overline{\Phi}$$

The fifth and final optimization tries to *quickly prune frame sizes that cannot result in feasible or improved solutions due to discretization*. The problem is that the frame size defines the minimum allocation granularity for the rate (steps of $1/f$), which results in that smaller frame sizes offer coarse-grained allocations. This causes considerable discretization of the rate, which can be seen in the first term of the max-expression in Constraint 7. They key idea here is to quickly check if this discretization causes the total rate that must be allocated to satisfy the rate requirements of all the clients to be larger than the total available rate or the best total allocated rate with a previously tested frame size. Formally, this means that Equation (13) must hold for the particular frame size to be evaluated by the solver. The benefit of this optimization is that it may allow many smaller frame sizes to be skipped for use-cases with high total load from real-time clients.

$$\frac{\sum_{c_i \in C} \underline{\phi}_i}{f} \leq \overline{\Phi} \tag{13}$$

VII. Heuristic Frame Size Filtering

The previous section presented a simple ILP formulation of our NP-hard TCP/LR problem using only four constraints. Five optimizations were then presented to reduce the computation time required by the solver to find an optimal solution. However, despite the presented optimizations, the optimal approach may suffer from scalability problems when considering more complex systems with more clients and larger frame sizes. To address this problem, this section presents a heuristic frame-filtering approach based on a simple notion of goodness to provide near-optimal solutions in just a fraction of the time.

The heuristic, called the K-heuristic, is shown in Algorithm 1. The algorithm has two inputs, the set of possible frame sizes, F, and an integer K that determines how many of the $|F|$ candidate frame sizes to explore with the solver. The output is an ordered set F' of selected candidates. The K-heuristic starts by quickly evaluating the goodness, g, of all possible frame sizes. The notion of goodness, expressed on Line 3, is determined by how well the allocation granularity of each frame size fits with the minimum required rate to satisfy both bandwidth and service latency requirements (Constraint 7) of all clients. This is computed by subtracting the minimum required rates of the clients, $\underline{\phi}_j/f$, from their non-discretized counterparts to get the total amount of over-allocation due to discretization. Next, the set of frame sizes is sorted in ascending order based on the computed goodness, g_i, (Line 5) and concludes by returning the K first entries, the ones with the least potential for over-allocation due to discretization, in the ordered set F' of candidate frame sizes.

Algorithm 1 K-heuristic

```
1: Inputs: $F; K \in \mathbb{N}, K \leq |F|$
2: for all $f_i \in F$ do
3:     $g_i = \sum_{c_j \in C}(\underline{\phi}_j - \max(\hat{\rho}_j \cdot f, \frac{f}{\hat{\Theta}_j + 1}))/f$
4: end for
5: sort $F$ ascending based on $g_i$
6: $F' = F[1 : K]$
7: Output: $F'$
```

The simple goodness metric used by the K-heuristic to minimize the total allocation is particularly well-suited for bandwidth-dominated clients (first term is dominant in max-expression in Constraint 7) whose allocation is often completely determined by the discretization of the bandwidth requirement. For these clients, the K-heuristic gives optimal results unless the slot assignment causes additional slots to be allocated to satisfy service latency requirements. This is only expected to happen if a bandwidth-dominated client is on the border of being latency dominated. Latency-dominated clients may need additional slots to satisfy their service latency requirements, as it may be difficult to find equidistant schedules, and the number of added slots depends on the frame size as well as the requirements of all other clients. There hence is no simple way to statically determine which frame size is better for these clients. However, as we experimentally show in Section VIII, the K-heuristic still provides good results in these cases as it tends to prefer larger frame sizes, which generally perform well with respect to over-allocation.

VIII. Experimental Results

This section experimentally evaluates the proposed TDM configuration methodology. First, the experimental setup is explained, followed by an experiment that evaluates the scalability of the approach and shows the trade-off between criterion value and computation time with and without the heuristic frame-filtering method. These approaches are furthermore compared to the continuous slot assignment algorithm used in [6]–[8], [19]. The approach is also demonstrated for a case study of a HD video and graphics processing system, where 7 clients share a memory.

A. Experimental Setup

Experiments are performed using two sets of 3×500 synthetic use-cases, each comprising 4, 8 or 16 clients. The two sets are bandwidth-dominated use-cases and latency-dominated use-cases, respectively. We proceed by explaining how bandwidth (rate) and service latency requirements are generated for the two sets.

Parameters for synthetic use-case generation are given in Table I. Here, α is an interval from which rate requirements for each client are uniformly drawn. Firstly, rate requirements of each client in a use-case is generated. The use-case is accepted if the total required rate of all clients is in the range [0.8, 0.95] for bandwidth-dominated use-cases and [0.35, 0.5] for latency-dominated use-cases. Otherwise, it is discarded and the generation process restarts. The bandwidth requirements are lower for the latency-dominated set to leave space for over-allocation to satisfy the tighter service latency requirements. Each time the number of clients is doubled, the range of bandwidth requirements is divided by 2. This is to make sure the total load is comparable across use-cases with different number of clients, which is required to fairly evaluate scalability.

TABLE I
PARAMETERS FOR USE-CASE GENERATION

Clients	*Bandwidth dominated*		*Latency dominated*	
	α	β	α	β
4	[0.12, 0.32]	[0.7, 1.05]	[0.04, 0.14]	[1.4, 3.2]
8	[0.06, 0.16]	[0.6, 0.9]	[0.02, 0.07]	[1.35, 3.1]
16	[0.03, 0.08]	[0.5, 0.75]	[0.01, 0.035]	[1.3, 3]

Service latency requirements are uniformly distributed according to $\frac{1}{\beta \cdot \hat{\rho}}$, where a larger value of β indicates a tighter requirement. The β values are given in Table I. Each time the number of clients doubles, the lower bound is reduced by 0.1 and 0.05 for bandwidth-dominated and latency-dominated use-cases, respectively, and the upper bound for bandwidth-dominated use-cases by 0.15 and for latency-dominated use-cases by 0.1. This reduction of service latency requirements is empirically determined to provide comparable difficulty by having similar total allocated rates for the final optimal schedules. Lastly, for latency-dominated use-cases, if the total load due to the service latency requirements (second term in max-expression in Constraint 7) is outside the interval $[0.7, 0.95]$, new latency requirements for the use-case are generated. For both sets, generated use-cases that are found unfeasible using the optimal approach are discarded and replaced to ensure a sufficient number of feasible use-cases. Lastly, the maximum frame size is set to $\overline{f} = n \cdot 8$ to make sure that the number of slots available to each client is constant across the experiment.

All in all, this generation process ensures that all use-cases are feasible, have comparable difficulty, and that all clients in bandwidth-dominated use-cases are bandwidth-dominated and all clients in latency-dominated use-cases are latency-dominated. Experiments were executed on a high-performance server equipped with 2x Intel Xeon E5-2620 processor (2.10 GHz, 12 cores total) and 64 GB memory. The ILP model was implemented in IBM ILOG CPLEX Optimization Studio 12.6 and solved with the CPLEX solver.

B. Results

The experiment evaluates the scalability of the approach and the trade-off between computation time and the total rate allocated to 4, 8, and 16 real-time clients (the criterion) for the ILP formulation with and without the K-heuristic. Without heuristic filtering, the ILP formulation was executed for all possible frame sizes from $\underline{f} = n$ to $\overline{f} = n \cdot 8$ to find the optimal solution. In contrast, the K-heuristic only considered the frame size with the highest goodness ($K = 1$). Other values of K are briefly discussed later.

Figure 4 shows the distributions of the criterion (left axis) and log_{10} of the computation time (right axis) for the bandwidth-dominated use-cases for the optimal approach and the K-heuristic for 4, 8 and 16 clients, respectively. Both the optimal approach and the K-heuristic found solutions for all 1500 use-cases. The results show that although the optimal approach provides the lowest criterion values, it takes approximately 4 days to solve all 1500 use-cases with the total time for 4, 8 and 16 clients being 80 seconds, 2 hours, slightly less than 4 days, respectively. The linear increase of computation time in the figure suggests exponential growth with the number of clients and the frame size, as expected, reflecting the NP-hard nature of the TCP/LR problem. Considering the heuristic approach, we see that the K-heuristic requires less than 31.5% of the computation time compared to the optimal approach (about 30 hours). The total computation time is 40 seconds for 4 clients, 56 minutes for 8 clients, and slightly more than one day for 16 clients. *The criterion value found by the K-heuristic is optimal for all instances with 8 and 16 clients. However, it gives sub-optimal results in 8 cases out of 500 with 4 clients, although this just amounts to a negligible loss of less than 0.03% of criterion value for those 500 cases.* The reason for the sub-optimal results is that the clients in these 4 cases are on the border of being latency dominated.

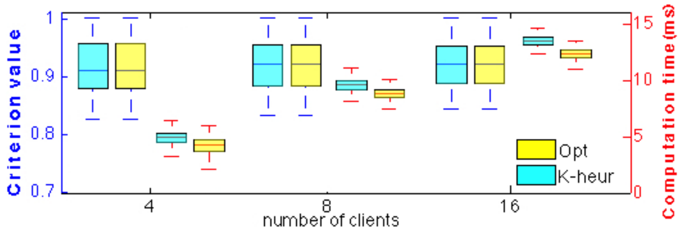


Fig. 4. Results for the bandwidth-dominated use-cases.

The results for the latency-dominated use-cases are shown in Figure 5. Here, the K-heuristic failed to solve 1/500 use-cases with 4 clients, 3/500 with 8 clients, but successfully solved all use-cases for 16 clients. The total computation time for these 1500 use-cases is 44 hours for the optimal solution. For the 1496 use-cases where solutions were found, the *K-heuristic used less than 17.8% of the computation time, while sacrificing less than 0.5% in terms of median criterion value*. The heuristic hence shows a larger deviation from the optimal criterion for the latency-dominated than for the bandwidth-dominated use-cases. This result is intuitive as the clients in these use-cases are more likely to need more than the minimum number of slots, $\underline{\phi}_j$, assumed by the heuristic, since it may not be possible to find equidistant schedules.

Increasing the value of K enables the quality of the K-heuristic to converge to that of the optimal solution at cost of higher computation time. For example, letting $K = 2$ reduces the number of failed use-cases with 8 clients from 3 to 2 while only increasing computation time with 36 seconds. In this case, all use-cases are successfully solved by using $K = 7$, which results in a total increase in computation time of 41 seconds.

From this experiment, we confirm the exponential complexity of the problem, although our implementation solves instances with 16 clients and 128 slots in less than 9 minutes on average for the 1000 use-cases. *We furthermore conclude that the K-heuristic provides near-optimal results in 28% of the time required to find an optimal solution on average.* In contrast, the commonly used continuous slot assignment algorithm only found a solution in 87/1500 (81, 6, 0) use-cases for the bandwidth-dominated set and 365/1500 (311, 53, 1) for the latency-dominated set and the criterion values

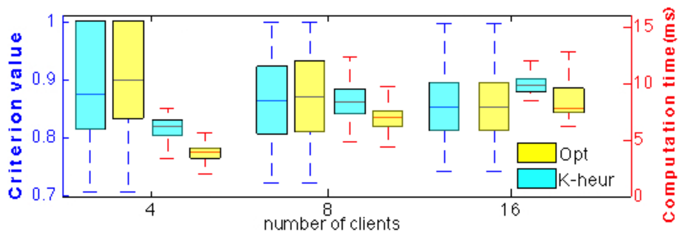


Fig. 5. Results for the latency-dominated use-cases.

of these solutions were always worse than for our heuristic approach. This poor result suggests that the applicability of the continuous assignment policy is limited.

C. Case Study

We now proceed by demonstrating our proposed TDM configuration methodology by applying it to a simple case study of an HD video and graphics processing system, where 7 memory clients share a 64-bit DDR3-1600 memory DIMM [29]. The considered system is illustrated in Figure 6. Similarly to the multi-channel case study in [19], we derive the client requirements from a combination of the industrial systems in [30], [31] and information about the memory traffic of the decoder application from [32]. However, we assume 720p resolution instead of 1080p and that all memory requests have a fixed size of 128 B to be able to satisfy the requirements with a single memory channel.

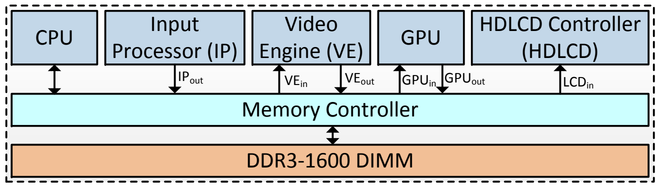


Fig. 6. Architecture of the HD video and graphics processing system.

The Input Processor receives an H.264 encoded YUV 4:2:0 video stream with a resolution of 720×480, 12 bpp, at a frame rate of 25 fps [30], and writes to memory (IP_{out}) at less than 1 MB/s. The Video Engine (VE) generates traffic by reading the compressed video and reference frames for motion compensation (VE_{in}), and writing decoder output (VE_{out}). The motion compensation requires at least 285.1 MB/s to decode the video samples at a resolution of 1280×720, 8 bpp, at 25 fps [32]. The bandwidth requirement to output the decoded video image is 34.6 MB/s.

The GPU is responsible for post-processing the decoded video. The bandwidth requirement depends on the complexity of the frame, but can reach a peak bandwidth of 50 MB/frame in the worst case [31]. Its memory traffic can be split into pixels read by GPU for processing (GPU_{in}) and writing the frame rendered by the GPU (GPU_{out}). For GPU_{in}, we require a guaranteed bandwidth of 1000 MB/s, which should be conservative given that the peak bandwidth is not required continuously. GPU_{out} must communicate the complete uncompressed 720p video frame at 32 bpp within the deadline of 40 ms (25 fps). With a burst size of 128 B, this results in a maximum response time (finishing time - arrival time) of 1388 ns per request. To provide a firm guarantee that all data from this client arrives before the deadline, we separate this into a service latency and a rate requirement according to the $\mathcal{LR}$ server approach. There are multiple (Θ, ρ) pairs that can satisfy a given response time requirement according to Equation (2), where a higher required bandwidth results in a more relaxed service latency requirement. Here, we require a bandwidth of 184.3 MB/s, twice the continuous bandwidth that is needed, to budget time for interference from other clients. According to Equation (2), this results in a service latency requirement of 718 ns (574 clock cycles for an 800 MHz memory).

The HDLCD Controller (HDLCD) writes the image processed by the GPU to the screen. It is latency critical [30] and has a firm deadline to ensure that data arrives in the frame buffer before the screen is refreshed. Similarly to GPU_{out}, HDLCD requires at least 184.3 MB/s to output a frame every 40 ms. Note that each rendered frame is displayed twice by the HDLCD controller to achieve a screen refresh rate of 50 Hz with a frame rate of 25 fps. Lastly, a host CPU and its associated Direct Memory Access (DMA) controller also require memory access with a total bandwidth of 150 MB/s to perform system-dependent activities [31].

The derived requirements of the memory clients in the case study are summarized in Table II. We conclude the section by explaining how to transform the requirements into the abstract units of rate and service latency (in slots) used by our approach. The rate is determined by dividing the bandwidth requirement of the client with the minimum guaranteed bandwidth provided by the memory controller. The service latency requirement in slots is computed by dividing the latency requirement in clock cycles by the WCET of a memory request. Given a request size of 128 B and assuming the real-time memory controller in [33], the WCET of a memory request to a DDR3-1600 is 46 clock cycles at 800 MHz and the memory guarantees a minimum bandwidth of 2149 MB/s [34]. For simplicity, we ignore effects of refresh interference in the memory, which may increase the total memory access time over a video frame with up to 3.5% for this memory. The total required bandwidth of the clients in the case study is 1839.3 MB/s. This corresponds to 85.6% of the guaranteed bandwidth of the memory controller, suggesting a suitably high load. In this use-case, all clients are bandwidth dominated for all frame sizes.

TABLE II
CLIENT REQUIREMENTS

Client	*Bandwidth [MB/s]*	*Latency [cc]*	$\hat{\rho}$	$\hat{\Theta}[slots]$
IP_{out}	1.0	-	0.0005	-
VE_{in}	285.1	-	0.1326	-
VE_{out}	34.6	-	0.0161	-
GPU_{in}	1000.0	-	0.4652	-
GPU_{out}	184.3	574	0.0858	12.5
LCD_{in}	184.3	574	0.0858	12.5
CPU	150.0	-	0.0698	-
Total	1839.3		0.8558	

We apply our configuration methodology to find the optimal TDM schedule to satisfy the client requirements, while minimizing the total allocated bandwidth. The range of considered frame sizes is set to [7, 64]. The minimum value ensures that there is at least one slot per client, while the maximum provides a reasonable trade-off between access granularity and total TDM schedule size for the number of clients in the case study. Due to the relatively small size of the system, the total computation time used by CPLEX is just over 5 seconds with all optimizations enabled. Here, we conclude that *the reduction in computation time provided by the optimizations is significant*, as only 11 out of the 58 possible frame sizes were run by the solver as others were rejected at an early stage by the discretization check (Equation (13)). Disabling only this optimization and the value propagation (Constraint 9) increases the computation time by more than a factor 5 to over 33 seconds, showing the benefits of rejecting candidate frame sizes that are not promising as early as possible.

The optimal solution was found with a frame size of 57, resulting in a total allocated rate of 0.895. This value is 2.7% lower than the value 0.922 achieved by using the largest possible frame size to achieve the finest possible allocation granularity. The minimal frame size that is able to satisfy the client requirements is 21 slots, resulting in a fully assigned TDM schedule and hence a total allocated rate of 1.

From this case study, we conclude that *optimal approaches matter*. In terms of frame size selection, 17 out of the 58 candidate frame sizes are not feasible and the difference in criterion value between the optimal and the worst feasible frame size is 10.5%. Determining the optimal frame size is not trivial and heuristically using the largest value to get the finest allocation granularity does typically not give optimal results. In contrast, our proposed K-heuristic gives optimal results for this case study. We also conclude that it is important to have optimal schedules for a given frame size, as the commonly used continuous allocation policy fails to satisfy the client requirements in this use-case for all frame sizes.

IX. Conclusions

This paper presents a methodology to configure resources shared by Time-Division Multiplexing (TDM) in a way that guarantees that bandwidth and latency requirements of real-time clients are satisfied, while minimizing their total allocation to improve performance of non-real-time clients. The problem entails both determining the number of slots in the TDM schedule (frame size) and to which clients the slots are assigned. We formalize this problem and show that it is NP-hard, and then propose an optimized integer-linear programming model to optimally solve it. However, the proposed model must be evaluated for all possible frame sizes to find an optimal solution, which may be time consuming. A heuristic algorithm, called the K-heuristic, is hence proposed to determine suitable candidate frame sizes.

We experimentally evaluate the scalability of the approach and quantify the trade-off between computation time and total allocation for the optimal and the heuristic algorithms. The results show that the heuristic provides near-optimal solutions with an average allocated bandwidth less than 0.26% from the optimum in less than 28% of the computation time. The approach is also demonstrated on a case study of a HD video and graphics processing system, where a memory is shared among 7 memory clients. Here, we show that even for a smaller system, optimal solutions can reduce the allocated bandwidth with almost 3% compared to simpler approaches. Throughout the experiments, our approach consistently outperforms the commonly used continuous allocation algorithm, which also fails to satisfy the requirements of the clients in our case study.

Acknowledgement

This work was supported by the Grant Agency of the Czech Republic under the Project GACR P103/12/1994, the Ministry of Education of the Czech Republic under project number CZ.1.07/2.3.00/30.0034 and CATRENE project CA505 BENEFIC.

References

[1] P. Kollig *et al.*, "Heterogeneous Multi-Core Platform for Consumer Multimedia Applications," in *Proc. DATE*, 2009.
[2] C. van Berkel, "Multi-core for Mobile Phones," in *Proc. DATE*, 2009.
[3] K. Goossens *et al.*, "Virtual Execution Platforms for Mixed-time-criticality Systems: The CompSOC Architecture and Design Flow," *SIGBED Rev.*, vol. 10, no. 3, 2013.
[4] S. Edwards and E. Lee, "The Case for the Precision Timed (PRET) Machine," in *Proc. DAC*, 2007.
[5] D. Stiliadis and A. Varma, "Latency-rate servers: a general model for analysis of traffic scheduling algorithms," *IEEE/ACM Trans. Netw.*, vol. 6, no. 5, 1998.
[6] S. Goossens *et al.*, "A reconfigurable real-time SDRAM controller for mixed time-criticality systems," in *Proc. CODES+ISSS*, 2013.
[7] S. Foroutan *et al.*, "A general framework for average-case performance analysis of shared resources," in *Proc. DSD*, 2013.
[8] S. Goossens *et al.*, "Conservative Open-page Policy for Mixed Time-Criticality Memory Controllers," in *Proc. DATE*, 2013.
[9] J. Vink *et al.*, "Performance analysis of SoC architectures based on latency-rate servers," *Proc. DATE*, 2008.
[10] J. Rosén *et al.*, "Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip," in *Proc. RTSS*, 2007.
[11] S. Chattopadhyay *et al.*, "Modeling shared cache and bus in multi-cores for timing analysis," in *Proc. SCOPES*, 2010.
[12] A. Schranzhofer *et al.*, "Worst-case response time analysis of resource access models in multi-core systems," in *Proc. DAC*, 2010.
[13] ——, "Timing Analysis for TDMA Arbitration in Resource Sharing Systems," in *Proc. RTAS*, 2010.
[14] G. Yao *et al.*, "Memory-centric scheduling for multicore hard real-time systems," *Real-Time Systems*, vol. 48, no. 6, 2012.
[15] T. Kelter *et al.*, "Bus-Aware Multicore WCET Analysis through TDMA Offset Bounds," in *Proc. ECRTS*, 2011.
[16] D. Tamas-Selicean *et al.*, "Synthesis of communication schedules for TTEthernet-based mixed-criticality systems," in *Proc. CODES+ISSS*, 2012.
[17] A. Hansson *et al.*, "A unified approach to mapping and routing on a network-on-chip for both best-effort and guaranteed service traffic," *VLSI design*, vol. 2007, 2007.
[18] Z. Lu and A. Jantsch, "Slot allocation using logical networks for TDM virtual-circuit configuration for network-on-chip," in *Proc. ICCAD*, 2007.
[19] M. D. Gomony *et al.*, "Architecture and Optimal Configuration of a Real-Time Multi-Channel Memory Controller," in *Proc. DATE*, 2013.
[20] S. Bhattacharyya *et al.*, "Synthesis of embedded software from synchronous dataflow specifications," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 21, no. 2, 1999.
[21] S. Stuijk *et al.*, "Throughput-buffering trade-off exploration for cyclo-static and synchronous dataflow graphs," *Computers, IEEE Transactions on*, vol. 57, no. 10, 2008.
[22] O. Moreira *et al.*, "Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor," in *Proc. EMSOFT*, 2007.
[23] B. Akesson *et al.*, "Real-Time Scheduling Using Credit-Controlled Static-Priority Arbitration," in *Proc. RTCSA*, 2008.
[24] R. Cruz, "A calculus for network delay. I. Network elements in isolation," *IEEE Trans. Inf. Theory*, vol. 37, no. 1, 1991.
[25] S. Sriram and S. Bhattacharyya, *Embedded multiprocessors: Scheduling and synchronization*. CRC, 2000.
[26] M. H. Wiggers *et al.*, "Modelling run-time arbitration by latency-rate servers in dataflow graphs," in *Proc. SCOPES*, 2007.
[27] A. Bar-Noy *et al.*, "Minimizing service and operation costs of periodic scheduling." *Mathematics of Operations Research*, vol. 27, no. 3, 2002.
[28] O. Koné *et al.*, "Event-based milp models for resource-constrained project scheduling problems," *Computers & Operations Research*, vol. 38, no. 1, 2011.
[29] *DDR3 SDRAM Specification*, JESD79-3F ed., JEDEC Solid State Technology Association, 2012.
[30] L. Steffens *et al.*, "Real-Time Analysis for Memory Access in Media Processing SoCs: A Practical Approach," *Proc. ECRTS*, 2008.
[31] A. Stevens, "Qos for high-performance and power-efficient hd multimedia," *ARM White paper, http://www.arm.com*, 2010.
[32] A. Bonatto *et al.*, "Multichannel SDRAM controller design for H.264/AVC video decoder," *Proc. SPL*, 2011.
[33] B. Akesson and K. Goossens, "Architectures and modeling of predictable memory controllers for improved system integration," in *Proc. DATE*, 2011.
[34] S. Goossens *et al.*, "Memory-Map Selection for Firm Real-Time Memory Controllers," in *Proc. DATE*, 2012.

Appendix

A. *Complexity Proof of TCP/LR*

This section proves Theorem 2 and shows that TCP/LR is NP-hard. The general strategy of the proof is to show that the decision version of TCP/LR belongs to the NP class of problems and that the Periodic Maintenance Scheduling Problem (PMSP) [27] can be polynomially transformed to the decision version of TCP/LR. Since the decision version of PMSP is known to be NP-complete, this is sufficient to prove NP-completeness of the decision version of TCP/LR. If this is the case, it follows from complexity theory that the optimization version of our problem is NP-hard. Definition 3 formulates the decision version of PMSP, followed by the decision version of TCP/LR in Definition 4.

Definition 3 (Decision version of PMSP): PMSP considers m machines with corresponding service intervals $l_1, l_2, \cdots, l_m \in \mathbb{N}$, such that $\sum_{i=1}^{m} 1/l_i \leq 1$. The problem is to check whether or not there exists an infinite maintenance service schedule of these machines in which consecutive service slots of machine i are exactly l_i time-slots apart and no more than one machine is serviced in a single time-slot.

Definition 4 (Decision version of TCP/LR): For a given criterion value Φ^*, does there exist a TDM schedule such that Constraints 1 and 2 are fulfilled and $\Phi \leq \Phi^*$?

First, we show that the decision version of TCP/LR is in NP. In this case, it is sufficient to show that a given schedule to the problem can be verified in polynomial time, which implies checking if Constraints 1 and 2 are satisfied and if $\Phi \leq \Phi^*$. By iterating over the schedule once, it is possible to determine if both Constraint 1 is satisfied for all clients and if $\Phi \leq \Phi^*$. Checking these constraints hence has linear complexity with respect to the maximum frame size, $\bar{f}$. In contrast, checking Constraint 2 using Equation (7) is linear with the number of clients and quadratic with the number of sub-schedules. Since the maximum number of sub-schedules is $\bar{f}/2$ (a sub-schedule has at least one idle and one allocated slot), it follows that verifying the service latency constraints has a complexity of $\mathcal{O}(n \cdot \bar{f}^2)$. Since each client requires at least one slot, it holds that $\bar{f} \geq n$, resulting in a complexity of $\mathcal{O}(\bar{f}^3)$. A given solution can hence be checked in polynomial time, which implies that the decision version of TCP/LR belongs to the NP set of problems.

Next, we show how PMSP polynomially transforms to TCP/LR. Consider an arbitrary instance of PMSP. The corresponding instance of TCP/LR is constructed by having m clients with rate requirements $\hat{\rho} = \{\frac{1}{l_1}, \frac{1}{l_2}, \cdots, \frac{1}{l_m}\}$ and service latency requirements $\hat{\Theta} = \{l_1 - 1, l_2 - 1, \cdots, l_m - 1\}$. Given an instance of the PMSP problem, it is possible to construct such an instance of the TCP/LR in linear time with respect to the number of clients, which implies there is a polynomial transformation. For the solution of our problem to be a valid solution to the corresponding PMSP instance, there must exist a TDM schedule with $\sum_{c_i \in C} \rho_i \leq \Phi^* = \sum_{i=1}^{n} \hat{\rho}_i$.

If there exists a schedule that fulfills the requirements of the PMSP instance, it means that it is a YES instance and that consecutive service slots of machine i are exactly l_i time-slots apart for each i. For TCP/LR, it implies a TDM schedule that has a perfectly equidistant allocation for each client. This means each client c_i has an allocated rate $\rho_i = \frac{1}{l_i}$ by Equation (3) and service latency $\Theta_i = l_i - 1$ by Equation (5). This schedule hence satisfies the created instance of TCP/LR, which means that both problem instances simultaneously answer YES.

On the other hand, if there is no schedule with perfectly equidistant slot allocation in TCP/LR then additional slots must be allocated to satisfy the service latency requirements of the created instance. This causes the value of the objective function to be $\Phi > \Phi^*$, making it a NO instance. However, the corresponding instance of PMSP is obviously a NO instance as well. The instances of PMSP and TCP/LR hence always provide YES and NO answers at the same time. Since the decision version of TCP/LR is in NP and there is a polynomial transformation of the NP-complete decision version of PMSP, this implies that the decision version of TCP/LR is also NP-complete. From this, it immediately follows that the optimization version of the problem is NP-hard.