Workload assignment for global real-time scheduling on unrelated multicore platforms

Antoine Bertout
antoine.bertout@univ-poitiers.fr
LIAS, Université de Poitiers, ISAE-ENSMA
Poitiers, France

Joël Goossens
joel.goossens@ulb.ac.be
Université libre de Bruxelles
Brussels, Belgium

Emmanuel Grolleau
grolleau@ensma.fr
LIAS, ISAE-ENSMA, Université de Poitiers
Chasseneuil-Futuroscope, France

Xavier Poczekajlo
xavier.poczekajlo@ulb.ac.be
Université libre de Bruxelles
Brussels, Belgium

ABSTRACT

Heterogeneous MPSoCs are being used more and more, from cellphones to critical embedded systems. Most of those systems offer heterogeneous sets of identical cores. In this paper, we propose new results on the global scheduling approach. We extend fundamental global scheduling results on *unrelated* processors to results on unrelated multicore platforms, a more realistic model. Every discussed result is optimal regarding schedulability, and all but one have a polynomial time complexity. We introduce several methods to construct the workload assignment taking advantage of this new model. Thanks to the model, their produced schedule has a limited degree of migrations. The benefits of those methods are demonstrated using simulation. We also discuss the practical limitations of the global scheduling approach on unrelated platforms and argue that it is still worth investigating considering modern MPSoCs.

CCS CONCEPTS

• **Computer systems organization → Real-time systems**; *Multicore architectures.*

KEYWORDS

real-time scheduling, global scheduling, multiprocessor, heterogeneous platform

ACM Reference Format:
Antoine Bertout, Joël Goossens, Emmanuel Grolleau, and Xavier Poczekajlo. 2020. Workload assignment for global real-time scheduling on unrelated multicore platforms. In *28th International Conference on Real-Time Networks and Systems (RTNS 2020), June 9–10, 2020, Paris, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3394810.3394823

1 INTRODUCTION

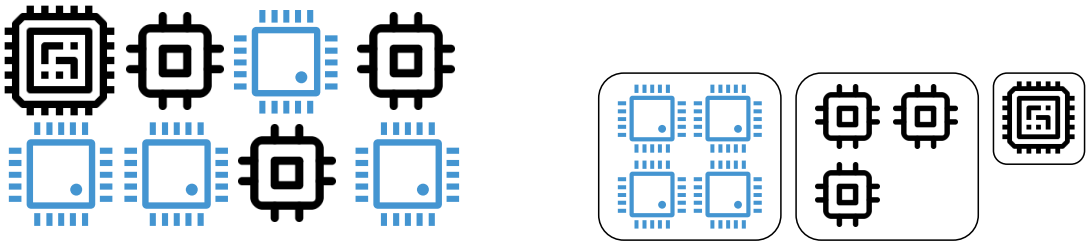
1.1 Motivation

In the keynote speech [12] of the 2019 RTNS edition, Marko Bertogna exposed how heterogeneous architectures found in current and incoming safety-critical innovations such as autonomous vehicles open promising opportunities for the real-time community to bridge practical application with theoretical aspects. *Heterogeneous MultiProcessor System-on-Chip platforms* (MPSoCs) are now widely spread in most embedded systems domains, from infotainment and automated driving system in cars to smartphones and drones. These platforms usually offer several different sets of identical computing cores, called *clusters*, and may also contain specific hardware like specialised processing units (e.g. GPU, NPU) or programmable logic tiles. The cluster architectures are typically inspired from multicore architectures and, through a hypervisor, allow a single operating system (*OS*) to globally schedule tasks easily and efficiently. Nevertheless on heterogeneous MPSoCs, different clusters may have different instruction set architectures (*ISA*), and may host different OSs. For example, the STM32MP157C-DK2® MPSoC from STMicroelectronics is composed of two clusters. On one hand its Cortex-A7® multicore cluster has a Memory Management Unit (*MMU*) allowing memory virtualisation, and can host a multi-purpose Linux OS. On the other hand, its microcontroller Cortex-M4® is a single-core cluster without MMU that can only support lightweight OS (e.g. FreeRTOS or minimal single process RTOS compliant with the POSIX 1003.13 PSE51 [1] profile) or be used directly bare metal. While belonging to the same ARM Cortex® family, these cores have different ISA. Thus, performing a task migration from one cluster to another requires the task code to be compiled for both types of architecture. Moreover, preemption cannot be allowed between every instruction since the low level instructions are different, and can

The research is done in the context of the SOFIST project, supported by Project ARC (Concerted Research Action) of Federation Wallonie-Bruxelles. This research is also supported by the European Union's Horizon 2020 research and innovation program under grant agreement N. 826610.

RTNS 2020, June 9–10, 2020, Paris, France

ACM ISBN 978-1-4503-7593-1/20/06…$15.00
https://doi.org/10.1145/3394810.3394823



(a) flat platform model (b) unrelated multicore platforms model

Figure 1: Illustration of flat versus clustered platform model

even be executed out of order on some architectures. On these platforms, migrating a task from a cluster to another would therefore require to determine specific migration points in the code.

This paper is focused on real-time scheduling of a set of tasks. Each task releases a —potentially— infinite set of jobs which have the same worst-case execution-time (WCET) on a *fictional core*. A fictional core is used as a reference to express core processing rates. It is modelled so that all tasks may be executed on it. Jobs must be completed by a given deadline to respect the real-time constraints. In the literature, a platform is often viewed as "flat", as represented in Figure 1(a). In extenso, there is no hierarchy between cores and all migrations are considered as having the same cost. This is an abstraction since most modern platforms are composed of one or several *clusters* of cores, as represented in Figure 1(b). Cluster cores are identical, but may differ from one cluster to another in the case of *unrelated multicore platforms*. In this paper, the jobs are executed on a computing platform of *unrelated clusters*. Each cluster is characterised by its number of *identical* cores, and each task has a specific processing rate on each cluster. In the literature, multiprocessor systems are generally classified into three categories [9, 17]. (1) *Identical*: all the processors are identical and execute the tasks at the same processing rate; (2) *Uniform*: each processor is characterised by a speed, e.g., a processor of speed 2 executes any task at two times faster than a processor of speed 1; (3) *Unrelated*: the processing rate depends on both the processor and the task. There exists a fourth category: *consistent architecture*. This is a particular case of *unrelated* architecture where the heterogeneity is *consistent*. Informally whenever a processor executes a task *faster* than the others processors then its also the case for all other tasks. While this notion was already considered in the literature, see for instance [5, 14] we provide in Section 2.3 a formalisation of this kind of architecture.

The scheduler on a multiprocessor platform can be *global* or *partitioned*. In *global* scheduling, any job may be executed on any core, i.e. migrate without restriction. By contrast, in *partitioned* scheduling each task is assigned to a single core and neither task nor job migration are not allowed. The multicore cluster model allows for an intermediary category: in *clustered* scheduling [9] each task is assigned to a single cluster and jobs can only migrate between cores within the cluster. In this paper, we assume a *global* scheduling of *unrelated* multicore platforms The migrations between cores of the same cluster are defined as *intra-cluster migrations* while the *inter-cluster migrations* correspond to migrations between cores of different clusters. On most platforms, *inter-cluster migrations* require software support and a specific development effort. This is very costly because of the online execution overhead and time consuming as it requires specific development effort. Today, popular scheduler implementations support symmetrical multiprocessing (SMP) that allows intra-cluster migrations (e.g. the Completely Fair Scheduler (CFS) of the Linux kernel). They are therefore transparent to the application developer, and the online overhead generated is smaller than the *inter-cluster migration* one.

1.2 State of the art

Partitioned scheduling on heterogeneous platforms is a NP-hard problem and has been studied in several works [7, 10, 28]. Global scheduling on heterogeneous platforms, also known as unrelated multiprocessor platforms, was initiated by the seminal paper [6]. Since then, the global scheduling on unrelated platforms has received less attention. This may be due to the fact that hardware platforms generally do not support inter-cluster migration of tasks, that may require a full software support. However, global scheduling allows theoretically a full utilisation of the platform. Moreover, in our case (details will follow: offline scheduling and independent tasks with implicit deadlines), there exists a polynomial-time feasibility test. In the literature, e.g. in [6, 15], the global scheduling of unrelated platforms is performed in two phases. All the computations are done offline. First, a workload assignment matrix is computed. The workload assignment decides which fraction of processing capacity of a core has to be assigned to each task. Secondly, giving this workload assignment, a template schedule is built. The template schedule is then directly used online.

Recently, MPSoCs with unrelated clusters sharing the same ISA, like the ARM big.LITTLE® architecture, have motivated some work [15] on the optimal global scheduling. Indeed, sharing the same ISA makes the inter-cluster migrations more realistic. In the latter work, the authors adopt a novel strategy, taking into account the hierarchical nature of the set of clusters. They first focus on the assignment of tasks to clusters, and then on cores, which limits the number of inter-cluster migrations. Nevertheless, this method, called Hetero-Split, is limited to a platform with only *two types* of clusters. These two-types platforms also motivated clustered approach with intra-migration like in [27]. New platforms, integrating more than two types of clusters like the Mediatek Helio X20® are developed. This MPSoC includes three clusters (two fast

Cortex-A72® cores, four middle speed Cortex-A53® cores and four slow Cortex-A53® cores) sharing the same ISA with a hardware support for inter-cluster migration. This revives interest in the global scheduling of unrelated clusters.

1.3 Contributions and organisation

In this work, we introduce a new model with a hierarchical platform view. To the best of our knowledge, this hierarchical platform model has only been addressed in the context of optimal global scheduling with two types of clusters. We take advantage of this model by proposing several new workload assignment methods derived from former methods. Those new methods are then tested by simulation, showing their advantages over the existing method. These workload assignment methods show a reduced amount of inter-cluster migrations thus improving their applicability. Finally, we discuss the gap between the current theoretical approaches to deal with heterogeneous platforms and the reality.

Section 2 introduces the new model and Section 3 presents the new workload assignment methods. We then evaluate the performances of the new methods in Section 4, and discuss the practicability of global scheduling on heterogeneous platforms in Section 5.

2 TASK AND PLATFORM MODEL

2.1 Task model

The workload is modelled by a set of n *periodic tasks* $\Gamma \doteq \{\tau_i \mid i = 1, \ldots, n\}$ (the symbol $\doteq$ means *is equal by definition to*). Each task τ_i is defined by two parameters (C_i, T_i) where C_i is the *worst-case execution time* on a *fictional processor* —chosen arbitrarily— for every task, and T_i is the release *period*. Each task releases a *job* every period T_i. The first job of a task is released at $t = 0$, the k^{th} at $t = k \times T_i$ and has to complete by $(k + 1) \times T_i$ (tasks are said to have implicit deadlines). The utilisation of a task τ_i is $u_i \doteq \frac{C_i}{T_i}$. It is defined on a *fictional core*.

2.2 Platform model

An unrelated multicore platform is modelled by a set Π of $\dot{m}$ clusters $\Pi \doteq \{\dot{\pi}_h \mid h = 1, \ldots, \dot{m}\}$. Each cluster $\dot{\pi}_h$ contains $\dot{m}_h$ identical cores $\dot{\pi}_h \doteq \{\pi_{h_1}, \ldots, \pi_{h_{\dot{m}_h}}\}$. A job of τ_i that is executed on a core π_{h_k} for t time units will progress by $\dot{r}_{i,h} \times t$ units of its execution time. Within the cluster $\dot{\pi}_h$, every core has the same processing rate $\dot{r}_{i,h}$ for each task τ_i. If $\dot{r}_{i,h} = 0$, then τ_i cannot be executed on the cluster $\dot{\pi}_h$, this couple task/cluster is said to be incompatible. A job of τ_i is completed when its progress reaches its WCET C_i.

2.3 Consistent clusters

Please find in this section a formalisation of the notion of *consistent* clusters. First to be *consistent* the platform must have a *relative order* on the clusters.

Definition 2.1 (Faster cluster). A cluster $\dot{\pi}_k$ is *faster* than cluster $\dot{\pi}_\ell$ ($\dot{\pi}_k \geq \dot{\pi}_\ell$) if

$$\forall 1 \leq i \leq n \qquad \dot{r}_{i,k} \geq \dot{r}_{i,\ell}$$

Now we introduce a tie-breaker to have the notion of the *fastest* cluster:

Definition 2.2 (Fastest cluster). $\dot{\pi}_k$ is defined to be *the fastest* processor if k is the smallest index such that $\forall 1 \leq \ell \leq m \quad \dot{\pi}_k \geq \dot{\pi}_\ell$

Wlog (by reordering the clusters) we can assume that $\dot{\pi}_1$ is the fastest cluster. By repeating the same definition on the remaining clusters and wlog we can assume, if the platform is *consistent*, that $\dot{\pi}_1 > \dot{\pi}_2 > \cdots > \dot{\pi}_m$, i.e., we have a *total* order on the clusters.

Scheduling tasks on a consistent clusters is a particular case of the unrelated setting but is more general than the uniform setting.

2.4 Assumptions

In this theoretical model, we make the following assumptions. We consider the time as continuous, and that a job may be preempted at any time (fluid schedule). The tasks are sequential so they cannot be executed in parallel. Preemptions and migrations are performed at no extra cost. Also, a task set is *feasible* on a given platform if, and only if, there exists a schedule where every job of every task can be completed by its deadline.

3 WORKLOAD ASSIGNMENT METHODS

As far as we know, every optimal scheduling method of the literature [28] for unrelated multiprocessor platforms (from real-time [6] or operational research [23] areas), starts with a workload assignment phase (made offline). From an input made of tasks parameters and platform rates, this phase decides the fraction of processing capacity of each core assigned to tasks. The tasks have to be completed within their period thanks to this assignment, without overloading the cores. With the exception of [15], presented in this section to serve as a comparison for the experiments of Section 4, most of the existing works have expressed the workload assignment phase as a LP problem. A LP problem can be solved in polynomial time [20].

The solution of the LP problem is a *cluster workload assignment* matrix $X = [x_{i,h}]_{i=1,\ldots,n}^{h=1,\ldots,\dot{m}}$ where $x_{i,h}$ is the fraction of a core in the cluster $\dot{\pi}_h$ used by a task τ_i.

It is a well-known fact that intra-cluster migrations are more costly in terms of time overhead and task programming effort. We quantify the impact of such migrations by the definition of the *presence* of a task on a cluster, introduced in [6]. Formally, a task τ_i has a *presence* on a cluster $\dot{\pi}_h$ iff $x_{i,h} > 0$. The *number of presences* $\dot{\text{Pr}}_i$ corresponds to the number of times where τ_i has a presence in a cluster: $\dot{\text{Pr}}_i \doteq |\{x_{i,h} > 0 \mid h = 1, \ldots, \dot{m}\}|$

A task τ_i will have to migrate between clusters if, and only if, $\dot{\text{Pr}}_i > 1$. Therefore, any presence greater than one is a *presence in excess* that will generate at least one inter-cluster migration.

In this section, we first formulate the cluster workload assignment as a LP problem and show that it is an exact feasibility test. This formulation extends the seminal LP problem of [6]. Then, we present a Mixed-Integer Linear Programming (*MILP*) formulation minimising the number of presences of tasks on clusters. Finally, we present succinctly a method from the literature limited to two types of clusters that will be experimentally compared to the other LP-based solutions.

3.1 Workload assignment as a LP problem

Assigning the workload of tasks on clusters can be expressed using three sets of constraints, defined in LP-Cluster:

LP 1 (LP-Cluster).

$$\sum_{h=1}^{\dot{m}} x_{i,h} \times \dot{r}_{i,h} = u_i \qquad i = 1, 2, \ldots, n \qquad (1)$$

$$\sum_{h=1}^{\dot{m}} x_{i,h} \leq 1 \qquad i = 1, 2, \ldots, n \qquad (2)$$

$$\sum_{i=1}^{n} x_{i,h} \leq \dot{m}_h \qquad h = 1, 2, \ldots, \dot{m} \qquad (3)$$

Equation 1 ensures that enough processing capacity is allocated to each task by reserving a processing capacity fraction on each cluster. Equation 2 constrains the total capacity fraction allocated to a task to be less than or equal to one. This ensures that the task can be scheduled without being executed on two cores at the same time (see Theorem 3.1). Equation 3 states that the used capacity of a cluster $\dot{\pi}_h$ is less than or equal to its total capacity, which is the capacity of its $\dot{m}_h$ cores.

If the *LP-Cluster* is successfully solved, the $x_{i,h}$ represent a successful cluster workload assignment (or assignment of tasks on clusters), as stated by Theorem 3.1. To construct a template schedule, our method requires a core workload assignment. Especially, we aim at minimising the number of cores used. Thus, we derive the cluster workload assignment by simply filling cores successively with capacity fractions in ascending order of the task index. In this manner, an available core is used if and only if there is no capacity left in the previous core visited. The result is a successful core workload assignment that we will be able to use to construct a template schedule.

Theorem 3.1. *A system is feasible on the platform if, and only if, LP-Cluster has a solution.*

Proof. First we prove that (i) if there is no solution to the LP, then the system is not feasible. This will occur if Equation 2 or Equation 3 are not satisfied. In the first case, there would be at least one task τ_i such that $\sum_{h=1}^{\dot{m}} x_{i,h} > 1$. It means that τ_i must be executed in parallel which is forbidden in our model of sequential tasks. In the second case, a cluster $\dot{\pi}_h$ would need a processing capacity higher than its total capacity $\dot{m}_h$.

Now we prove that (ii) finding a solution to this LP problem guarantees that the system is feasible. The proof sketch is depicted in Figure 2. The cluster workload assignment matrix X is of dimension $n \times \dot{m}$. Indeed, by construction of the LP problem, it has n rows with a sum of coefficients less than one, and $\dot{m}$ columns with a sum of coefficients less than $\dot{m}_h$, $h = 1, \ldots, \dot{m}$. First, we replace each column h, corresponding to the task assignment to cluster $\dot{\pi}_h$ by $\dot{m}_h$ columns, one for each core, such that the sum of the coefficients on each of the columns is not greater than one. For the sake of the proof, we simply consider, on each row i of the new columns $k = 1, \ldots, \dot{m}_h$, $x'_{i,h_k} \doteq \frac{x_{i,h}}{\dot{m}_h}$, such that the total capacity fraction allocated to each task on each cluster is evenly distributed on each of its cores. In this manner, we obtain a workload assignment matrix on the cores X_c of dimension $n \times M$, where $M \doteq \sum_{h=1}^{\dot{m}} \dot{m}_h$ is the total number of cores. On each column, $x'_{i,h_k} \doteq \frac{x_{i,h}}{\dot{m}_h}$ represents the capacity fraction of core π_{h_k} allocated to task τ_i. By construction, since originally the used capacity of cluster $\dot{\pi}_h$ to tasks was $\sum_{i=1}^{n} x_{i,h} \leq \dot{m}_h$,

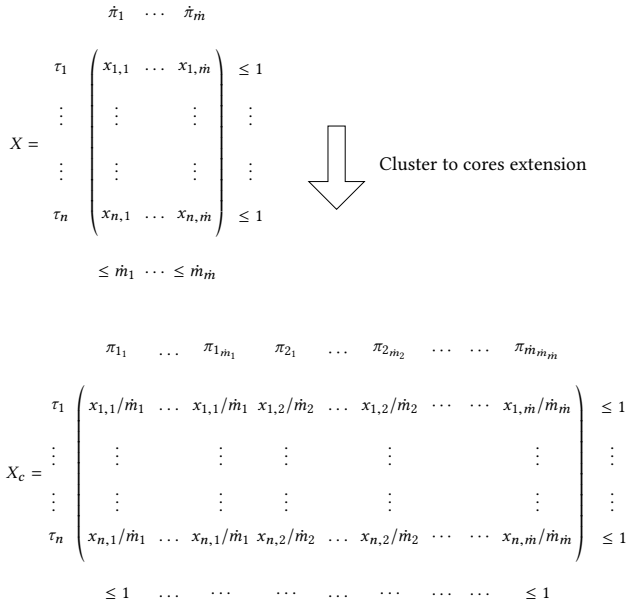


Figure 2: Proof sketch for Theorem 3.1

we have on each column for π_{j_k}, $\sum_{i=1}^{n} x'_{i,h_k} \leq 1$ (see X_c on Figure 2). From this cluster cores workload assignment matrix, we can easily create a bistochastic matrix B of size $(n+M) \times (n+M)$, as done in [23]. A *bistochastic* (or doubly stochastic) matrix is a square matrix of non-negative real numbers, having each of its rows and columns summing to 1. Formally, $\forall i = 1, \ldots, n : \sum_{h=1}^{n+M} B_{i,h} = 1$ and $\forall h = 1, \ldots, M : \sum_{i=1}^{n+M} B_{i,h} = 1$. B is constructed as follows:

$$B \doteq \left(\begin{array}{c|c} X_c & B_n \\ \hline B_M & X_c^t \end{array}\right)$$

B_n is a $n \times n$ diagonal matrix, such that $B_n(i,i) \doteq 1 - \sum_{h=1}^{\dot{m}} \sum_{k=1}^{\dot{m}_h} x'_{i,h_k}$ $\forall i$. The diagonal coefficients of B_n correspond to the *laxity* of the task τ_i, i.e. the fraction of time during which τ_i is left idle. B_M is a $M \times M$ diagonal matrix, such that $B_M(h_k, h_k) \doteq 1 - \sum_{i=1}^{n} x'_{i,h_k}$ $\forall h_k$. The diagonal coefficients of B_M correspond to the *slack* of the core π_{h_k}, i.e. the fraction of time during which π_{h_k} is left idle. X_c^t is the transpose of the core workload assignment matrix X_c, and has a dimension $M \times n$. By construction, we obtain a square bistochastic matrix B of dimension $(n+M) \times (n+M)$ expressing the fraction of each core that has to be allocated to each task, as well as the slack of the cores and the laxity of the tasks. Following the Birkhoff-von Neumann (BvN) theorem, such a matrix can be decomposed into a convex combination of permutation matrices $A \doteq \delta_1 P_1 + \delta_2 P_2 + \cdots + \delta_k P_k$ [23], where δ_i is a real coefficient $\in (0, 1]$, $\sum_{i=1}^{k} \delta_i = 1$, and P_i is a permutation matrix. A *permutation matrix* is a binary square matrix where there is exactly one 1 on each row and each column. This can be seen as a matching between tasks (rows) and cores (columns). Indeed, one and only one coefficient $P_i(h,k) = 1$ means that task on column k will be assigned to the core of the row h for a duration δ_i. The assignment

matrix X_c states that assigning a ratio of x'_{i,h_k} of core π_{h_k} to task τ_i during each of its periods ensures that its jobs will be completed. However, we need to ensure that a job is never executed on two different clusters at the same time.

For each time window $[t_1, t_k)$, between two successive releases at times t_1 and t_k (or deadlines since tasks have implicit deadlines), we can use the BvN decomposition to create such a schedule. We use the matching P_1 on the time window $[t_1, \delta_1 \times (t_k - t_1))$, by definition of a permutation matrix, this matching ensures that a task is assigned to at most one core in this time windows. Similarly, we can use the following permutation matrices obtained in the BvN decomposition, each permutation matrix P_i covering a sub-interval of duration $\delta_i \times (t_k - t_1)$. Since by the BvN theorem, $\sum_{i=1}^{k} \delta_i = 1$, we can completely schedule every task on the interval $[t_1, t_k)$, ensuring that a task is never executed on more than one core at the same time. This one time unit schedule can then be *stretched* to fit into intervals of time delimited by successive task release dates. This technique is also referred to as deadline partitioning[25]. □

3.2 About linear algebra for scheduling purposes

Theorem 3.1 shows that finding a solution to *LP-Cluster* asserts the feasibility of the system. Moreover, it shows that building a schedule from a workload assignment matrix is exactly equivalent to finding a BvN decomposition of this matrix. This result indicates that linear algebra results could be used to improve the schedule construction.

One may note that minimising the number of permutation matrices in a BvN decomposition is similar to minimising the number of scheduling decisions. Indeed, each different permutation matrix corresponds to a different schedule decision (i.e. which jobs are executed at a given instant, and on which cores). Taking schedule decisions lead to preemptions and/or migrations (both inter- or intra-cluster). Therefore, minimising the number of scheduling points may be a solution to reduce the number of preemption and migrations. This is an example of optimisation of the template schedule construction [6, 13, 15].

The next property concerns the complexity of the BvN decomposition:

Theorem 3.2 (Dufossé 2016 [18]). *The problem of deciding if there is a BvN decomposition of a given doubly stochastic matrix with k permutation matrices is NP-complete in the strong sense.*

Since the decision problem is NP-complete in the strong sense, the optimisation problem of minimising the number of permutation matrices in a BvN decomposition is NP-hard in the strong sense. Thus, optimising the number of scheduling decisions cannot be done efficiently.

In the remainder, we focus only on modifying the workload assignment to reduce the number of preemption and migration. However, using linear algebra techniques to *sub-optimally* reduce the number of scheduling decisions will be explored in future works.

3.3 LP-Feas and LP-CFeas

In [6], author presents a LP-Feas, a LP model for assigning the workload on an unrelated real-time multiprocessor platform. This work was primarily focused on feasibility, and does not aim at minimising the number of presences. It is very close to the LP formulation of the makespan minimisation in job shop scheduling on unrelated single-core processors given in [23]. In this work, the model is using a *flat* hardware representation. To fit our model notations, we consider a hierarchical hardware with one core per cluster, i.e. $\forall h, \dot{m}_h = 1$.

LP 2 (LP-Feas [6]). *the workload assignment is solution of the following LP:*

$$\sum_{h=1}^{\dot{m}} x_{i,h} \times \dot{r}_{i,h} = u_i \qquad i = 1, 2, \ldots, n \tag{4}$$

$$\sum_{h=1}^{\dot{m}} x_{i,h} \leq \ell \qquad i = 1, 2, \ldots, n \tag{5}$$

$$\sum_{i=1}^{n} x_{i,h} \leq \ell \qquad h = 1, 2, \ldots, \dot{m} \tag{6}$$

Minimise makespan objective: Minimise ℓ, the system is feasible if, and only if, $\ell \leq 1$. □

The immediate extension of LP-Feas to clusters is the following:

LP 3 (LP-CFeas). *The workload assignment is solution of the following LP:*

$$\sum_{h=1}^{\dot{m}} x_{i,h} \times \dot{r}_{i,h} = u_i \qquad i = 1, 2, \ldots, n \tag{7}$$

$$\sum_{h=1}^{\dot{m}} x_{i,h} \leq \ell \qquad i = 1, 2, \ldots, n \tag{8}$$

$$\sum_{i=1}^{n} x_{i,h} \leq \dot{m}_h \times \ell \qquad h = 1, 2, \ldots, \dot{m} \tag{9}$$

Minimise makespan objective: Minimise ℓ, the system is feasible if, and only if, $\ell \leq 1$. □

LP-Feas and LP-CFeas differ in Equations 6 and 9: since on a unrelated multicore platform, a cluster $\dot{\pi}_h$ has $\dot{m}_h$ cores, a total capacity of $\dot{m}_h$ can be allocated to tasks. It is straightforward that the condition $\ell \leq 1$ constrains solutions of LP-CFeas to be solutions of LP-Cluster. Therefore by Theroem 3.1, a solution of LP-CFeas with $\ell \leq 1$ can be used to build a feasible schedule.

3.4 LP-Load and LP-CLoad

LP-Feas and LP-CFeas tend to reduce the makespan of the schedule that will be stretched between successive releases. As an example, let two tasks be scheduled on two very different cores: one being ten times faster than the other one for all the tasks. Consider the system of two tasks $\Gamma = \{\tau_1, \tau_2\}$, with both WCET given by $C_1 = C_2 = 5$ and both periods given by $T_1 = T_2 = 10$. The platform is composed of two clusters of one core each, with $\Pi = \{\dot{\pi}_1, \dot{\pi}_2\}$, both clusters having only one core $\dot{m}_1 = \dot{m}_2 = 1$, and having respective rates $\dot{r}_{1,1} = \dot{r}_{2,1} = 10$ for $\dot{\pi}_1$, and $\dot{r}_{1,2} = \dot{r}_{2,2} = 1$ for $\dot{\pi}_2$. The workload assignment matrix computed by LP-CFeas (or equivalently LP-Feas since clusters have one core) is given by $X_{LP-CFeas} = \begin{bmatrix} 5/11 & 5/11 \\ 5/11 & 5/11 \end{bmatrix}$ ($5/11 \approx 0.4545$). This would lead to a

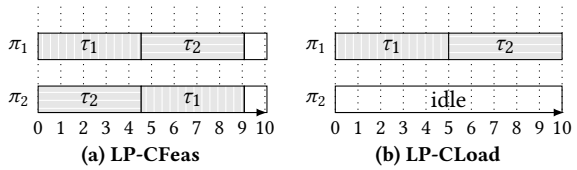


Figure 3: Rectangle schedule computed from LP-CFeas versus schedule favouring fast cores utilisation computed from LP-Cload

schedule repeated between every successive release (which is every ten time units in our simple example since both tasks have a period of 10), as shown in Figure 3(a).

When considering the number of presences of tasks on clusters, a more interesting workload assignment would favour a high utilisation, or load, on faster cores: $X_{LP-CLoad} = \begin{bmatrix} 1/2 & 1/2 \\ 0 & 0 \end{bmatrix}$. Such workload assignment could lead to a schedule such as Figure 3(b), which does not produce any inter-cluster migration. LP-CLoad is a LP formulation with the same constraints as LP-Cluster, with the objective of minimising the used capacity of the system. On the unrelated multicore platforms problem, it is defined for LP-Cluster as: ***LP-CLoad:*** Minimise $\sum_{i=1}^{n} \sum_{h=1}^{\dot{m}} x_{i,h}$.
LP-CLoad can be used in the context of a flat platform model. To do so, one simply has to assume that each core is a cluster of size one, i.e. $\dot{m}_h = 1$ for every cluster $\dot{\pi}_h$.

3.5 Minimal number of presences: ILP-CMig

Even if non polynomial, an optimal method minimising the number of presences of tasks on clusters can be useful. Indeed, a system designer may prefer spending a couple of hours waiting for the assignment to be computed rather than spending development time and facing the complexity to implement an inter-cluster migration. Since we are working at the cluster level, the size of the problem, at least in the number of clusters, can be relatively small in practice. We propose a Mixed Integer Linear Programming (*MILP*) formulation called *ILP-CMig*, based on the LP-Cluster. In addition, we introduce a boolean variable $b_{i,h}$. Variable $b_{i,h}$ is 1 if task τ_i is present on cluster $\dot{\pi}_h$, and 0 otherwise. The objective is to minimise the total number of presences.

LP 4 (ILP-CMig). *The workload assignment is solution of LP-Cluster (Equations 1, 2, 3) with the following additional constraints:*

$$b_{i,h} \in \{0,1\} \qquad i = 1,\dots,n; h = 1,\dots,\dot{m} \qquad (10)$$

$$x_{i,h} \leq b_{i,h} \qquad i = 1,\dots,n; h = 1,\dots,\dot{m} \qquad (11)$$

$$b_{i,h} < 1 + x_{i,h} \qquad i = 1,\dots,n; h = 1,\dots,\dot{m} \qquad (12)$$

Minimise $\sum_i^n \sum_h^{\dot{m}} b_{i,h}$.

The non-clustered version ILP-Mig has the same set of constraints than LP-Cluster where each cluster is considered as a single core with the computing capacity of m cores.

3.6 Hetero-Split

In order to compare the previous methods to an efficient existing one, we consider Hetero-Split [15]. This algorithm solves an equivalent expression of LP-Cluster with a $O(n \log n)$ time complexity. It is restricted to systems having only two different types of clusters. A property of this algorithm is to limit the number of tasks having a number of presences in excess higher than one (i.e., assigned to the two different clusters) to the total number of cores. Since there are only two clusters, tasks are classified into two categories: either cluster $\dot{\pi}_1$ is more efficient, or it is $\dot{\pi}_2$ that is more efficient for their execution. The method exploits this dual property and thus cannot be easily extended to more than two types of clusters. Nevertheless, it allows the use of McNaughton wrap-around rule to efficiently create a schedule conforming to the workload assignment.

4 EXPERIMENTAL COMPARISON OF WORKLOAD ASSIGNMENT METHODS

When neglecting the migration cost, every workload assignment method presented in Section 3 is optimal regarding the feasibility. Since we know that this hypothesis is unrealistic, we compare the number of presences in excess $\dot{\Pr}_i - 1$ for the six presented methods. The number of presences in excess is a lower bound on the number of inter-cluster migrations. The LP based methods, as well as Hetero-Split, are polynomial time methods, while the MILP based method has an exponential time complexity regarding the number of clusters. In this section, we compare the following methods:

- LP-Feas is the method minimising the makespan proposed in [6] considering the "flat" core model, while its clustered version LP-CFeas presented in Section 3.3 considers the hierarchical clustered model;
- LP-Load (see Section 3.4), whose objective is to minimise the total core utilisation, its clustered version is LP-CLoad;
- Hetero-Split ([15]) a linear algorithm limited to two types of clusters;
- ILP-Mig is the "flat" core-based version of ILP-CMig, a MILP problem minimising the number of presences on clusters. In practice, ILP-CMig uses significantly fewer variables than ILP-Mig.

4.1 Experimental setup

In Section 2.3 we formalised the notion of *consistent* clusters. In our opinion, it fits more precisely to certain realistic platforms where cores have different micro-architectures but identical ISA, as the big.LITTLE® or the Helio X20®.

For the number of presences and simulation experiments, we have generated the systems as follows. The number of types of clusters $\dot{m}$ is either 2 or 5. The former in order to compare Hetero-Split to the other methods, and the latter because five different types of clusters is considered a large size for a heterogeneous MPSoC nowadays. Then, the number of cores per type of cluster is set in [2, 5]. The number of tasks n is arbitrary bounded as follows: $\dot{m} \leq n \leq 10 \times \dot{m}$. We then generate every task such that its period T_i is determined using [19]. The parameter C_i is based on T_i: $\frac{T_i}{2} \leq C_i \leq T_i$. We then generate the rates randomly and adjust them so that the tasks fit the given utilisation. For experimentation purposes, the clusters (the rates in particular) may be set to consistent.

Using this generator, we generate 1 000 systems per total utilisation range $u \in [p - 0.1, p)$, increasing p from 0.4 to 1, for both $\dot{m} = 2$ and $\dot{m} = 5$. The ratio $p = 1$ corresponds to a full utilisation

of the platform by the tasks. Here, p is equal to the value of the LP-CFeas objective function result, which is the minimal platform utilisation. The experimentation compares the different scheduling methods over 28 000 randomly generated test systems. As ILP-Mig and ILP-CMig have an exponential time complexity, they are tested using only a subset of the generated systems.

4.2 Inter-cluster number of presences in excess

The workload assignment methods are compared in terms of inter-cluster presences in Figure 4. First note that the scale is 10^{-2}, meaning that in average, very few tasks are assigned to different clusters, for both two and five types of clusters.

On the left-hand graph (with $\dot{m} = 2$), we observe that Hetero-Split performs close to LP-CFeas for low platform utilisation. At higher platform utilisation, Hetero-Split dominates the other polynomial time assignment methods. We can see that both the *Feas*-based LP solutions perform poorly at low platform utilisation compared to the *Load*-based LP solutions for both two-types and five-types of cores. This is due to *Feas* objective that tends to create "rectangular" (i.e. all processors tend to be idle at the same instant) schedules by balancing the tasks workload on different cores or clusters, as illustrated in Figure 3. While the platform utilisation increases, the slack left at the right-hand side of this rectangle reduces, and the solutions provided by both objective functions tend to be similar. At high platform utilisation, we thus see that both clustered versions of the LP outperform both non-clustered versions. When combining the two advantages —both the clustered version and the Load objective function—, we observe two to four times fewer inter-cluster migrations compared to the seminal non-clustered Feas objective function. On the right-hand graph, we see the proportion of generated systems for which the assignment is completely clustered for two types of clusters. It is close to 100% for the ILP-CMig, while the clustered LP-CLoad dominates all the other methods in terms of ratio of completely clustered workload assignments. When comparing both graphs, we can observe that LP-CLoad performs better than Hetero-Split regarding tasks clustering on consistent two-types systems. However, Hetero-split performs better in average on arbitrary clusters.

4.3 Runtime measurement

The performance of the LP/ILP based solution in terms of execution time are depicted in Table 1. The experiment has been conducted on a Intel I7500® multicore processor. The left table gives the performances of the LP/ILP based solution with the same test systems. In this experiment, the system utilisations are uniformly distributed in the range [0.3, 1.0]. The rest of the system parameters are generated as in Section 4.1. The table on the right gives the average performances with test systems ordered by number of tasks. Thus, both tables are not comparable because they do not have the same test systems. The left table gives the average computation time, per LP or ILP for both $\dot{m} = 2$ and $\dot{m} = 5$ on unrelated clusters. For example, ILP-Mig took an average of 0.061 second to compute the workload assignment with $\dot{m} = 2$. We observe that the clustered version of a LP or an ILP is always faster than the non-clustered version, which is normal since there are fewer variables in the clustered versions. Also, the execution time from $\dot{m} = 2$ to $\dot{m} = 5$ increases drastically

	$\dot{m} = 2$	$\dot{m} = 5$
LP-Feas	0.013	0.464
LP-Load	0.012	0.562
LP-CFeas	0.002	0.027
LP-CLoad	0.002	0.029
Hetero-Split	0.007	NA
ILP-Mig	0.061	NA
ILP-CMig	0.018	0.068

ILP-Mig, $\dot{m} = 2$	
n	time (s)
10	0.811
11	1.736
12	3.562
13	8.271
14	16.665
15	28.492
16	69.782
17	130.582

Table 1: Average execution time of the workload assignment methods in seconds

and this affects less the clustered versions, since there are fewer additional cluster variables than core variables. The table on the right gives the performance of ILP-CMig with $\dot{m} = 2$ for n tasks. It clearly shows that the execution time grows exponentially with the number of tasks, making it intractable for large systems.

5 GLOBAL SCHEDULING APPLICABILITY ON HETEROGENEOUS PLATFORMS

In this section, we examine the applicability of the provided results. Specifically, we discuss the applicability of the global scheduling on unrelated platforms. First, we study the validity of considering migrations upon an unrelated platform. Second, we expose some limitations on the use of a template to schedule tasks in practice.

5.1 How realistic is the migration of tasks upon unrelated platforms?

Real heterogeneous platforms. The global scheduling on unrelated processors makes the assumption that the migration of tasks between processors with heterogeneous architectures is possible. As mentioned in [26] to motivate partitioned scheduling, the migration between processors with different instruction sets is at least challenging if not unrealistic. Indeed, tasks code should be compiled for both architectures and migration points determined beforehand if jobs migration is allowed. Consequently, the context of the migrating task —the state of all active job local variables— should also be saved and transferred to the destination core (e.g. by using the OpenAMP framework) to ensure the continuation of this task at the point it has stopped. Nevertheless, recent platforms as ARM big.LITTLE® are game-changing. Those platforms embed clusters of cores different in their micro-architecture (*asymmetric*) but having similar ISA (instructions set architecture) with full cache coherence. This allows migrations between cores belonging to different clusters. Specifically, ARM big.LITTLE® platforms are made of a cluster of fast but energy-consuming (big) cores and a cluster of slower but more energy-efficient (LITTLE) cores. These platforms are unrelated and cannot be classified as uniform as the rate of execution still depends both on the task and the executing core. They are *consistent* (see Section 4.1) in the sense that there exists an order of magnitude between the core processing rate of different clusters: big cores executes tasks always faster than LITTLE cores but not always with the same magnitude.

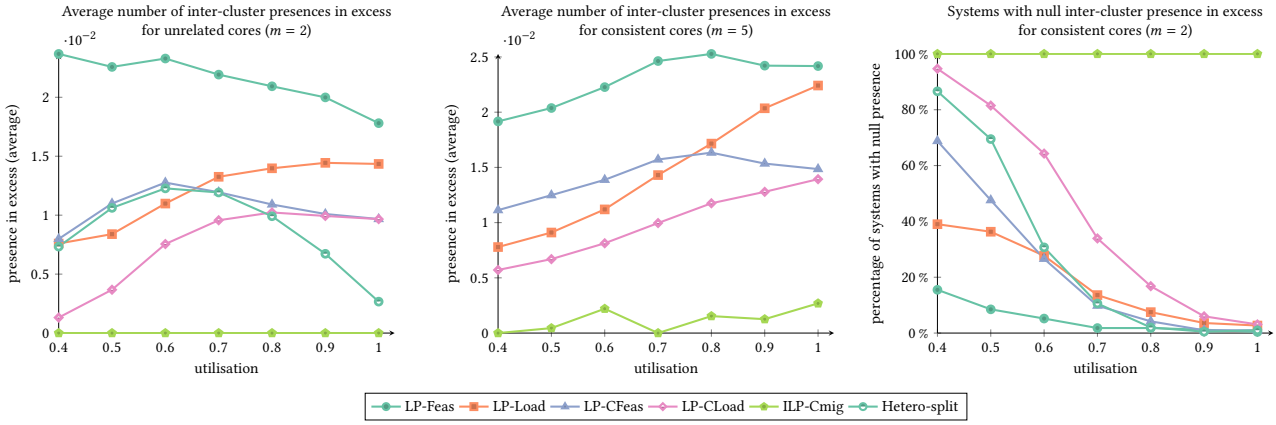
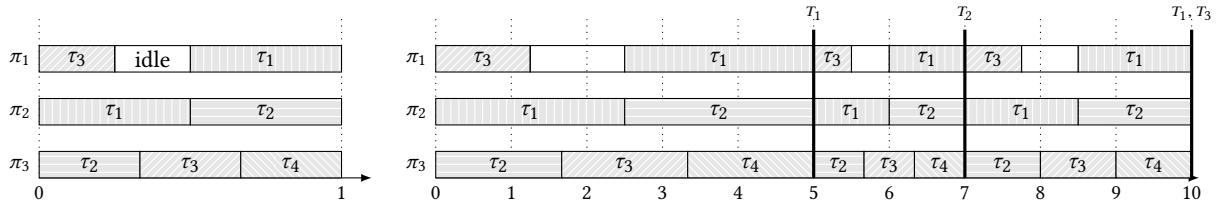


Figure 4: Number of presences by workload assignment method for unrelated and consistent clusters



(a) Template schedule example

(b) Stretched template schedule example

Figure 5: Template-based scheduling

Applied scheduling in heterogeneous platforms. In practice, the Energy Aware Scheduler (EAS) [22] has been recently integrated in the mainline linux kernel (from version 5.0) and supports migration between single-ISA cores of asymmetric clusters. Based on a model of the cores energy consumption, EAS selects which core to use —the most energy efficient— and at which frequency using the dynamic voltage and frequency scaling (DVFS) mechanism. In line with the existing Completely Fair Scheduler (CFS) [21], EAS aims at providing a fair distribution of the cores utilisation to non real-time tasks while maximising the overall performance but also optimising energy usage. When CFS is designed for identical cores (symmetrical multi-processing or SMP in the OS terminology), EAS takes advantage of recent asymmetric multi-processing (AMP) platforms as ARM big.LITTLE®. SCHED_DEADLINE [24] is an implementation of the global EDF scheduler [8] for real-time tasks in Linux together with the Constant Bandwidth Server (CBS) [4] algorithm to manage non real-time tasks. SCHED_DEADLINE was designed for identical cores (SMP) and may starve all the tasks upon a consistent (AMP) platform. This issue amongst other was discussed by Luca Abeni in talks given during the last two editions of the OSPM summit [3, 16]. He proposed some practical solutions (submitted as a linux kernel patchset in [2]) as adapting the admission control test of tasks and selecting the least energy consuming processor capable of executing the pending task. Interestingly, the latter proposal is valid because the two clusters of cores are consistent. Consequently, studying the global scheduling of unrelated platform is not only theoretical but practicable and the particular case of consistent cores fits well to those *modern* platforms.

5.2 Template-based scheduling

In Section 3, we detail the workload assignment phase of tasks to cores. To the best of our knowledge, any global scheduler on heterogeneous platform starts with such a workload assignment phase. This workload assignment phase is then followed by the construction of a *template schedule*. This *template schedule* contains a feasible schedule of the periodic task set, over a time instant. An example of a template schedule is given in Figure 5(a). We will now discuss the possible usages of this template schedule.

In [6, 15], this template schedule is stretched between every absolute task releases, as illustrated in Figure 5(b). Repeating the template schedule every time unit is acceptable for a feasibility test. It is however not acceptable in practice, due to the number of preemptions and migrations involved. To limit the online overhead, some techniques have been developed to improve the average case, as [11]. This work is designed in the context of an affinity mask model, with sporadic tasks. In the worst case, however, this optimisation has no effect. Other works, as [29], loose the hypothesis of hard real-time tasks to soft real-time tasks with bounded tardiness and allow intra-task parallelism via a DAG-task model. This change of paradigm allows to reduce the overhead and thus improves the use in practice. However, this change of paradigm may not be applied in the general case. One could also argue that considering that the rate of a task is constant on a given core is unrealistic. For example, if a task has a first part involving intensive integer computation, followed by a second part of intensive floating point computation, the rate of execution of both parts would differ depending on whether the core as a FPU or not. As a result, the

task should be split in more homogeneous sub-tasks, such that each task can be considered as having a constant rate of execution on each core. This requires a DAG-task model or at least a model that handles chains of tasks to be considered.

The use of identical platform scheduling techniques for unrelated platform scheduling has been explored. However, those techniques seem to be hard to generalise to unrelated platforms. In [15], the authors propose a global scheduling algorithm for periodic tasks on a 2-types heterogeneous platform. After the workload assignment phase *hetero-split*, it performs the *2-types McNaughton hetero-wrap*. This *2-types McNaughton* assigns the fractions of processing capacity of a task for both cluster at once while preventing parallelism. To fill both cluster at once, the first cluster is filled from left to right, while the second one is filled symmetrically from the right to the left. Adapting this seminal *McNaughton* to two types platform required several transformations. As indicated in Section 3.6, this transformations based on a symmetrical filling seems hard to generalise to unrelated platforms with any number of types.

As mentioned in [11], the template schedule produced could be optimised in terms of preemptions with heuristics reordering the windows delimited by scheduling points. However, it would not decrease the number of presences and the schedule would remain static.

Repeating the same sequence over and over reduces the adaptability of the system. To avoid this, an option would be to avoid the use of a template schedule. Designing a more dynamic scheduler such as EDF or Last Laxity First (LLF) may be difficult or require a very pessimistic approach. We believe that for consistent platforms, the design of dynamic schedulers will be made much simpler due to the monotonous characteristic of such platforms. The use of consistent platform would ease both the design of the platform and the usability in practice, as those platforms become more and more common on the market.

6 CONCLUSION

Starting from the observation that inter-processor migrations are more costly than intra-processor migrations, we propose a new model to handle those two types of migrations differently. Based on previous works, we use this cluster-based model to design the workload assignment of an optimal scheduler on unrelated platforms. To do so, we propose a LP formulation, with several objective functions. We show that this LP formulation is an exact feasibility test and that its output may be used to construct an offline schedule. We also propose an ILP formulation that is optimal regarding both schedulability and the number of inter-cluster migrations. Using simulation, we demonstrate the impact of our model on the number of inter-cluster migrations. Our new solution outperforms the existing one. The optimal ILP is used as a reference.

This new workload assignment thus improves the applicability of global scheduling on unrelated platform, by reducing the online overheads. We discuss the applicability of global scheduling in the context of unrelated platform. We show that global scheduling for heterogeneous platforms is actually used in practice. Consistent platforms are likely to be used in this context, because the theoretical migration model is close from their behaviour for such platforms. We also discuss the existing usages of global scheduling for the general unrelated case and their limitations.

In the future, we intend to evaluate the performance of an online scheduler, such as a global EDF as such an algorithm may be more usable in practice. Closing the gap between theory and practice could also be done by observing more complex task model, such as Gang scheduling or a DAG-based task model.

REFERENCES

[1] 2004. IEEE Standard for Information Technology- Standardized Application Environment Profile (AEP)-POSIX Realtime and Embedded Application Support. *IEEE Std 1003.13-2003 (Revision of IEEE Std 1003.13-1998)* (2004), i–164.

[2] Luca Abeni. 2019. *RFC PATCH 0/6] Capacity awareness for SCHED_DEADLINE.* https://lkml.org/lkml/2019/5/6/11

[3] Luca Abeni. 2019. SCHED_DEADLINE on heterogeneous multicores. (2019). https://lwn.net/Articles/793281/ Power Management and Scheduling in the Linux Kernel (OSPM summit III).

[4] Luca Abeni and Giorgio C. Buttazzo. 1998. Integrating Multimedia Applications in Hard Real-Time Systems. In *Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain, December 2-4, 1998.* 4–13. https://doi.org/10.1109/REAL.1998.739726

[5] Robert K. Armstrong. 1997. *Investigation of effect of different run-time distributions on smartnet performance.* Master's thesis. Naval Postgraduate Scholl, Monterey, California.

[6] Sanjoy K. Baruah. 2004. Feasibility Analysis of Preemptive Real-Time Systems upon Heterogeneous Multiprocessor Platforms. In *Real-Time Systems Symposium.* IEEE, 37–46.

[7] Sanjoy K. Baruah. 2004. Partitioning Real-Time Tasks among Heterogeneous Multiprocessors. In *33rd International Conference on Parallel Processing (ICPP).* IEEE, 467–474.

[8] Sanjoy K. Baruah and Theodore P. Baker. 2008. Schedulability analysis of global EDF. *Real-Time Systems* 38, 3 (2008), 223–235. https://doi.org/10.1007/s11241-007-9047-9

[9] Sanjoy K. Baruah, Marko Bertogna, and Giorgio Buttazzo. 2015. *Multiprocessor Scheduling for Real-Time Systems.* Springer.

[10] Sanjoy K. Baruah, Vincenzo Bonifaci, Renato Bruni, and Alberto Marchetti-Spaccamela. 2019. ILP models for the allocation of recurrent workloads upon heterogeneous multiprocessors. *J. Scheduling* 22, 2 (2019), 195–209.

[11] Sanjoy K. Baruah and Björn Brandenburg. 2013. Multiprocessor feasibility analysis of recurrent task systems with specified processor affinities. In *Real-Time Systems Symposium.* IEEE, 160–169.

[12] Marko Bertogna. 2019. A View on Future Challenges for the Real-Time Community. (2019). https://www.irit.fr/rtns2019/keynote/ RTNS 2019 Keynote.

[13] Antoine Bertout, Joël Goossens, Emmanuel Grolleau, and Xavier Poczekajlo. 2020. Template schedule construction for global real-time scheduling on unrelated multiprocessor platforms. In *Design, Automation and Test in Europe Conference (Grenoble, France, March 2020).*

[14] Hua Chen, Albert Mo Kim Cheng, and Ying-Wei Kuo. 2011. Assigning real-time tasks to heterogeneous processors by applying ant colony optimization. *Journal of Parallel and Distributed computing* 71, 1 (2011), 132–142.

[15] Hoon Sung Chwa, Jaebaek Seo, Jinkyu Lee, and Insik Shin. 2015. Optimal Real-Time Scheduling on Two-Type Heterogeneous Multicore Platforms. In *Real-Time Systems Symposium.* IEEE, 119–129.

[16] Jonathan Corbet. 2018. Power-aware and capacity-aware migrations for real-time tasks. (2018). https://lwn.net/Articles/754923/ Power Management and Scheduling in the Linux Kernel (OSPM summit II).

[17] Robert I Davis and Alan Burns. 2011. A survey of hard real-time scheduling for multiprocessor systems. *ACM computing surveys (CSUR)* 43, 4 (2011), 35.

[18] Fanny Dufossé and Bora Uçar. 2016. Notes on Birkhoff–von Neumann decomposition of doubly stochastic matrices. *Linear Algebra Appl.* 497 (2016), 108–115.

[19] Joël Goossens and Christophe Macq. 2001. Limitation of the hyper-period in real-time periodic task set generation. In *Proceedings of the RTS Embedded System.* 133–148.

[20] Narendra Karmarkar. 1984. A new polynomial-time algorithm for linear programming. *Combinatorica* 4, 4 (1984), 373–396. https://doi.org/10.1007/BF02579150

[21] The kernel development community. 2019. The Linux Kernel documentation: Completely Fair Scheduler. https://www.kernel.org/doc/html/latest/scheduler/sched-design-CFS.html. Accessed: 2019-12-18.

[22] The kernel development community. 2019. The Linux Kernel documentation: Energy Aware Scheduling. https://www.kernel.org/doc/html/latest/scheduler/sched-energy.html. Accessed: 2019-12-18.

[23] Eugene L. Lawler and Jacques Labetoulle. 1978. On Preemptive Scheduling of Unrelated Parallel Processors by Linear Programming. *J. ACM* 25, 4 (1978), 612–619.

[24] Juri Lelli, Claudio Scordino, Luca Abeni, and Dario Faggioli. 2016. Deadline scheduling in the Linux kernel. *Software: Practice and Experience* 46, 6 (2016), 821–839.

[25] Greg Levin, Shelby Funk, Caitlin Sadowski, Ian Pye, and Scott A. Brandt. 2010. DP-FAIR: A Simple Model for Understanding Optimal Multiprocessor Scheduling. In *22nd Euromicro Conference on Real-Time Systems, ECRTS 2010, Brussels, Belgium, July 6-9, 2010*. IEEE Computer Society, 3–13. https://doi.org/10.1109/ECRTS.2010.34

[26] Gurulingesh Raravi, Björn Andersson, and Konstantinos Bletsas. 2013. Assigning real-time tasks on heterogeneous multiprocessors with two unrelated types of processors. *Real-Time Systems* 49, 1 (2013), 29–72.

[27] Gurulingesh Raravi, Björn Andersson, Vincent Nélis, and Konstantinos Bletsas. 2014. Task assignment algorithms for two-type heterogeneous multiprocessors. *Real-Time Systems* 50, 1 (2014), 87–141.

[28] Jagpreet Singh and Nitin Auluck. 2016. Real time scheduling on heterogeneous multiprocessor systems - A survey. In *Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC)*. IEEE, 73–78.

[29] Stephen Tang, Sergey Voronov, and James H. Anderson. 2019. GEDF Tardiness: Open Problems Involving Uniform Multiprocessors and Affinity Masks Resolved. In *31st Euromicro Conference on Real-Time Systems, ECRTS 2019, July 9-12, 2019, Stuttgart, Germany*. 13:1–13:21.