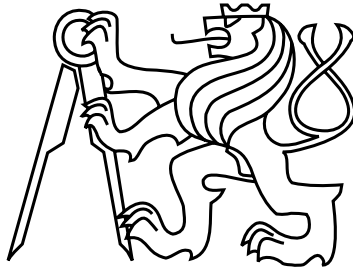


Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering



Master's Thesis

Optimal Scheduling for Time-Division-Multiplexed Real-Time Systems

Anna Minaeva

Supervisor: MSc. Benny Akesson, Ph.D.

Study Programme: Open informatics

Field of Study: Artificial intelligence

May 21, 2014

Acknowledgements

I would like to thank both of my supervisors, MSc. Benny Akesson, Ph.D. and Ing. Přemysl Šůcha, Ph.D. for their inspiring leading and helping me with this master thesis. This research and development was supported by the Technology Agency of the Czech Republic under the Centre for Applied Cybernetics TE01020197.

Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.

I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

In Prague on May 12, 2014

.....

Abstract

Nowadays, the number of applications in complex systems increases rapidly and some of them have real-time requirements, while others do not. These applications share resources, such as memories, processors or interconnect, in order to reduce cost and power consumption of the systems. However, sharing causes contention between sharing applications, which must be resolved by a resource arbiter. This master thesis uses Time-Division Multiplexing (TDM) for these purposes. TDM employs a table of a fixed length with slots allocated to the applications. Although it is a commonly used arbiter, it is a difficult problem to find a TDM frame size and slot assignment that satisfy the bandwidth and latency requirements of the real-time applications that use the resource. Moreover, it is important to minimize the resource utilization of these applications to provide non-real-time clients with better performance.

This master thesis considers the described problem and proposes an efficient methodology to solve it. The five main contributions are: 1) An optimal integer linear programming (ILP) model, assuming a given frame size is proposed; 2) An optimal branch-and-price approach for a given frame size is presented; 3) Two filtering heuristics for choosing a limited set of frame sizes for the models, mentioned above, are presented. The optimal frame size for this and the next model is found by iterating over all frame sizes in a given interval; 4) An optimal ILP model, synthesizing the frame size is presented; 5) The four approaches have been implemented and experimentally evaluated. From our experiments, we conclude that the first ILP model significantly outperforms the other optimal approaches in terms of computation time and that the heuristics provide solid reduction of computation time, sacrificing only less than 3% of the resource utilization on average.

Contents

1	Introduction	1
2	Background	3
2.1	Latency-Rate Servers	3
2.2	Time-Division Multiplexing	4
3	Problem Statement	7
3.1	Problem Formulation	7
3.2	Problem Complexity	8
4	ILP Model with a Given Frame Size	11
4.1	Model Formulation	11
4.2	Optimizations of Computation Time	12
5	Branch-and-Price Approach	15
5.1	Branch-and-Price Algorithm	15
5.2	Master Model Formulation	17
5.3	Sub-Model Formulation	18
5.4	Details of the Branch-And-Price Approach	20
6	ILP Model with Synthesized Frame Size	23
6.1	Model Formulation	23
6.2	Optimizations of Computation Time	26
7	Heuristic Frame Size Filtering	29
7.1	K-heuristic	29
7.2	KL-heuristic	30
8	Experimental Results	31
8.1	Experimental Setup	31
8.2	Experiments	32
9	Related Work	39
10	Conclusions and Future Work	41
	Bibliography	43

Chapter 1

Introduction

The number of applications in modern systems is increasing nowadays and some of them may have hard real-time requirements. It means that deadline or/and throughput requirements must be met, otherwise there is a danger of human injury or great economic loss. In contrast, there are also non-real-time applications that do not have strict deadlines, however, they are concerned about average performance. The applications are called clients and they compete for shared resources, such as memories, buses or peripherals. This sharing causes contention that must be resolved by an arbiter. Time-Division Multiplexing (TDM) is a commonly used arbiter because it is easy to understand and analyze and TDM arbitration has efficient implementations both in hardware and software.

For using TDM arbiters it is important to find a schedule. Construction of this schedule requires fulfilling both bandwidth and latency requirements of the client. To define such a schedule, two problems must be solved, determining the schedule length (frame size) and assigning time slots to the clients. Moreover, as there are non-real-time applications, resource utilization (number of allocated slots) of real-time applications must be minimized, since more left-over capacity means higher performance for non-real-time applications. On the other hand, computation time cannot be too large, as this impacts the total design time. Thus, we want to quickly find a TDM schedule for real time applications with as low utilization as possible.

This master thesis considers TDM arbitration for non-CPU resources, such as memory controllers and buses. Existing works about TDM configuration for such resources do not consider exactly the same configuration problem for at least one of the following three reasons: 1) They assign consecutive slots to the clients, which is known as the continuous allocation strategy. Although this strategy is easy to implement, understand and it requires negligible computation time, it is provably the least efficient approach in terms of resource utilization. 2) The frame size is assumed to be given, not respecting the fact that it is an important and non-trivial design decision to make. 3) Often authors only give heuristic approaches, proposed without comparison to any optimal solution, leaving the efficiency of the approach an open question.

The five main contributions of this master thesis are: 1) An optimal integer linear programming (ILP) model (ILP1) is presented. It assumes given frame size. The optimal frame size is found by running many iterations of this model with increasing frame sizes; 2) An optimal branch-and-price approach (B&P) for a given frame size is presented; 3) Two filtering

heuristics for choosing a limited set of frame sizes for ILP1 and B&P, giving a good trade-off between computation time and utilization are presented; 4) An optimal ILP model, synthesizing frame sizes (ILP2), is presented; 5) The three approaches and the heuristics have been implemented and experimentally evaluated. Their results are compared to quantify, on the one hand, the difference between different optimal approaches in terms of computation time and, on the other hand, the trade-off between computation time and utilization of the heuristics and the optimal approaches. From this evaluation, we conclude that ILP1 significantly outperforms the commonly used continuous allocation strategy in terms of utilization and that the heuristics provide solid reduction of computation time, sacrificing only less than 3% of the utilization on average.

This master thesis is organized as follows: Chapter 2 presents necessary background; Chapter 3 formulates the problem and the problem is shown to be NP-hard; Chapters 4 and 5 present the optimal approaches (ILP1 and B&P correspondingly) for a fixed frame size. In Chapter 6, one can find the complex model with synthesized frame size (ILP2), followed by Chapter 7, describing the two proposed frame-filtering heuristics. Chapter 8 presents experiments to verify the proposed claims and Chapter 9 gives a description of existing work and justifies the uniqueness of the proposed methods. Finally, Chapter 10 concludes and discusses done and possible future work.

Chapter 2

Background

This section provides a summary of relevant background information required to understand this master thesis. Firstly, the concept of latency-rate servers is presented, then the notion of TDM arbitration along with service latency and rate concepts are introduced.

2.1 Latency-Rate Servers

To begin with, the concept of latency-rate (\mathcal{LR}) servers is presented in Figure 2.1. An \mathcal{LR} server is a shared resource abstraction that ensures a client that shares a resource a minimum allocated rate (bandwidth), ρ , after a maximum service latency, Θ . The red line shows how the requests from the client are arriving at the resource over time, the blue one - how they are serviced by the resource. However, the most important feature of \mathcal{LR} servers is incorporated in the black dashed line: an \mathcal{LR} server guarantees that all of the requests of a client are serviced not later than the points on this line, i.e. the provided service line is never below the service bound.

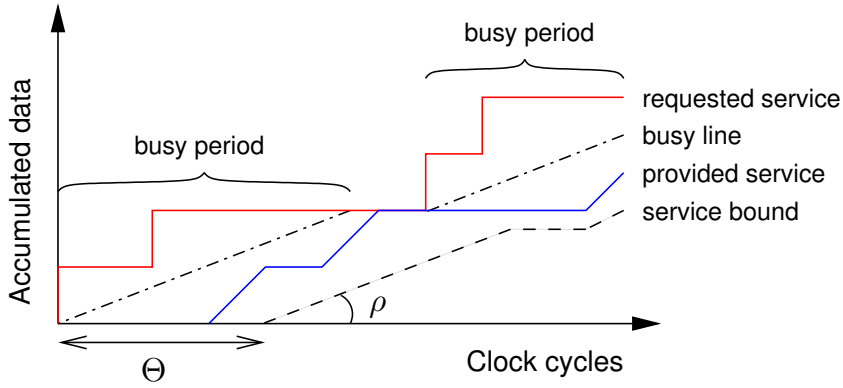


Figure 2.1: An \mathcal{LR} server.

An \mathcal{LR} server is a suitable tool for performance analysis of streaming applications, such as audio and video encoders/decoders or wireless radios, to handle sequences of requests instead

of just single ones. Considering only a single request results in a pessimistic estimation of a worst-case service time, since the time to serve r requests is typically less than r times serving one request, assuming that data arrives regularly and keeps the server busy (which is more likely the case with the streaming applications). One of the main benefits of \mathcal{LR} servers is that *based on the \mathcal{LR} guarantee, it is possible to compute the upper bound on the execution time of a client using the resource and this execution time considers sequences of requests, resulting in a better bound.*

The values Θ and ρ of a client are determined by the choice of an arbiter and how it is configured. Not every arbiter is an \mathcal{LR} server. To this class belong, among others, TDM arbiters and several varieties of the Round-Robin algorithms. The second major advantage of the \mathcal{LR} abstraction is that *performance analysis of systems with shared resources can be done in a unified manner regardless of the chosen arbiter and configuration.*

Note, however, that \mathcal{LR} service guarantees are conditional and they hold only in case the client sends enough requests for server to be busy. The concept of a busy period reflects this condition and means a period when client requests at least as much service as it has been allocated, ρ , on average. It is illustrated in Figure 2.1. The client is in a busy period, when the requested service line is above the dash-dotted black line with slope ρ .

2.2 Time-Division Multiplexing

In this section, we present the definition of the TDM arbiter, its properties and how to compute its service latency and bandwidth. As it was told in Chapter 1, TDM arbitration is used in this master thesis. It is known that TDM arbitration belongs to the class of \mathcal{LR} servers [1], therefore we can use all of the advantages of this class.

A schedule for a TDM arbiter is a table of slots allocated to the clients. It is repeating (infinitely) many times during execution of the system. One can find an example of a TDM schedule in Figure 2.2. The frame size f is defined as a number of slots in a TDM table.

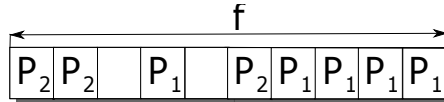


Figure 2.2: An example of a TDM table with frame size $f = 10$, shared between two clients P_1 and P_2 .

A schedule for a TDM table determines a service latency, Θ , and an allocated rate, ρ , for a client. The allocated rate is defined as the ratio of allocated slots to the TDM frame size, as it is stated in Equation (2.1). For the example in Figure 2.2 the allocated rate for the first client $\rho_1 = \frac{5}{10}$ and the second $\rho_2 = \frac{3}{10}$.

$$\rho = \frac{\phi}{f}. \quad (2.1)$$

The service latency depends on the slot assignment policy that determines how the allocated slots are distributed in the table. Mostly a continuous allocation strategy is

used [2], [3], [4], where slots are allocated to the client one after the other. Figure 2.3 shows an example of a TDM table with continuously allocated slots. The main benefit of this strategy is that it is easy to understand and implement and the service latency is computed straightforwardly by Equation (2.2), where p is the number of allocated slots. However, this is provably the least efficient strategy in terms of total allocated rate, i.e. it is required to allocate the largest amount of slots to satisfy a particular latency requirement. The opposite case is the equidistant allocation, where slots are allocated with an equal spacing and therefore it is a strategy with the least required number of allocated slots due to latency requirements. Since two or more clients may need the same slot to achieve an equidistant allocation, it is not always possible to allocate all the clients with equal spacing. Note that service latency and allocated rate are coupled in TDM, i.e. the only way to reduce service latency is to increase rate.

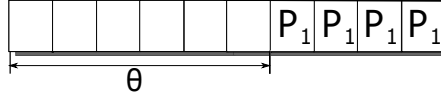


Figure 2.3: An example of a TDM table with continuous slot allocation.

$$\Theta = f - p = f \cdot (1 - \rho) \quad (2.2)$$

For an arbitrary slot assignment, intuitively, the service latency is just the largest gap between allocated slots in the table. This assumption, however, is not correct. The difficulty lies in the definition of service latency, since it is about the maximum time before the allocated rate is guaranteed to be continuously provided during a busy period. The problem is that busy period can start at any time and last for an arbitrary number of slots.

The service latency computation of a client can be done quadratically with the frame size f . The latency computation algorithm is described in detail in [5] and here only the main equations and intuition are given. The core idea of the algorithm is to split a given TDM table into a set of subtables with continuous allocation, compute their local latencies and latency offsets and afterwards combine these parameters to obtain the service latency of the client. The offset value of a subtable t Δ_t represents the ability of the allocated slots in the subtable to maintain the global rate guarantee through the non-allocated part (first $f_t - p_t$ slots) of this subtable. Basically, it means that a busy period starts in a given subtable and goes left in the TDM table, i.e we are looking at the TDM table not from left to right but from right to left. If the offset of the subtable is positive, its value shows how much the local latency L must be increased for the guarantee to be valid throughout this subtable. For example, considering the schedule for the first client in Figure 2.2, there are two subtables: one starts in the first slot and ends in the fourth and the second starts at the fifth slot and ends in the tenth. Local latencies are $L_1 = 3 + (1 - (\frac{1}{2})^{-1})$ and $L_2 = 2 + (1 - (\frac{1}{2})^{-1})$ and latency offsets are $\Delta_1 = 4 - (\frac{1}{2})^{-1} \cdot 1 = 2$ and $\Delta_2 = 6 - (\frac{1}{2})^{-1} \cdot 4 = -2$.

Equations for local latencies, L_t , and latency offsets, Δ_t , computation for each subtable t are given in Equations (2.3) and (2.4), respectively:

$$L_t = f_t - p_t + (1 - \rho^{-1}), \quad (2.3)$$

$$\Delta_t = f_t - \hat{\rho}^{-1} \cdot p_t, \quad (2.4)$$

where f_t is the subtable length, p_t is the number of allocated slots in the subtable t and ρ is the allocated rate in the whole TDM table. The local latencies are computed according to Equation (2.2) for the case of continuous slot allocation. The addend $\rho^{-1} - 1$ appears because of slightly different definitions (difference is just in subtracting a constant) of service latency between this work and [5]. More on this you can read in [6].

Afterwards the global analysis that considers interaction between subtables is done. The resulting service latency Θ for a TDM table is computed in Equation (2.5):

$$\Theta = \max_{j \in \{1, 2, \dots, \beta\}} \max_{r \in \{1, 2, \dots, \beta\}} \left(\sum_{t=r}^{r+j-1} (\Delta_{t \bmod \beta}) + L_{(r+j) \bmod \beta} + (\rho^{-1} - 1) \right), \quad (2.5)$$

where β is the number of subtables in the TDM table. Thus, the global service latency is the maximum value between the local latencies with sums of offsets of the previous subtables, where the last sum goes from 0 to β . As it was mention already, to obtain the global service latency, it is necessary to look at the busy periods that starts in each subtable and lasts for arbitrary many subtables (in Equation (2.5) it starts in subtable r and ends in subtable $(r + j) \bmod \beta$). The worst (maximum) value is a service latency of the TDM table. Note, however, that $(1 - \rho^{-1})$ and $(\rho^{-1} - 1)$ as a part of the local latencies, L , cancel each other in the global latency computation. For the TDM table in Figure 2.2 service latency is computed in Equation (2.6). Note that latency here is not equal to the largest gap.

$$\Theta = \max(L_1, L_2, \Delta_1 + L_2, \Delta_2 + L_1) = \max(3, 2, 2 + 2, -2 + 3) = 4. \quad (2.6)$$

Chapter 3

Problem Statement

The way of computing provided latency-rate guarantees from a given TDM table with arbitrary slot allocation along with the definition of TDM tables are stated earlier in Chapter 2. In this chapter, the problem of determining the best frame size value and a slot allocation, that satisfies given requirements and minimizes the total allocated rate is formulated and then this problem is proved to be NP-hard in the general case.

3.1 Problem Formulation

The considered *TDM Configuration Problem/Latency-Rate* (TCP/LR) is formalized in this section. An instance of the problem is defined by a tuple $\langle P, \hat{\Theta}, \hat{\rho} \rangle$, where:

- $P = \{1, \dots, n\}$ is the set of clients, using the resource, where n is the number of clients.
- $\hat{\Theta} = [\hat{\Theta}_1, \hat{\Theta}_2, \dots, \hat{\Theta}_n] \in \mathbb{R}^n$ and $\hat{\rho} = [\hat{\rho}_1, \hat{\rho}_2, \dots, \hat{\rho}_n] \in \mathbb{R}^n$ are given *service latency* and *bandwidth requirements* of the clients, respectively.

Note that floating point service latency requirements make sense, since requirements may be generated not in terms of TDM slots, but in terms of CPU clock cycles, that after transferring to the slots definition may result in non-integer requirements. Moreover, the service latency provided by a given TDM table could have a non-integer value, as it is given in Chapter 2 by the way service latency is computed. A TDM table is formalized in the following way:

- f denotes the TDM frame size, $f \in \mathbb{Z}^+$. It is bounded as $\underline{f} \leq f \leq \bar{f}$, where \underline{f} and \bar{f} are the TDM frame size lower and upper bounds, respectively. Set $F = \{1, 2, \dots, \bar{f}\}$ denotes TDM slots.
- $S = [s_1, s_2, \dots, s_f]$ is a schedule we want to find, where $s_i \in \{P, 0\}$. Nonzero s_i means the client number s_i is allocated to slot i and $s_i = 0$ stands for a free slot.
- $\phi = \{\phi_1, \phi_2, \dots, \phi_n\}$ is the number of allocated slots for each client, i.e.

$$\phi_i = \sum_{j \in F} |\{s_j\} : s_j = i|.$$

- $\Theta = [\Theta_1, \Theta_2, \dots, \Theta_n] \in \mathbb{R}^n$ and $\rho = [\rho_1, \rho_2, \dots, \rho_n] \in \mathbb{R}^n$ are *service latency* and *allocated rate*, respectively, provided by the TDM table. The service latency and bandwidth notions and ways of its computation is defined in Chapter 2.

The goal of TCP/LR is to find a schedule $S = \{s_1, s_2, \dots, s_f\}$ for n clients that share the resource, such that the service latency and bandwidth constraints (Constraints (3.2) and (3.3)) are fulfilled and the total allocated rate is minimized (3.1).

$$\text{Minimize : } \sum_{i \in P} \rho_i = \Phi \quad (3.1)$$

$$\rho_i \geq \hat{\rho}_i, i \in P \quad (3.2)$$

$$\Theta_i \leq \hat{\Theta}_i, i \in P \quad (3.3)$$

3.2 Problem Complexity

So far, the problem formulation is stated and the NP-hardness of the TCP/LR problem is proven here. The original proof can be found in [15] and this work has contributed to the proof.

The problems that are most similar to the TCP/LR are the pinwheel scheduling problem [8] and the Periodic Maintenance Scheduling Problem (PMSP) [10]. Since the general pinwheel problem does not have any positive or negative result about NP-completeness yet, the proof of NP-hardness of our problem is based on the PMSP. Using polynomial reduction from PMSP to the TCP/LR problem, NP-hardness is shown. The decision version of the PMSP and the decision version of the TCP/LR problems are formulated in Definition 1 and 2, respectively.

Definition 1 (Decision version of PMSP) *PMSP considers m machines and service intervals (integers) l_1, l_2, \dots, l_m such that $\sum_{i=1}^m 1/l_i \leq 1$. The problem is to check the existence of an infinite maintenance service schedule of these machines in which consecutive servicing of machine i are exactly l_i time-slots apart and no more than one machine is serviced in a single time-slot.*

Definition 2 (Decision version of TCP/LR) *With a given criterion value Φ^* does there exist a TDM schedule such that requirements (3.2), (3.3) are fulfilled and $\Phi \leq \Phi^*$? If there is such a schedule, its frame size is f^* .*

Next, Theorem 1 states NP-hardness of TCP/LR by polynomially reducing decision version of PMSP on the decision version of TCP/LR.

Theorem 1 *TCP/LR is NP-hard.*

Proof.

The first step of our proof is to show that if we have an arbitrary instance of the PMSP problem, formulated in Definition 1, it is possible to construct an instance of TCP/LR problem in polynomial time, such that these instances have YES and NO answers simultaneously. Thus, the constructed instance of the TCP/LR looks the following way: there are m clients and requirements are given as: $\hat{\rho} = \{\frac{1}{l_1}, \frac{1}{l_2}, \dots, \frac{1}{l_m}\}$, $\hat{\Theta} = \{l_1 - 1, l_2 - 1, \dots, l_m - 1\}$. And

the question is whether there exists a schedule with $\sum_{i \in F} \rho_i \leq \sum_{i=1}^f \hat{\rho}_i$. Clearly, construction of a TCP/LR instance having an PMSP instance is linear in the number of clients.

If there exists a schedule that fulfills requirements of the PMSP problem, it means, firstly, that this schedule is periodic, i.e. there is a sequence of slots that repeats infinitely many times. It is given by the fact that infinite schedule in PMSP is completely defined by the vector of the first slot occurrence of each client; secondly, the fact that it is a "YES" instance of a PMSP means that consecutive servicing of machine i are exactly l_i time-slots apart for each i . For our problem it defines a schedule that has a perfectly equidistant allocation for each client. This leads to the conclusion that the schedule would have latency exactly $\Theta = l_i - 1$ and bandwidth $\rho = \frac{1}{l_i}$. Thus, it is a "YES" instance as well.

In case we have a "NO" instance for PMSP problem, the constructed instance for our problem is "NO" instance as well since if there is no schedule with perfectly equidistant slot allocation then it is inevitable to use additional slots in TCP/LR. It causes increasing of the

criterion value to $\sum_{i \in F} \rho_i > \sum_{i=1}^f \hat{\rho}_i$, which is a violation. ■

Note that the NP-hardness of TCP/LR with an arbitrary frame size was proven. Nonetheless, for the case with fixed frame size it is always possible to set the frame size $\underline{f} = \bar{f} = f = lcm(l_1, l_2, \dots, l_m)$ and thus the theorem is valid in both cases. Naturally, the first machine would need $g_1 \cdot l_1$ slots in a schedule to fulfill requirements, the second $g_2 \cdot l_2$, ..., the m th machine needs $g_m \cdot l_m$, where $g_i \in \mathbb{Z}^+$, $i = 1, 2, \dots, m$. Indeed, the resulting periodic schedule must be a multiplier of $lcm(l_1, l_2, \dots, l_m)$.

Chapter 4

ILP Model with a Given Frame Size

Considering the NP-hardness of the TCP/LR problem, it is clear, that there is no optimal algorithm that would solve the problem in polynomial time (unless $P=NP$). Thus, the use of an ILP technique is justified. In this chapter, an ILP model formulation with four constraints is presented, followed by five optimizations, that significantly reduce the solution space and the computational time, but does not influence optimality of the solution.

4.1 Model Formulation

In this chapter, the optimization problem, described in Chapter 3, is formulated as an ILP problem. This problem is later referred to as ILP1. The frame size f is fixed, i.e. $\underline{f} = f = \bar{f}$. This model is based on the time-indexed scheduling approach [9]. It means that for each client $i \in P$, there are exactly f binary variables $x_{i,j}$,

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

As stated in Equation (3.1), allocated rate is minimized with the objective function

$$\text{Minimize : } \frac{\sum_{i \in P} \sum_{j \in F} x_{i,j}}{f}. \quad (4.1)$$

The solution space is defined by the following set of constraints. Equation (4.2) guarantees that at most one client in each slot can be allocated.

$$\sum_{i \in P} x_{i,j} \leq 1, \quad j \in F \quad (4.2)$$

Bandwidth requirements are satisfied by Constraint (4.3). Number of allocated slots for each client is required to be greater or equal than bandwidth requirement $\hat{\rho}_i$.

$$\sum_{j=1}^f x_{i,j} \geq f \cdot \hat{\rho}_i, \quad i \in P \quad (4.3)$$

An intuitive approach to guarantee service latency requirements is used in this model. Firstly, one needs to compute worst-case provided service $r_{i,j}$ offered by the TDM table to each client i starting in any slot $j \in F$ lasting for $k \in F$ slots. The worst possible service curve points are expressed by Constraint (4.4). Note that since the bandwidth requirements are already satisfied by Constraint (4.3), it is sufficient to check only the range $j \in F$.

$$r_{i,j} \leq \sum_{l=k}^{(k-j) \bmod f} x_{i,l}, \quad k \in F, j \in F, i \in P \quad (4.4)$$

The service latency requirements are guaranteed by the worst-case provided service curve points. Since \mathcal{LR} server is considered, the worst-case service curve must always lie above the service bound, as shown in Figure 2.1. Since the service bound is completely defined by bandwidth and service latency requirements, it can be expressed using Equation (4.5). In essence, the expression $\hat{\rho}_i \cdot (j - \hat{\Theta}_i)$ constructs the service bound line from Figure 2.1, where j represents the clock cycles. Thus, a given latency guarantee is defined as follows:

$$r_{i,j} \geq \hat{\rho}_i \cdot (j - \hat{\Theta}_i), \quad j \in F, i \in P. \quad (4.5)$$

4.2 Optimizations of Computation Time

By exploiting some information about the problem, five optimizations are introduced. All of them reduce the solution space and consequently decrease computational time, while preserving optimality.

The first optimization bounds the minimal number of allocated slots $\check{\phi}_i$ for each client. As a matter of fact, it is useful to bound number of allocated slots not only by bandwidth requirements, but by service latency requirements as well. Generally, it is impossible to predict the exact number of slots, necessary to satisfy the latency requirements, since it depends on the schedule, which is what we are trying to determine. It is possible though to bound the minimum number by considering equidistant slot allocation. It is easy to show that the minimum number of slots are used with this allocation strategy.

$$\sum_{j=1}^f x_i^j \geq \check{\phi}_i = \max(\lceil \hat{\rho}_i \cdot f \rceil, \left\lceil \frac{f}{\hat{\Theta}_i + 1} \right\rceil), \quad i \in P. \quad (4.6)$$

Since the solution representation allows rotational symmetry, i.e. in the best case there are f rotated schedules (ones with cyclical left- or right- shifting) that have the same objective function value, it is crucial to fix allocation of the first slot to the client with the least number of minimal required slots $w = \arg \min_{i \in P} \check{\phi}_i$. By fixing a slot to be allocated by the client with minimal possible number of slots $\check{\phi}_w$, we allow less rotationally symmetrical schedules. It happens because in the fixed position (in our case in the first slot) one can put each slot that is allocated to the client, so in the end there are maximally $\check{\phi}_w$ rotational symmetries. Therefore, choosing minimal $\check{\phi}_w$ helps us to reduce the rotational symmetry. Note that generality is not lost, since every client must be allocated by at least one slot.

Thus, Constraint (4.7) fixes allocation of the first slot to the client with the least minimal number of required slots to reduce the symmetry.

$$x_{w,1} = 1; \quad (4.7)$$

The next optimization excludes redundant constraints in the set of Constraints (4.4) and (4.5). One can see in Figure 2.1 that for $j \leq \hat{\Theta}$ constraints are fulfilled by definition, as the minimum number of provided service units should be not less than 0, which is satisfied always since number of provided service units is non-negative by definition. Therefore, it is enough to generate Constraints (4.4) and (4.5) only for $j > \hat{\Theta}$.

More interesting, in case the minimum number of allocated slots due to the latency requirements is not less than required by to the bandwidth requirements (call it *latency-dominated clients* otherwise we name it *bandwidth-dominated clients*), it is sufficient to check only the point in time right after the maximum possible gap $\hat{\Theta}$, i.e. $j = \hat{\Theta} + 1$. The intuition behind is that just satisfying latency requirements is enough to provide service guarantees during the whole gap, i.e. the bandwidth requirements are fulfilled by default. Lemma 1 states that offsets of each subtable of latency dominated clients is non-positive. It implies that Equation (2.5) always picks the maximum between the local latencies, i.e. the maximum gap in the table, which proves that we are allowed to use this optimization.

Lemma 1 *For latency-dominated clients offsets Δ_t of each subtable $t \in \{1, 2, \dots, \beta\}$ is always non-positive, $\Delta_t \leq 0$.*

Proof. Four steps are done to prove the lemma. Equation (4.8) substitutes Δ_t from Equation (2.4). Equation (4.9) then uses the fact that the client is latency dominated, i.e. $\hat{\rho} \cdot f \leq \frac{f}{\hat{\Theta} + 1}$, which means $\hat{\Theta} + 1 \leq \frac{1}{\hat{\rho}}$ and makes a substitution. Equation (4.10) is obtained by realizing that $f_t - p_t$ is the number of non-allocated slots in the subtable t and it is bounded as $f_t - p_t \leq \hat{\Theta}$. The last expression is always not more than 0 as $p_t \geq 1$ by definition of a subtable.

$$\Delta_t = f_t - \hat{\rho}^{-1} \cdot p_t \quad (4.8)$$

$$\leq f_t - p_t \cdot (\hat{\Theta} + 1) \quad (4.9)$$

$$\leq \hat{\Theta} - p_t \cdot \hat{\Theta} \leq 0 \quad (4.10)$$

■

Since optimal models with fixed frame size use a loop over frame sizes in the range $\underline{f} \dots \bar{f}$ to find the frame size with the best allocated rate, two more computational time optimizations are added. The first one is the *criterion value propagation*. Each time the solver finds any better solution $\bar{\Phi}$ with a new frame size, we save and reuse it with following frame sizes to cut branching earlier. In the beginning $\bar{\Phi} = 1 + \epsilon$ with $\epsilon > 0$. Thus, Constraint (4.11) is added to ILP1.

$$\frac{\sum_{i \in P} \sum_{j \in F} x_i^j}{f} \leq \bar{\Phi} \quad (4.11)$$

Finally, some frame sizes simply do not provide allocation granularity at the required level, i.e. the discretization of the allocation for a given frame size is always bounded by $\frac{1}{f}$. The fifth and final optimization *prunes frame sizes* that will not result in feasible or better schedules. The key idea here is to quickly check before starting the solver whether this granularity possibly can improve the solution in terms of criterion value. In case Constraint (4.12) does not hold, the frame size f_k is skipped.

$$\frac{\sum_{i \in P} \check{\phi}_i}{f_k} \leq \bar{\Phi} \quad (4.12)$$

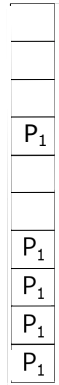
Chapter 5

Branch-and-Price Approach

The simple ILP model with fixed frame size was previously described in Chapter 4. However, we are dealing with a problem that is difficult to solve, since the LP relaxation is bad. Therefore, a column generation approach covered with a branch-and-bound method seems to be a reasonable way to solve the considered TCP/LR problem optimally. First of all, background on the branch-and-price algorithm is given. Afterwards, all the necessary components with details of the algorithm are given.

5.1 Branch-and-Price Algorithm

Branch-and-price [13] is an optimal method to solve integer linear programming problems, which combines column generation and branch-and-bound approaches. The essence of the column generation technique is to create partial solutions, columns, with help of a so called *sub-model* that are combined into a complete solution by a *master-model*. In case of the considered TCP/LR problem, columns are TDM schedules for a given client, which are combined into a complete solution for all clients by the master model. An example of a column for client P_1 related to Figure 2.2 is shown in Figure 5.1.



P_1
P_1
P_1
P_1
P_1

Figure 5.1: Example of a column for a client P_1 .

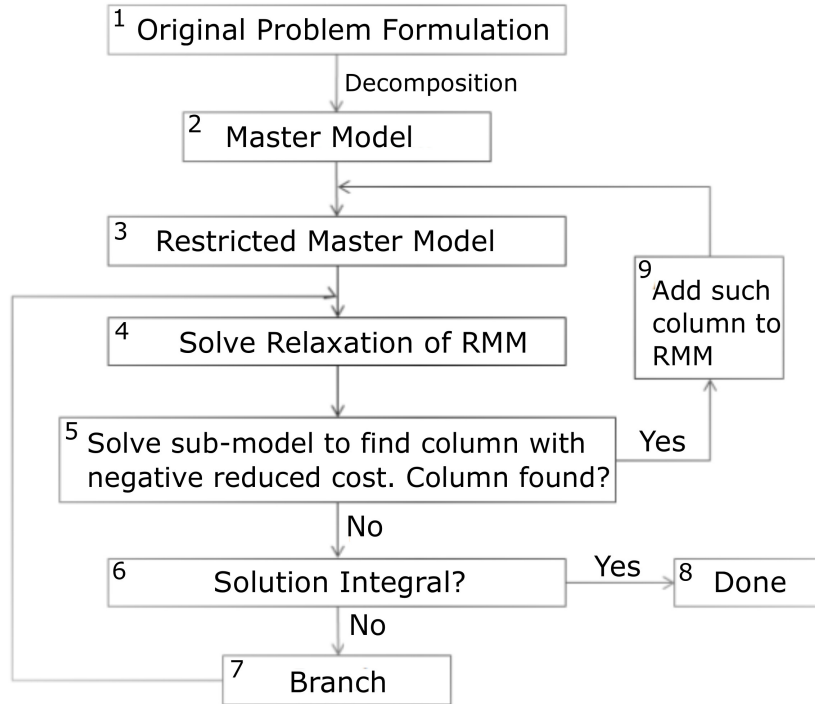


Figure 5.2: Outline of the branch-and-price algorithm [17].

The overall scheme of the branch-and-price method is shown in Figure 5.2. The master problem is extracted from the original problem by Dantzig-Wolfe decomposition [13]. Initially the master problem is considered to be an ILP, but during the column generation procedure the relaxation of this problem is solved, i.e. all decision variables are considered to be rational. The decomposition is performed to obtain a problem formulation that gives better bounds when the relaxation is solved than when the relaxation of the original formulation is solved.

Next, the notion of a restricted master problem takes place. It is impossible and unnecessary to have all the columns (typically there are a lot of them) in the master model column list. It is enough to generate only the promising columns at each iteration and due to the sub-model construction it is possible to tell whether or not better solution exists. The promising columns, from the objective function point of view, are those that have the least possible number of allocated slots and these slots are distributed in a desirable manner, i.e. the slots are not already assigned to other clients in existing chosen columns. Thus, as the master model works only with a restricted number of columns, it is called the restricted master model. To find promising columns and reduce the objective function, a sub-model is solved. This involves constructing a new column that has a negative reduced cost, which is influenced by number of allocated slots in the column and by a particular slot assignment. Formally, reduced cost is the amount by which an objective function coefficient would have to improve before it would be possible for a corresponding variable to assume a positive value in the optimal solution. To start the branch-and-price algorithm, initial feasible columns

that only have to be valid for individual clients should be generated beforehand.

If the sub-model is able to find a column with some negative reduced cost, indicating that the solution can potentially be improved, the column is added to the master model column list and the next iteration of the column generation algorithm starts. Otherwise, due to the properties of the reduced cost value (see [13]), the given solution is optimal for the master model. Then the obtained solution of the master model is checked on integrality. In case all the required decision variables are either 0 or 1, it is a candidate for being an optimal solution to the initial ILP problem (ILP1), and node is closed. If not, branching takes place and new run of column generation starts after some branching decision, explained later.

Each node in a branching binary tree contains a sequence of branching decisions, made earlier. In our case, the branching decision is defined by which slot is allocated/not allocated to a particular client. Thus, there is a partial slot assignment in each node, i.e. which slots are and are not allocated to which clients. Moreover, in every node an upper bound U on the criterion is known. In the beginning, it is set to some large value and each time a better integral solution is found, the upper bound is updated. The upper bound is required for cutting in nodes. It is done in the following way: each time the column generation procedure completed in Step 7 of Figure 5.2, the lower bound, which is the resulting optimal value of the relaxed master model, is compared to the upper bound. In case the lower bound is greater than the upper bound (in other words the current optimal solution is already worse than the best found integral solution), it does not make sense to continue with this branch, since adding new decision in the best case does not change the criterion value, and typically it becomes worse. Furthermore, every time we branch, it is necessary to somehow exclude columns that violate the decision just made, i.e. those columns that have not allocated a required slot to a required client or vice versa in case the decision is negative.

5.2 Master Model Formulation

To formalize the master model of the branch-and-price approach, decision variables indicating whether client $i \in P$ uses a schedule given by column $c \in \Omega$, are introduced:

$$\omega_{i,c} = \begin{cases} 1, & \text{if client } i \text{ uses column (schedule) } c. \\ 0, & \text{otherwise.} \end{cases}$$

In case it would be an ILP model by itself, we would intuitively let $\omega_{i,c}$ be integral, but as we are dealing with the branch-and-price technique, it is required to relax this condition and to have $0 \leq \omega_{i,c} \leq 1, i \in P, c \in \Omega$. Thus, we can interpret this values as the "probabilities" with which client i uses schedule c .

As it was said earlier, since the restricted master problem in the branch-and-price method is considered, there is no need to enumerate all of the possible feasible columns for each client. We start with an initial set of feasible columns and in each iteration new columns are added to the master model. The way of initial column generation is described in detail later in Section 5.4.

Minimization of the total allocated rate in case of the branch-and-price method is formalized in the objective function:

$$\text{Minimize : } \frac{\sum_{i \in P} \sum_{c \in \Omega} v_{i,c} \cdot \omega_{i,c}}{f} + M' \cdot \sum_{j \in F} y_j, \quad (5.1)$$

where $v_{i,c}$ is the cost (the total allocated rate), associated with schedule c ; M' is some sufficiently large number. Note that index i is put only for unification in $v_{i,c}$, and is technically not required here, since the total cost is the same regardless of which client uses the schedule. Variables y_j allow initial columns to create schedules, in which more than one client can be allocated to a single slot if the schedules/columns are combined. We call this situation overlapping. It is pretty important as it is not always possible to find any feasible solution in reasonable time. These variables are multiplied by M' to provide a dominating penalty for overlapping schedules that ensures that they are weeded out quickly.

The master model comprises only two constraints. The first one, Constraint (5.2), computes how many columns allocate the same slots and reflects it in variable y_j . If any y_j is non-zero, the solution is not feasible yet, as there exist at least two clients that share the same slot. By adding the sum of them to the criterion value, we push each slot to be allocated by maximally one client, eventually guaranteeing a feasible solution. The variables $a_{j,c}$ are in essence schedules of the columns in Ω . It takes value of 1 in case column c has allocated slot j and 0 otherwise.

$$\sum_{i \in P} \sum_{c \in \Omega} a_{j,c} \cdot \omega_{i,c} \leq y_j + 1, \quad j \in F. \quad (5.2)$$

The second and last Constraint (5.3) forces the solver to choose at least one column for each client to make sure its requirements are satisfied, or in case of the relaxation the sum of probabilities of following column c for allocation for each client must be greater than or equal to 1. Certainly, this sum in the optimal solution of the initial problem is always 1, as criterion minimization pushes this down. This trick relaxes the search space and results in a slightly lower computation time, which was proven experimentally.

$$\sum_{c \in \Omega} b_{i,c} \cdot \omega_{i,c} \geq 1, \quad i \in P, \quad (5.3)$$

where

$$b_{i,c} = \begin{cases} 1, & \text{if column } c \text{ is created for client } i. \\ 0, & \text{otherwise.} \end{cases}$$

Note that these variables are necessary in the model, since every client has its own requirements and a schedule (column), created for one client, may not satisfy the requirements of another one.

5.3 Sub-Model Formulation

As previously explained, the sub-model generates new columns for each client in such a way that reduced costs of these columns are at least negative in the general case and minimal in the used version of the approach. Thus, the criterion is the reduced price of the column

and the constraints are basically the same as in the simple ILP model (ILP1), but omitting client indices. Since we are searching for a feasible schedule that potentially improves the solution for one client, there is no use in having i indices.

Firstly, the dual ILP to the master model is created and shadow prices (values of dual variables) are extracted to use as the criterion in a sub-model. Note that the procedure of creating a dual problem formulation to an ILP does not contain design decisions, but follows straight-forwardly from the formulation of the initial ILP [14]. The dual ILP is what allows one to estimate how far the current solution is from an optimal one, which is one of the main reasons one knows when to stop even if all possible columns were not generated yet. Having $\lambda_j, j \in F$ as the dual variables for the set of Constraints (5.2) and $\sigma_i, i \in P$ for Constraints (5.3), the dual problem to the master model is formulated as:

$$\text{Maximize : } - \sum_{j \in F} \lambda_j + \sum_{i \in P} \sigma_i. \quad (5.4)$$

with respect to

$$- \sum_{j \in F} a_{j,c} \cdot \lambda_j + b_{i,c} \cdot \sigma_i \leq v_{i,c}, \quad i \in P, c \in \Omega, \quad (5.5)$$

$$\lambda_j \leq 10, \quad j \in F \quad (5.6)$$

$$\lambda_j \geq 0, \sigma_j \geq 0 \quad j \in F \quad (5.7)$$

Meanwhile, the reduced cost equals to the value given dual solution violates Constraints (5.5)-(5.7). However, the column generation procedure of the sub-model can influence only Constraints (5.5), since the dual variables λ are given in advance by the master model. Therefore, we consider only its violation. Hence, the criterion minimized in the sub-model is formulated in terms of the master model variables in (5.8):

$$\text{Minimize : } \sum_{j \in F} a_{j,c} \cdot \lambda_j + v_{i,c} - b_{i,c} \cdot \sigma_i. \quad (5.8)$$

Note that $a_{j,c}$ are simply x_j from ILP1 for the column we are searching for, for a given client. Index c does not mean anything in the sub-model as a new column is created. At the same time $v_{i,c}$ is the cost of a schedule we are looking for, $v_{i,c} = \sum_{i \in P} \sum_{j \in F} x_{i,j}$, i.e. its number of allocated slots. Values of the dual variables λ_j and σ_i are extracted from the master model after a run (after Step 4 from Figure 5.2). Finally, $b_{i,c}$ is always 1 in the sub-model, because the sub-model is always generating columns for the given client. Therefore, the sub-model has the following criterion:

$$\text{Minimize : } \sum_{j \in F} x_j \cdot \lambda_j + \sum_{j \in F} x_j - \sigma_i. \quad (5.9)$$

As a matter of fact, the last term σ_i is a constant, it is used only to check whether the obtained schedule has a negative reduced price. Consequently, the sub-model tends to find a schedule for a given client, i.e. to find $x_j, j \in F$. The constraints copy constraints of

ILP1 omitting index i . Constraint (5.10) states that the bandwidth requirement must be fulfilled, Constraint (5.11) computes points in time of the worst-case provided service line and Constraint (5.12) guarantees satisfying the service latency requirement for a given client.

$$\sum_{j=1}^f x_j \geq f \cdot \hat{\rho} \quad (5.10)$$

$$r_j \leq \sum_{l=k}^{(k-j) \bmod f} x_k, \quad k \in F, \quad j \in F \quad (5.11)$$

$$r_j \geq \hat{\rho} \cdot (j - \hat{\Theta}), \quad j \in F. \quad (5.12)$$

Moreover, all optimizations except the first one (4.6) from ILP1 are used for the branch-and-price method as well. The first optimization, the bound on minimal number of allocated slots, does not help much, since we are looking for a schedule of a single client.

5.4 Details of the Branch-And-Price Approach

The branch-and-price method requires specification of many details and each of them may influence computation time a lot. In this section, all the details are explained.

First of all, the important decision is how to choose a variable to branch. Branching on the master model variables is not effective in this case, since they are not decisive enough and they are constantly being added to the column list of the master model. We branch on the slot to be allocated or not to be allocated by the client. Thus, two parameters for the branching decision must be set, which client allocate to which slot. The procedure looks the following way:

1. Choose maximal $\omega_{i,c} \in (0,1)$. The intuition under choosing the column with the highest probability is that it should be a good column, therefore it is a reasonable decision to follow its assignment. Note that it is senseless to pick the column with $\omega_{i,c} = 1$ for the reason that it cannot introduce any new integrality to the solution. Client i' and column c' are chosen.
2. Considering slots from left to right in column c' , i.e. first $j = 1$, then $j = 2$ and so on, check whether:
 - (a) column c' allocates slot j to the client i' ($a_{j,c'}^{i'} = 1$);
 - (b) slot j is not scheduled yet in the branching for any client;

Whenever both conditions hold, slot $j' = j$ is chosen for branching and then the procedure stops.

3. In case we are at the end of the TDM table of column c' and there is no slot to look at anymore, the column with the second largest value of $\omega_{i,c} \in (0,1)$ is considered, then the third one etc. Finally go to 2.

Branching uses a depth-first search considering positive decisions earlier, i.e. branching goes first to the branch where slot j' is allocated to the client i' and only then to the branch where it is not. This branching scheme allows obtaining feasible solutions early, since we go for assignments with high $\omega_{i,c}$.

Furthermore, since an initial solution is allowed to have overlapping columns due to Constraint (5.2), there is no strict requirements on this initial set of columns. In this work, the creation of initial solutions is done by calling the sub-model n times. In the beginning, the shadow prices are set to 0 and a column for the first client is generated by the sub-model. Then, 1 is added to those shadow prices whose slots are allocated by the first column. For instance, if the first obtained column allocates the first slot, 1 is added to the corresponding shadow price. The general rule is given by Equations (5.13) and (5.14):

$$\lambda_j^1 = 0, \quad j \in F \quad (5.13)$$

$$\lambda_j^i = \lambda_j^{i-1} + x_j^i, \quad j \in F, i \in P \quad (5.14)$$

where i indicates the client considered and simultaneously iteration of the initial columns generation. Therefore, for each client, we are trying to allocate slots that were not allocated in the previously generated columns yet. This approach has its drawbacks, but it is relatively fast and sometimes gives feasible solutions even at the beginning.

Thirdly, inside the column generation loop (Steps 3-5, and 9 in Figure 5.2) the sub-model is launched for each client in a Round Robin manner. This means that it first searches for a schedule for the first client, then for the second, etc. until a column for the last client is found and afterwards a new iteration of the column generation starts. In case there are no columns for any client (since all of them are excluded since they do not have the required slots allocated/not allocated to the required clients) due to the branching decisions, new columns are generated by the heuristic for initial column generation that was described above. Whenever the heuristic is not able to generate new columns (even with positive reduced price) for some client in the case described above, the chain of branching decisions cannot offer any feasible solution, which results in closure of this node.

Moreover, for branch-and-bound cutting in nodes more effectively, we use problem-specific information to set bounding condition to $\Phi > U - 1/f$, where the discretization step is subtracted as we want a better solution that can take the maximal value of $U - 1/f$. Note that U is not the same as Φ^* , since the earlier is the best criterion value, obtained inside the model for some frame size and the latter is the best value between all the frame sizes run so far.

Chapter 6

ILP Model with Synthesized Frame Size

So far the ILP model and branch-and-bound method with a fixed frame size are described. However, it is reasonable to think that synthesizing a frame size into the model might be faster than trying all possible frame sizes with ILP1. In this chapter the integer linear programming model ILP2 with variable frame size is formulated and described.

6.1 Model Formulation

The model is based on a cluster representation of schedules. It means that for each client $i \in P$ a sequence of clusters $C_i = \{C_{i,1} \mapsto C_{i,2} \mapsto \dots \mapsto C_{i,\beta_i}\}$ is introduced, where β_i is the maximal number of clusters belonging to client i . For each cluster, the starting time $b_{i,k}$ and processing time $p_{i,k}$, the number of allocated slots in cluster k , both in terms of slots are defined.

The concept of cluster is described below on an example. In Figure 6.1, one can see a TDM table of size 10 with the cluster concept for the first client. Meanwhile in Figure 6.2, the same schedule for two clients is shown with S representation (see Chapter 3). There are two clusters for client one $C_1 = C_{1,1} \mapsto C_{1,2}$ here. Each cluster is defined by its starting time $(b_{1,1}, b_{1,2})$ and by its processing time $(p_{1,1}, p_{1,2})$. Thus, the starting slot of the first cluster $b_{1,1} = 4$ and it has one slot, $p_{1,1} = 1$. For the second cluster $b_{1,2} = 7$ and $p_{1,2} = 4$.

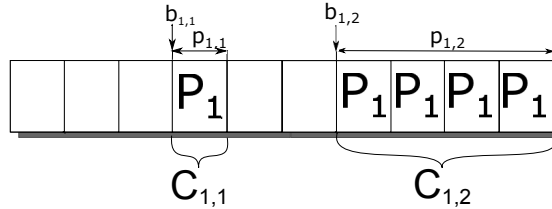


Figure 6.1: TDM with C representation.

$$S = [0001001111]$$

Figure 6.2: TDM with S representation.

With the defined concept of clusters, the model is formalized. The same objective function is considered, i.e. to minimize the total rate $\frac{\sum_{i \in P} \rho_i}{f}$. However, since f is a decision variable, the total rate in this model is formulated as

$$\frac{\sum_{i \in P} \sum_{C_{i,k} \in C_i} p_{i,j}}{f},$$

and it is not a linear function. Therefore a new set of variables needs to be introduced. Variables $z_q \in \{0, 1\}$, $\forall q \in \{0, 1, \dots, \bar{f} - \underline{f}\}$ select the frame size f such that:

$$z_q = \begin{cases} 1, & \text{if } f = \underline{f} + q. \\ 0, & \text{otherwise.} \end{cases}$$

The optimization problem is defined as follows:

$$\text{minimize } \Phi, \tag{6.1}$$

where Φ is defined by Constraints (6.2)-(6.17). The following constraints are satisfied:

Equation (6.2) defines the size of the TDM table. Due to Constraint (6.3), only one frame size is possible.

$$f = \sum_{q \in \{0, 1, \dots, \bar{f} - \underline{f}\}} z_q \cdot (\underline{f} + q), \tag{6.2}$$

$$\sum_{q \in \{0, 1, \dots, \bar{f} - \underline{f}\}} z_q = 1. \tag{6.3}$$

The set of Constraints (6.4) is a "linearization" of our criterion, where M is a sufficiently large number. The first addend is simply the criterion of ILP1 considering the frame size of $f = \underline{f} + i$. The second addend, $(1 - z_i) \cdot M$, makes constraints for all the frame sizes, but the one it is, invalid by pushing the value on the right side to a very large number. Thus, there are $\bar{f} - \underline{f} + 1$ Constraints (6.4) and only one of them is valid, as it is given by Constraint (6.3).

$$\Phi \geq \frac{\sum_{j \in P, k \in C} p_{j,k}}{\underline{f} + i} - (1 - z_i) \cdot M, \quad \forall i = \{0, 1, \dots, q\} \tag{6.4}$$

Note that ILP2 is based on the three main sets of variables: b_i, p_i and $\sigma_{i,k}^{j,l}$. Variables b_i and p_i were defined in the beginning of this chapter and $\sigma_{i,k}^{j,l}$ defines the order of clusters of different clients:

$$\sigma_{i,k}^{j,l} = \begin{cases} 1, & \text{if cluster } C_{i,k} \text{ is scheduled before cluster } C_{j,l} \\ 0, & \text{otherwise,} \end{cases}$$

$$i, j \in P : i \neq j; C_{i,l} \in C_i; C_{j,k} \in C_j$$

Next, Constraint (6.5) interconnects start time, the number of allocated slots and the frame size

$$b_{i,k} + p_{i,k} \leq f, \quad i \in P, C_{i,k} \in C_i. \quad (6.5)$$

Practically, Constraint (6.5) means that a cluster cannot go beyond the end of a TDM table.

Furthermore, another set of binary variables $y_{i,k} \in \{0, 1\}, i \in P, C_{i,k} \in C_i$ is introduced, indicating whether $p_{i,k} = 0$, or in other words whether it is a *dummy cluster*. This notion is introduced since it is not possible to predict an exact number of clusters, thus some clusters are unused in the solution, i.e. they have $p_{i,k} = 0$.

$$y_{i,k} \leq p_{i,k} \leq y_{i,k} \cdot M, \quad i \in P, C_{i,k} \in C_i. \quad (6.6)$$

Thus, in case $p_{i,k}$ is equal to zero, $y_{i,k}$ must also be 0 and 1 otherwise.

Moreover, within the same client, clusters are required to be scheduled one after each other. In case of dummy clusters, it is necessary that the starting time of the next cluster is equal to the completion time of the previous one since their processing time is zero. On the other hand, in case of a real cluster, free space between two real clusters must be at least one slot by definition of a cluster. Two clusters without space is one larger cluster.

$$b_{i,k} + p_{i,k} + y_{i,k+1} \leq b_{i,k+1}, \quad i \in P, C_{i,k} \in C_i. \quad (6.7)$$

Constraint (6.8) pushes start time of dummy clusters to the completion time of the previous cluster.

$$b_{i,k+1} \leq b_{i,k} + p_{i,k} + M \cdot y_{i,k+1}, \quad i \in P, C_{i,k} \in C_i. \quad (6.8)$$

Finally, clusters of different clients are required to be scheduled in a non-overlapping manner as well. Thus, each one must be scheduled before or after another, not simultaneously. Variables $\sigma_{i,k}^{j,l}$ help to define the order of scheduling. In the case cluster $C_{j,l}$ goes before cluster $C_{i,k}$ in the TDM table, Constraint (6.9) holds, otherwise Constraints (6.10). Constraints (6.9), (6.10) check for each pair of clusters which one is first.

$$b_{i,k} + p_{i,k} \leq b_{j,l} + M \cdot \sigma_{i,k}^{j,l}, \quad i \in P, j \in P : i \neq j; C_{i,k} \in C_i, C_{j,l} \in C_j. \quad (6.9)$$

$$b_{j,l} + p_{j,l} \leq b_{i,k} + M \cdot (1 - \sigma_{i,k}^{j,l}), \quad i \in P, j \in P : i \neq j; C_{i,k} \in C_i, C_{j,l} \in C_j. \quad (6.10)$$

The bandwidth requirements are handled in the set of Constraints (6.11).

$$\sum_{C_{i,k} \in C_i} p_{i,k} \geq \hat{p}_i \cdot f, \quad i \in P. \quad (6.11)$$

The service latency is computed according to the subtables algorithm [5]. We compute the total number of slots $f_{i,j}$, allocated and non-allocated, in each subtable and there are two cases: whether it is the first cluster for a client (6.12) or not (6.13). In case it is the first subtable in our schedule, it is necessary to compute its $f_{i,j}$ looking at the end of the schedule, meantime other cases are treated by looking to the previous cluster of the same client.

$$f_{i,1} = f + b_{i,1} + p_{i,1} - (b_{i,\beta_i} + p_{i,\beta_i}), \quad i \in P \quad (6.12)$$

$$f_{i,k} = b_{i,k} + p_{i,k} - (b_{i,k-1} + p_{i,k-1}), \quad i \in P, k = \{2, 3, \dots, n\} \quad (6.13)$$

The local latencies and offsets of the subtables, introduced in Equations (2.3) and (2.4), correspondingly, are computed in Constraints (6.14) and (6.15):

$$L_{i,k} = 1 + f_{i,k} - p_{i,k} - \hat{\rho}_i^{-1}, \quad i \in P, k \in C \quad (6.14)$$

$$\Delta_{i,k} = f_{i,k} - p_{i,k} \cdot \hat{\rho}_i^{-1}, \quad i = 1, \dots, n, k \in C \quad (6.15)$$

Finally, the service latency can be computed according to Equation (6.16). Each equation for a client i considers a situation, where the start subtable is r and the end subtable is $r+j$. Note that for each client there are $r \cdot j$ Constraints (6.16), which provides us with the maximum value over all starting and ending points.

$$\Theta_i \geq \sum_{t=r}^{r+j-1} (\Delta_{i,t \bmod \beta_i}) + L_{i,(r+j) \bmod \beta_i} + (\hat{\rho}_i^{-1} - 1), \quad i \in P, j \in C, r \in C \quad (6.16)$$

The service latency requirements must be satisfied by Constraint (6.17):

$$\Theta_i \leq \hat{\Theta}_i, \quad i \in P. \quad (6.17)$$

6.2 Optimizations of Computation Time

Similarly to the previous model, the optimizations are crucial to improve the computation time. One of the most important issues in this model is to effectively *set the maximum possible number of clusters* β_i for each client. The naive solution could be to use $\beta_i = \bar{f}/2$ for each $i \in P$, since there should be at least one empty slot between two non-dummy clusters, which leaves us $\bar{f}/2$ possible clusters in the worst case. However, this value could be improved by considering the minimum possible number of allocated slots for each client $\check{\phi}_i, i \in P$:

$$\beta_i = \min(\bar{f}/2, \bar{f} - \sum_{j \in P} \check{\phi}_j + \check{\phi}_i) \quad (6.18)$$

Firstly, the maximum number of slots, potentially free to be allocated to a client is computed by adding the maximum number of potentially free slots to the minimum number of allocated slots for this client. In case this number is greater than $\bar{f}/2$ it cannot happen, so the maximum possible number of clusters is reduced to $\bar{f}/2$. It is a valid upper bound since by summation of minimal number of slots that a particular client requires and the maximum number of potentially non-allocated slots in the table, we compute the maximal possible number of slots, available for the client. There are no other possible slots for allocation of the client. Lastly, each slot is assumed to become a cluster of its own and thus the number of clusters is bounded.

The second optimization is assigning *higher priority* in the branching scheme in the solver to more important variable f . Intuitively, in comparison with other variables this one has the

highest impact on the computation time of the solution - changing number of slots makes the solver to start from the scratch in comparison with, for instance, changing a starting time of a cluster.

Since symmetry is an even more significant problem in this model, a simple constraint to prevent symmetry in dummy clusters allocation is introduced by the set of Constraints (6.19). There is not only rotational symmetry here, but also every permutation of the dummy clusters for a given schedule results in an equivalent solution. Recall that variables $y_{i,k}$ identify whether or not a cluster is dummy, thus the set of Constraint (6.19) *pushes all of the dummy clusters to follow the non-dummy ones*, reducing the symmetry.

$$y_{i,k} \geq y_{i,k+1}, \quad i \in P, \quad C_{i,k} \in C_i \quad (6.19)$$

Similarly to ILP1, Constraints (6.20) and (6.21) that bound rotational symmetry in slot allocation are used. Generality is not lost here since an arbitrary number of slots for any client can be allocated in the end of the TDM table as it is not required for the last cluster to finish not later than in the slot $f - 1$ unlike other non-dummy clusters, that have to be allocated at least 1 slot ahead. Similarly to ILP1, w is the client with the minimal required slots to allocate, i.e. the same as in Equation (4.7).

$$b_{w,1} = 1; \quad (6.20)$$

$$p_{w,1} = 1; \quad (6.21)$$

As it was done in ILP1, *non-promising frame sizes are discarded* by Constraint (6.22). This time decision variable f is present in both sides of the constraint. If the frame size is less than the sum of minimum possible required slots over all the clients, then it is not possible to find any feasible solution with this frame size.

$$f \geq \sum_{i \in P} \check{\phi}_i \quad (6.22)$$

Chapter 7

Heuristic Frame Size Filtering

The considered NP-hard TCP/LR problem was formalized earlier along with two optimal ILP models and the branch-and-price method. Although a significant number of computation time optimizations were introduced for each of the methods, ILP1 and B&P still require quite long time to iterate over all the frame sizes, when more clients with larger frame sizes are considered. In order to deal with this problem, two heuristic approaches are presented below.

The naive approach to find the optimal frame size is to iterate over some range. A more refined, however not optimal, approach is to choose promising frame sizes and to run the solver only for them. The main difference between the presented heuristics is how many and which of the frame sizes are explored.

7.1 K-heuristic

In Algorithm 1, one can see the first heuristic, called the K-heuristic. The algorithm requires two inputs, the set of frame sizes, F , to be explored and an integer number K that defines number of frame sizes out of $|F|$ candidates to be tested with the solver on a model. The output is an ordered set F' of K selected frame sizes. The goodness, g , of all the frame sizes in F is evaluated first. As it can be seen from Line 3 of the algorithm, the notion of goodness is defined as a measure of quality of a given frame size in terms of discretization. Frankly speaking, goodness for a given frame size is computed as an overallocation due to bandwidth requirements. Similarly to the computation time optimization in the Chapter 4, it computes the total amount of overallocation due to discretization. The next step sorts the set F in ascending order based on the computed goodness, g_i , on Line 5 and returns the ordered set F' , which are the K first elements of the sorted set F .

As it was already said, the K-heuristic does not necessarily provide us with the optimal solution. The reason is the latency-dominated instances behave unpredictably as they may need additional slots to satisfy their latency requirements. Thus, it may happen that the K-heuristic discards optimal frame size(s) to reduce computation time. However, as shown in the experiments Chapter 8, the loss in terms of criterion value is less than a percent and the computation time is 85% of the optimal iterating approach on average.

```

1: Inputs:  $F; K \in \mathbb{N}, K \leq |F|$ 
2: for all  $f_i \in F$  do
3:    $g_i = \sum_{j \in P} [f \cdot \hat{\rho}_j] / f \cdot \hat{\rho}_j$ 
4: end for
5: sort  $F$  ascending based on  $g_i$ 
6:  $F' = F[1 : K]$ 
7: Output:  $F'$ 

```

Algorithm 1: K-heuristic

7.2 KL-heuristic

The second heuristic aims to reduce the computation time of K-heuristic even more by preferring small frame sizes with potentially smaller solving time. Algorithm 2 presents the way the KL-heuristic works. Just like the K-heuristic, the KL-heuristic takes as an input set of frame sizes F to explore and an integer number K . The difference lies in the L parameter that plays an important role. Firstly, on Line 2 of the algorithm the K-heuristic is used to obtain the ordered set F' of frame sizes with the best discretization. Afterwards the set F' is sorted in an ascending order of frame sizes. It continues on Line 4 by taking L first elements of the set F' and sending it as an output. This behavior results in that smaller numbers, providing possibly slightly worse discretization, are preferred over larger numbers with typically higher computation time. As a matter of fact, computation time grows in an exponential manner with growing frame sizes. Frequently, preferring smaller frame sizes results in increasing criterion value, although because of its lower computation time it is possible to choose how fast the solution should be versus how much one can sacrifice in terms of criterion value.

```

1: Inputs:  $F; K, L \in \mathbb{N}, L \leq K \leq |F|$ 
2:  $F' = \text{K-heuristic}(F, K)$ 
3: sort  $F'$  ascending based on  $f$ 
4:  $F'' = F'[1 : L]$ 
5: Output:  $F''$ 

```

Algorithm 2: KL-heuristic

However, in spite of a substantial speed up of the heuristics due to reduced number of considered frame sizes, it is not theoretically guaranteed to always solve faster than the optimal approach with iterating. Filtering of frame sizes causes it by not considering some frame sizes that can be useful in terms of criterion value propagation, introduced as an optimization for ILP1 and B&P. Overlooking promising candidates may require larger frame sizes to be evaluated without a good cutting point, $\bar{\Phi}$, to bound the solution space. Nonetheless experimental results show significant reduction of the computation time with the described heuristics.

Chapter 8

Experimental Results

This section experimentally evaluates the proposed models ILP1, B&P and ILP2 with heuristics for solving the considered TCP/LR problem. It begins with the experimental setup, followed by three experiments.

8.1 Experimental Setup

Two sets of 500 synthetic use-cases, one with bandwidth-dominated use-cases and one with latency-dominated use-cases, are used for experiments. The reason to provide the experiments for two completely different types of use-cases is to show the impact of the requirements on the computation time of the algorithm and at the same time evaluate the quality of the heuristics, which are focused more on bandwidth-dominated use-cases since they are based on the quality of the discretization. The process of generation of both latency- and bandwidth- dominated use-cases is described below.

In order run many use-cases in a reasonable time, the number of clients is set to 4. Therefore, it remains to generate bandwidth and latency requirements for both sets of use-cases. Bandwidth requirements are generated such that the total required allocated rate is distributed uniformly in $[0.8, 0.95]$ for the bandwidth-dominated set of use-cases and $[0.35, 0.5]$ for the latency-dominated set. Particular bandwidth requirements for each client are distributed uniformly between 5% and 40%, where in case the last client has less than 5% of the total load, the use-case is discarded and a new one is generated.

As far as the service latency requirements are concerned, they are generated as $\frac{1}{\alpha \cdot \rho}$, where for the bandwidth-dominated set $\alpha \in [0.7, 1.0]$ and latency-dominated use-cases have $\alpha \in [1.5, 3.5]$. Moreover, if the minimum possible total load for latency-dominated use-cases due to the latency requirements is outside the interval $[0.7, 0.9]$, computed with maximal frame size ($f = 64$), the latency requirements for this use-case are discarded and new ones are generated.

Experiments were performed on a computer equipped with AMD Phenom II X4 945 processor (3.0 GHz, 4 cores) and 8 GB memory. The ILP1 and ILP2 models were implemented in IBM ILOG CPLEX Optimization Studio 12.6 and solved with the CPLEX solver. The B&P model was implemented in Java 7.3.1 using CPLEX concert technology.

8.2 Experiments

Three experiments are presented in this section. The first experiment evaluates the trade-off between the criterion value and the computation time for ILP1 with and without the two heuristic frame-filtering methods. This experiment shows how much computation time is saved versus how much the criterion value increases with the heuristics. The second one shows the behavior of the K- and KL- heuristics for different K and L values in order to provide us with an intuition which values of heuristics parameters are reasonable. In the last experiment, the results of the heuristics and ILP2, bounded in time, are compared to demonstrate which approach is better. Bounded time is achieved by setting a time limit in the solver after which it presents the best solution found so far.

8.2.1 Comparison of optimal solution and heuristics

The first experiment evaluates the trade-off between the computation time and the criterion value of the optimal ILP1 model and the two heuristic filtering algorithms. ILP1 initially finds the optimal frame size by iterating over all sizes in the range $f = \{\underline{f} = 1, \dots, \bar{f} = 64\}$. This range is reasonable in terms of computation time and has a reasonable cost in terms of hardware implementation. The K-heuristic considers only the best frame size ($K = 1$) and in the KL-heuristic the settings ($K = 10$, $L = 5$) are used. Other values of K and L are evaluated in the following experiments. Moreover, the results of these experiments are compared with the commonly used continuous allocation strategy [2], [3], [4].

The results of experiments on 500 bandwidth-dominated use-cases are shown in Figure 8.1. The criterion value (blue) and the computation time (red) of the experiments are presented there. Because the computation time significantly differs for distinct use-cases, the logarithm of the computation time in milliseconds is shown on the right y-axis. ILP1 iterating over all frame sizes and with the K-heuristic and KL-heuristic were run on the same set of use-cases. All three methods successfully solved all 500 use-cases. As expected, the figure shows that the optimal approach provides the lowest criterion values, it runs approximately 2 hours though for all use-cases. For the heuristic approaches, we see that the K-heuristic requires less than 34% (around 39 minutes) of the computation time of the optimal approach and sacrifices just 0.5% of the allocated rate. The KL-heuristic shows the following results: the median of the criterion value is roughly 3% worse compared to the optimal approach, while the results are computed in 1.5% (2 minutes) of the total time.

Figure 8.2 shows the results for latency-dominated use-cases. Compared to the bandwidth-dominated set, the computation time is generally higher in this experiment, requiring approximately 20 hours for 500 use-cases. A possible explanation of this fact could be that latency-dominated use-cases are harder to solve, since if the client has higher bandwidth requirements with relaxed latency requirements, it is quite simple to find a good schedule, so the nodes in the branch-and-bound algorithm would be cut very efficiently later. In case the requirements are dominated by service latency requirements, collisions appear repeatedly, which causes necessity to over-allocate slots for some clients, generally resulting in a more difficult decision process. The K-heuristic shows a similar distribution of the computation time as the optimal approach: it loses less than 1.5% of the criterion value, while using approximately 3% (37 minutes) of the computation time. However, in 1 use-case out of 500 the

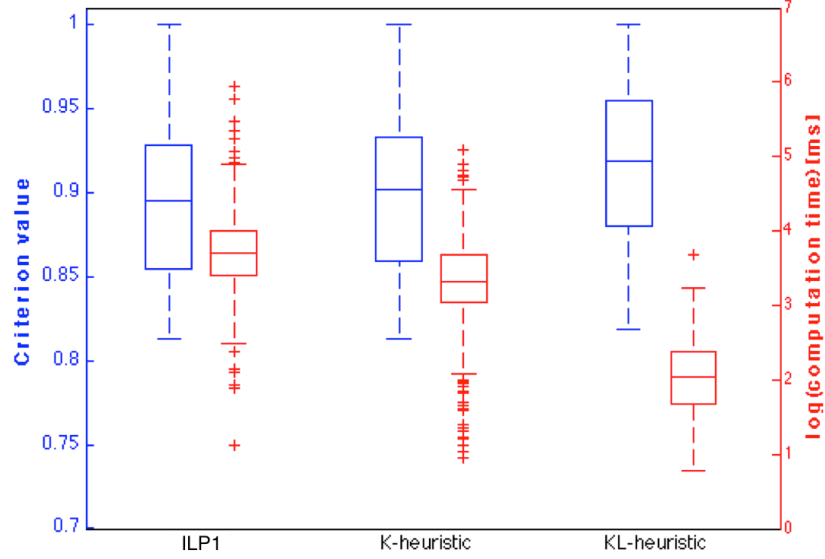


Figure 8.1: Criterion value / computation time trade-off for the bandwidth-dominated use-cases. ILP1 with and without heuristics.

K-heuristic was not able to find a feasible solution. The KL-heuristic results in a 3% increase of the criterion value of the optimal approach and it uses only 0.04% of its computation time (36 seconds). In 6/500 use-cases KL-heuristic failed to find a feasible solution.

Although it is possible to use B&P for this experiment, it requires a lot of improvement before it starts to have a reasonable computation time. The branch-and-price method was tested on the first 5 latency-dominated and first 5 bandwidth dominated use-cases from this experiments. It provides the optimal solution (the same as ILP1 gives), but it runs significantly longer than ILP1. For bandwidth-dominated use-cases it took more than 2 days to compute it (versus less than a minute with ILP1). For 5 latency-dominated use-cases even going to frame size f of 64 was extremely long, thus it was bounded to the frame size f of 50 and it is more than 3 days versus approximately 40 seconds for ILP1. From the computation time analysis that were done on B&P follows that a possible reason is a huge amount of nodes in the branch-and-bound algorithm. Hopefully, processing one node for a longer time with other optimization methods, such as a tabu search, coupled with the sub-model and solving in the upper nodes not to the optimum, but close to it, will help to significantly reduce number of nodes in branching tree. Another possible optimization is to use another branching scheme, i.e. to pick slot and client to branch differently.

From these experiments, we draw the following conclusions: ILP1 finds optimal solutions by iterating over all the frame sizes, but it takes a relatively long time to do so. In contrast, the K-heuristics and KL-heuristics provide us with near-optimal results in only a fraction of the time required to find an optimal solution. The continuous allocation algorithm, in comparison, only found a solution in 36/500 use-cases for the bandwidth-dominated set and

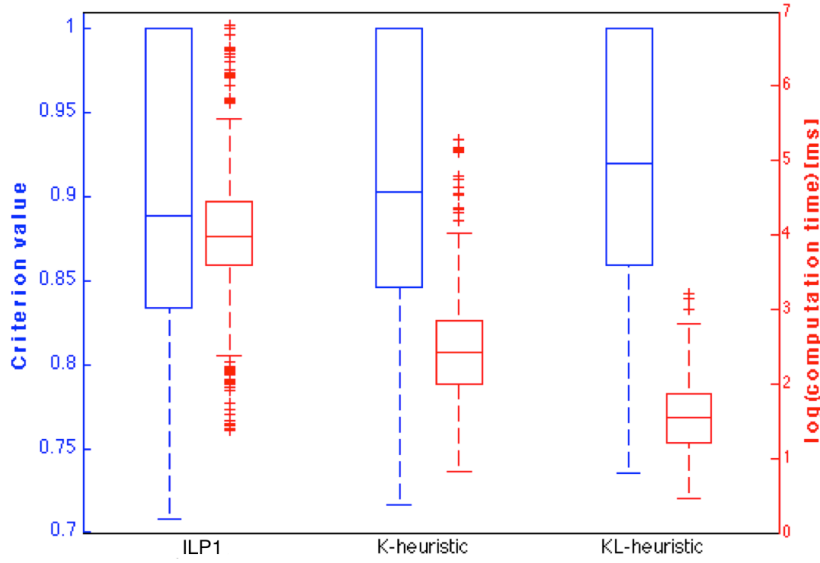


Figure 8.2: Criterion value / computation time trade-off for the latency-dominated use-cases. ILP1 with and without heuristics.

149/500 for the latency-dominated set. Moreover, out of 185 cases where it found solutions, 24/185 instances provide the same allocated rate (equal to 1) as our heuristics and all other are worse. This suggests that the usefulness of the continuous allocation strategy is limited.

8.2.2 Evaluation of K and L values

While in the first experiment fixed values of K and L were used, it is valuable to see the difference of particular K and L settings in terms of computation time and criterion value. For this purpose, the 1000 use-cases from the previous experiments were rerun with different K and L values and the average criterion value and computation time are compared.

Figure 8.3 shows the results for the K-heuristic. One can see that the computation time grows linearly with the K value, which is expected due to the nature of the K-heuristic. It often chooses larger frame sizes in order to get better granularity, resulting in approximately the same computation time for each new value of K . The criterion value shows a convergence towards the optimal value (red dotted line), which is guaranteed to finish by $K = \bar{f}$. It is noticeable that smaller values of K provide good results for the small number of clients. Note, however, that larger number of clients can increase the effect of discretization, making it worthwhile to consider larger values of K for more clients.

The results of the KL-heuristic are presented in Figure 8.4 with K value fixed to 10 and L value going from 1 to 10. Unlike the K-heuristic, the computation time seems to grow exponentially with L value, which can be intuitively explained as the KL-heuristic chooses L smallest frame sizes out of the K best ones and run them in increasing order. The next

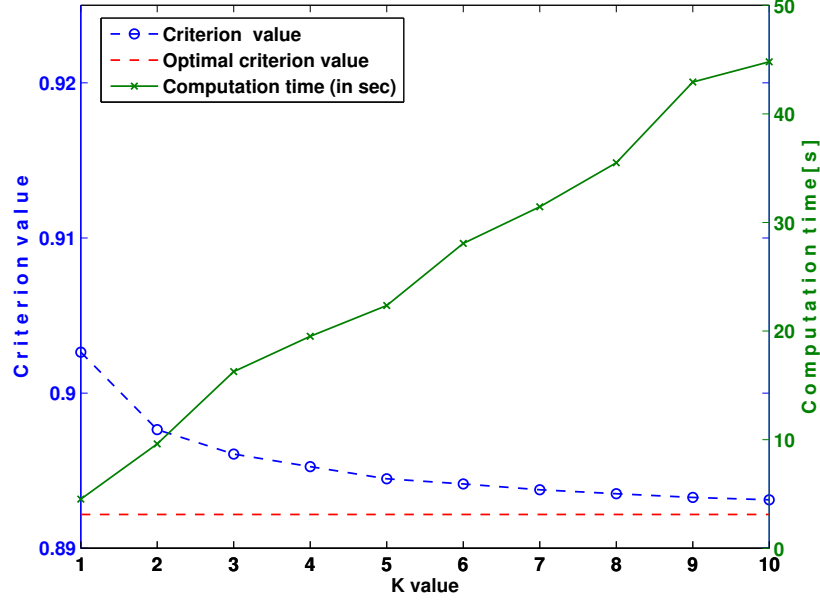


Figure 8.3: Average criterion value and computation time of K-heuristic for different values of K .

noticeable detail in the figure is that the criterion value with growing L approaches the optimal criterion value at a lower pace than in the case of the K-heuristic and convergence theoretically is only guaranteed for $K = L = \bar{f}$. The key advantage of the KL-heuristic that it gives the solution with reasonable loss in the criterion value faster than the K-heuristic. For instance, the KL-heuristic with $K = 10$ and $L = 8$ provides a better solution than the K-heuristic with $K = 3$, while both require approximately 10 seconds on average per instance.

8.2.3 Comparison of heuristics and ILP2

Although ILP2 is an optimal approach, it takes significantly longer time to provide an optimal solution than the optimal approach with ILP1 (hundreds to thousands times longer). For this reason, we bounded time of each use-case to be less than 100 seconds. This value is obtained as approximately the maximum (not considering outliers) value for a single instance to run with the heuristics. The ILP2 is compared to ILP1 with the filtering heuristics since there is a chance that it finds a reasonable solution quickly but takes long to find an optimal one.

Figure 8.5 shows the results of experiments on 500 bandwidth-dominated use-cases that were used in the previous experiments. Basically, the results of the K- and KL- heuristics are the same as in Figure 8.1, they are just shown to enable comparison with ILP2 in the same figure. In 13/500 use-cases, ILP2 was not able to find any feasible solution during 100 seconds. The results for unsolved use-cases are not included to the statistics. Figure 8.5

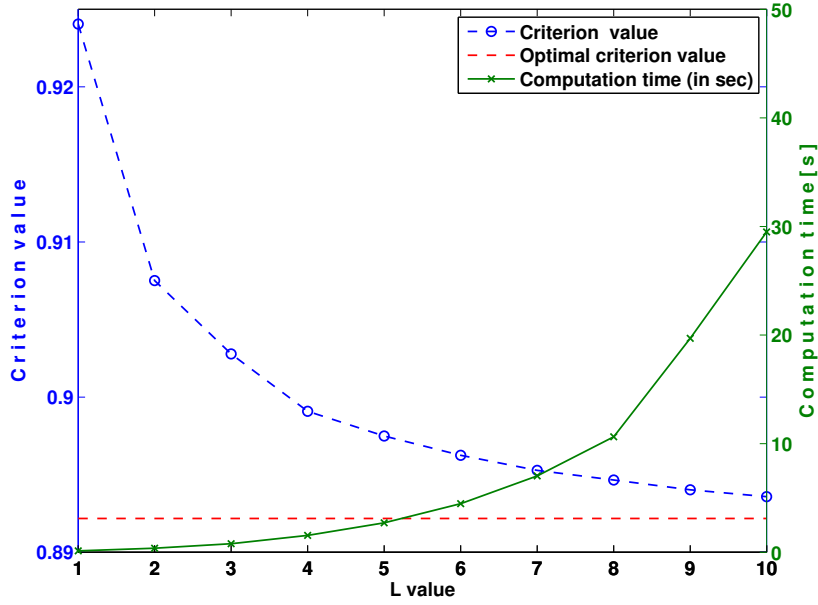


Figure 8.4: Average criterion value and computation time of KL-heuristic for different values of L with $K = 10$.

shows that ILP2 provides approximately the same solution (the median value for both is 0.92) as K-heuristic in terms of the criterion value, however, in a significantly longer time, 97 seconds versus 4 seconds on average for one use-case.

The results for the 500 latency-dominated use-cases can be seen in Figure 8.6. A feasible solution was not found in 83/500 use-cases during 100 seconds. For this set of use-cases, it is obvious that ILP2 does not provide lower criterion values even with more computation time.

In conclusion of the experimental evaluation, note that the scalability of the presented approaches is an important issue. The provided experiments only consider smaller problems with 4 clients. The reason of choosing this size of instances is that just the first experiments require a day. It is important to use many instances for the evaluation, since there are large differences between different instances. However, it is not a scalability problem since one instance takes less than 83 seconds on average, which is not particularly long for a product under development.

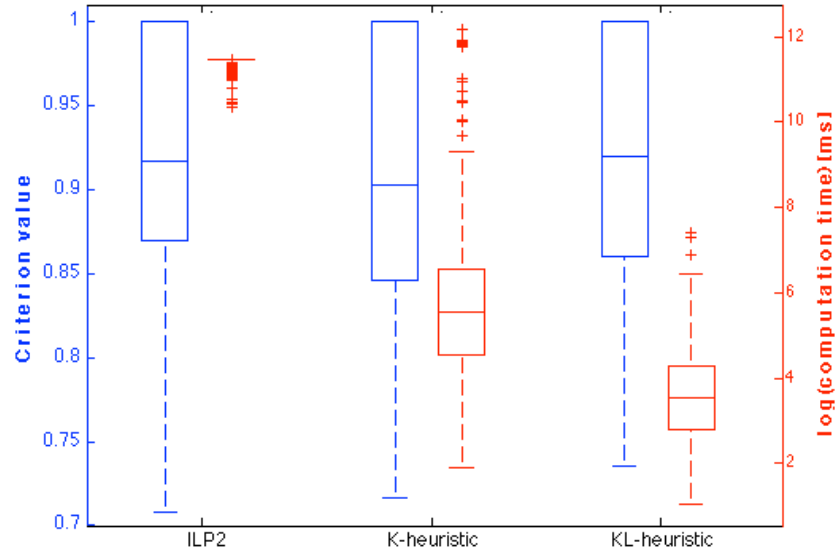


Figure 8.5: Criterion value / computation time trade-off for the bandwidth-dominated use-cases. ILP1 using heuristic frame filtering and ILP2.

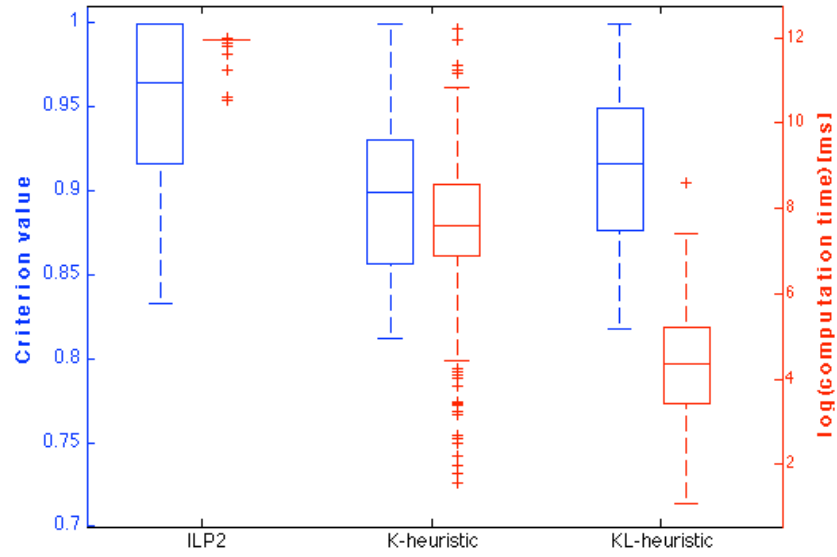


Figure 8.6: Criterion value / computation time trade-off for the latency-dominated use-cases. ILP1 using heuristic frame filtering and ILP2.

Chapter 9

Related Work

A lot of works, related to the TCP/LR problem have been published in the real-time, system-on-chip and scheduling communities. We continue by discussing these works with a focus on non-CPU resources, which is the main focus of this thesis.

Firstly, works from the real-time and system-on-chip communities are reviewed. In [16] authors propose a TDM configuration methodology that heuristically determines and optimizes bus schedules in order to satisfy the response time requirements of task graphs. Unlike the approach proposed in this thesis, the authors consider coarse-grained schedules, i.e. those that work with slots of fixed or variable duration that is typically much longer than a single resource access. Meanwhile, here a slot is just a single resource access and it is called fine-grained arbitration. The main disadvantage of the coarse-grained arbitration is that by prolonging the slot in terms of clock cycles, the service latency increases, resulting in impossibility to create a feasible schedule for some requirements. The second main difference between [16] and the proposed approach is that they rely on application information about when requests are delivered, while we provide \mathcal{LR} guarantee that is independent of the application model.

Other methodologies for configuration of TDM arbiters are proposed in [18, 19, 20]. These works deal with slot allocation in contention-free TDM networks-on-chips. They find a global TDM schedule for all clients through a network such that each client always has consecutive TDM slots along the path from source to destination and the schedule satisfies latency and bandwidth requirements. The key difference is that scheduling along multiple resources is involved. Besides, it depends on the problem of path finding in the network. Additionally, similarly to [16] all these approaches are heuristic and the results are not compared to the corresponding results of optimal approaches.

An optimal configuration technique for multi-channel memory controllers that uses ILP is presented in [2]. It contains TDM slot allocation to satisfy requirements, but the proposed approach is limited to continuous slot allocation and assumes a given frame size. However, it was experimentally proven in Chapter 8 that the continuous allocation strategy in the best case over-allocates and more often it is not able to provide any feasible schedule.

It was already mentioned in the proof of the complexity theorem in Chapter 3 that the two most related problems from the scheduling community are the periodic maintenance scheduling problem (PMSP) [10] and the pinwheel problem [8]. PMSP is just a special

case of TCP/LR, where the bandwidth requirements, $\hat{\rho}_i$, are in the following relation with service latency: $\hat{\Theta}_i$, requirements: $\hat{\rho}_i = \frac{1}{\hat{\Theta}_i + 1}$. Moreover, equidistant schedules are always required, $\Phi \leq \sum_{i \in P} \hat{\rho}_i$. Generally, it is not possible to use solutions of PMSP problem for TCP/LR, since it is not clear how to transform bandwidth and latency requirements for TCP/LR to service intervals l in PMSP, i.e. because PMSP is a special case of TCP/LR, not the other way around.

As far as the pinwheel scheduling problem is concerned, although it does not require a perfectly equidistant schedule, it has a different notion of latency requirements. The pinwheel scheduling problem expects each client to be allocated at least once for some amount of time steps. Satisfying the pinwheel conditions do hence not necessarily mean satisfying our bandwidth and latency requirements.

Chapter 10

Conclusions and Future Work

This master thesis presents three approaches to configure resources shared by Time-Division Multiplexing (TDM) arbitration. A guarantee on bandwidth and latency requirements of real-time clients is provided and at the same time total allocated rate of a TDM table is minimized to allow non-real-time clients to have as high performance as possible. The considered problem requires both the size of the TDM table and a slot allocation to clients to be determined. The problem is formalized and shown to be NP-hard. Furthermore, three optimal models, based on integer linear programming (ILP) are presented to solve it. Two of them (ILP1 and B&P) consider the frame size to be given, resulting in iteration over frame sizes in some interval to find the best one, while the other (ILP2) synthesizes the frame size. However, requiring iterations, the presented approaches take long time to solve the problem optimally for large problem instances. We address this problem by presenting two frame-filtering heuristic algorithms, the K-heuristic and the KL-heuristic, to find promising candidate frame sizes that require some sacrifice in total allocated rate, but reduce computation time.

The experiments show that ILP1 significantly outperforms the commonly used continuous allocation strategy in terms of utilization and the heuristics provide reduction of computation time of 20% on average, sacrificing less than 3% of the criterion value. Moreover, comparison between ILP1 and ILP2 demonstrates a clear advantage of the earlier in terms of computation time. As far as the heuristics are concerned, the K-heuristic typically results in a smaller number of candidate frame sizes to perform well, although larger ones to achieve a good discretization, while the KL-heuristic needs a larger number of explored smaller frame sizes. As a result the K-heuristic provides a better criterion value in a longer time than the KL-heuristic, thus providing distinct trade-offs.

Future work involves optimization of the B&P approach. Experiments show that the number of nodes in the search tree is very large in comparison with other already refined known versions of B&P method for other problems. Hence, the future possible optimizations focus on reducing the number of nodes by adding heuristic algorithm(s) inside each one. The other possibility is to generate more than one good column during each iteration of column generation algorithm, since we lose a lot of time to find the only column. Combining using a heuristic and generating more than one column might significantly reduce the number of nodes in the branching tree. The third possible improvement is to change the branching

scheme. Good choice of a branching scheme is important and influences computation time a lot.

Bibliography

- [1] A. Varma and D. Stiliadis. Latency-rate servers: a general model for analysis of traffic scheduling algorithms. *IEEE/ACM Transactions on Networking (TON)*, Volume 6 Issue 5, 1998, pp. 611 - 624
- [2] M. D. Gomony, B. Akesson and K. Goossens. Architecture and Optimal Configuration of a Real-Time Multi-Channel Memory Controller. *Proc. DATE*, 2013, pp. 1307-1312.
- [3] S. Foroutan, B. Akesson, K. Goossens and F. Petrot. A Reconfigurable Real-Time SDRAM Controller for Mixed Time-Criticality Systems. *CODES*, 2013, pp. 1-10.
- [4] S. Goossens, J. Kuijsten, B. Akesson and K. Goossens. A General Framework for Average-Case Performance Analysis of Shared Resources. *Digital System Design (DSD)*, 2013, pp. 78-85.
- [5] A. Nelson. Conservative Application-Level Performance Analysis through Simulation of a Multiprocessor System on Chip. Master's thesis. Eindhoven University of Technology, 2009.
- [6] H. Shah, A. Knoll and B. Akesson. Bounding SDRAM interference: detailed analysis vs. latency-rate analysis. *Proceedings of the Conference on Design, Automation and Test in Europe*, 2013, pp. 308-313.
- [7] B. Akesson, A. Molnos, A. Hansson, J. A. Angelo and K. Goossens. Composability and predictability for independent application development, verification, and execution. *Multiprocessor System-on-Chip*. Springer New York, 2011, pp. 25-56.
- [8] R. Holte, L. Rosier, I. Tulchinsky and D. Varvel. The Pinwheel: A realtime scheduling problem. *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences*, 1989, pp. 693-702.
- [9] O. Kone, C. Artigues, P. Lopez and M. Mongeau. Event-based MILP models for resource-constrained project scheduling problems. *Computers and Operations Research*, Volume 38, Issue 1, January 2011, pp. 3-13.
- [10] A. Bar-noy, R. Bhatia, J. Naor and B. Schieber. Minimizing Service and Operation Costs of Periodic Scheduling. *tech. rep.*, Tel Aviv University, Volume 95, 1997, p.11.
- [11] B. Akesson and K. Goossens. *Memory Controllers for Real-Time Embedded Systems*. Springer, 978-1-4419-6496-0, First edition, 2011.

- [12] S. Martello and P. Toth. Knapsack Problems: Algorithms and Computer Implementations. John Wiley&Sons, Inc, 1990.
- [13] D. Feillet. A tutorial on column generation and branch-and-price for vehicle routing problems. 4OR, 2010, Page 1.
- [14] R. J. Vanderbei. Linear Programming: Foundations and Extensions. 1996.
- [15] B. Akesson, A. Minaeva, P. Šůcha, A. Nelson and Z. Hanzálek. Efficient Configuration Methodologies for Time-Division Multiplexed Resources. 2014. Submitted paper to RTSS.
- [16] A. Andrei, P. Eles, P. Zebo, J. Rosen. Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip. VLSI Design, 2008, pp. 103-110.
- [17] "Branch and price." Wikipedia: The Free Encyclopedia. Wikimedia Foundation, Inc. 22 July 2004. Web. 10 Aug. 2004.
- [18] A. Hansson, K. Goossens and A. Radulescu. A unified approach to mapping and routing on a network-on-chip for both best-effort and guaranteed service traffic. VLSI Design, 2007.
- [19] Z. Lu and A. Jantsch. Slot allocation using logical networks for TDM virtual-circuit configuration for network-on-chip. ICCAD, 2007, pp. 18-25.
- [20] R. Stefan and K. Goossens A TDM slot allocation flow based on multipath routing in NoCs. Microprocessors and Microsystems - Embedded Hardware Design. 2011, pp. 130-138.