HAL
archives-ouvertes.fr

A heuristic to minimize the cardinality of a real-time task set by automated task clustering

Antoine Bertout, Julien Forget, Richard Olejnik

To cite this version:

Antoine Bertout, Julien Forget, Richard Olejnik. A heuristic to minimize the cardinality of a real-time task set by automated task clustering. Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC 2014), Apr 2014, Gyeongju, South Korea. 10.1145/2554850.2554958 . hal-01016182

HAL Id: hal-01016182
https://hal.inria.fr/hal-01016182

Submitted on 28 Jun 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A heuristic to minimize the cardinality of a real-time task set by automated task clustering

Antoine Bertout, Julien Forget and Richard Olejnik
University Lille1, LIFL
Villeneuve D'Ascq, France
{antoine.bertout,julien.forget,richard.olejnik}@lifl.fr

Abstract

We propose in this paper a method to automatically map functionalities (blocks of code corresponding to high-level features) with real-time constraints to tasks (or threads). We aim at reducing the number of tasks functions are mapped to, while preserving the schedulability of the initial system. We consider independent tasks running on a single processor. Our approach has been applied with fixed-task or fixed-job priorities assigned in a Deadline Monotonic (DM) or a Earliest Deadline First (EDF) manner.

1 Introduction

Our work falls within the scope of real-time systems programming. Usually, real-time system developers design a system as a set of functionalities with real-time constraints. A functionality is here considered a block of code corresponding to a high-level feature. Implementing such systems requires to map each functionality to a real-time task (thread). On the one hand, the number of those functionalities is quite high. For instance, it ranges from 500 to 1000 in the flight control system of an aircraft or of a space vehicle [7, 11]. On the other hand, a large number of threads implies a significant time overhead in context switching [27, 15] and an important memory footprint (e.g. task control block, size of the stack, etc.). Thus, the number of tasks supported by embedded real-time operating systems is limited, rarely over one hundred, and developers cannot map each functionality to a different task. This mapping is currently mainly performed manually and, given the number of functionalities to process, this work can be tedious and error-prone.

In our work, we address this question from the scheduling point of view. We model a system as a set of tasks with real-time constraints, where each task is characterized by an execution time, an activation period and a deadline, in the same way as Liu and Layland's task model [18]. With respect to this model, functionalities can simply be considered as finer grain tasks, while threads are just coarser tasks. Thus, mapping functionalities to tasks amounts to gathering several tasks into a single one, which we call *task clustering*. Clustering several tasks implies to choose only one deadline for the cluster, which effectively reduces some task deadlines. As a consequence, we have to check that the system schedulability is preserved after the clustering.

Related Work In the literature, task clustering is most often studied in the context of distributed systems implementation, where it consists in distributing a set of tasks over a set of computing nodes (processors or cores). This is different from our context, because in the distributed systems context a cluster corresponds to the set of tasks allocated to the same computing resource. For instance, [23, 1] aim at minimizing communications by clustering tasks that communicate a lot. The approaches in [22, 13] cluster tasks based on communications, in order to reduce the system makespan. The number of tasks of the resulting implementation is however not reduced.

Functionality to task mapping is known as runnable-to-task mapping and is identified as a step of the development process in the augmented real-time specification for AUTomotive Open System ARchitecture (AUTOSAR) [5]. This document and [27] also provide guidelines defining under which conditions runnables can be mapped to the same tasks. Authors in [32] propose an automated mapping in that context, but that work is restricted to functionalities that have deadlines equal to their periods. In [8, 21], the authors study the multi-task implementation of multi-periodic synchronous programs and must allocate the different elements of the program to tasks. The clustering is out of the scope of [21], while the heuristic proposed in [8] is very specific to the language structure.

In [26], authors aim at reducing the number of tasks in order to reduce the complexity of the scheduling problem. However, they only focus on functional requirements to group tasks, without considering timing constraints.

This research Our objective is to automate the task clustering, so as to reach a minimal task number, while preserving the system schedulability. The number of possible clusterings of a task set is equal to the number of partitions of the set, which is in the range of the *Bell number* [24]. The Bell number is exponential with respect to the cardinality of the set, so given the huge number of possibilities to explore, we use a greedy heuristic to search the partitions space. For now, we do not consider communications and the execution platform is made up of a single processor. These are strong restrictions, which will be lifted in future work. The aim of the paper is to properly define the problem and to study it in a simple setting, so as to serve as a basis for future work.

Organization The rest of the paper is organized as follows. In Section 2, we describe our clustering model. Section 3 is dedicated to the verification of cluster schedulability. We describe the way we generate solutions and the heuristic applied in Section 4. Section 5 contains the experimental results conducted on large sets of tasks, randomly generated. Finally, we expose our conclusion and the future work involved in the Section 6.

2 Problem definition

Our model, illustrated in Figure 1, is based on Liu and Layland's model [18]. A system consists of a synchronous (i.e. with offsets equal to zero) set of real-time tasks $\mathcal{S} = (\{\tau_i(C_i, D_i, T_i)\}_{1 \leq i \leq n})$ where C_i is the worst-case execution time (WCET) of τ_i, T_i is the activation period, D_i is the relative deadline with $D_i \leq T_i$. We denote $\tau_{i.k}$ the $(k+1)^{th}$ $(k \geq 0)$ instance, or *job*, of τ_i. The job $\tau_{i.k}$ is released at time $o_{i.k} = kT_i$. Every job $\tau_{i.k}$ must be completed before its absolute deadline $d_{i.k} = o_{i.k} + D_i$

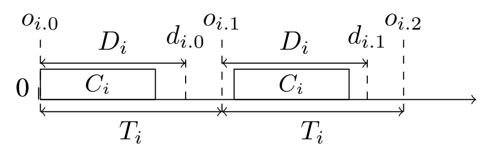


Figure 1: Task Diagram.

2.1 Scheduling

In this paper, we focus on priority-based scheduling policies, either fixed-job with Earliest Deadline First (EDF) [18] or fixed-task priority policies with Deadline Monotonic (DM) [16].

Let $\mathcal{J}$ denote the infinite set of job $\mathcal{J} = \{\tau_{i.k}, 1 \leq i \leq n, k \in \mathbb{N}\}$. Given a priority assignment Φ, we define two functions $s_\Phi, e_\Phi : \mathcal{J} \to \mathbb{N}$, where $s_\Phi(\tau_{i.k})$ is the start time and $e_\Phi(\tau_{i.k})$ is the completion time of $\tau_{i.k}$ in the schedule produced by Φ.

Definition 1. *Let $\mathcal{S} = (\{\tau_i\}_{1 \leq i \leq n})$ be a task set and Φ be a priority assignment. S is* schedulable *under Φ if and only if:* $\forall \tau_{i.k}, e_\Phi(\tau_{i.k}) \leq d_{i.k} \wedge s_\Phi(\tau_{i.k}) \geq o_{i.k}$

In the sequel, we will also rely on the notion of *laxity*.

Definition 2. *Laxity L (or* slack time*) indicates the maximum delay that can be taken by the task without exceeding its deadline: $L_i = D_i - C_i$.*

2.2 Clustering

Definition 3. *Clustering τ_i and τ_j, where $D_i \leq D_j$, produces a cluster τ_{ij} with the following parameters:*

$C_{ij} = C_i + C_j$

$T_{ij} = T_i = T_j$

$D_{ij} = D_i$

The cluster deadline is the shortest of the two tasks. Taking the minimum deadline ensures we respect both initial deadlines, even though the constraints will be, in general, more stringent than the initial constraints. By definition, we only group tasks with identical periods.

Definition 4. *Let $\mathcal{S} = (\{\tau_i\}_{1 \leq i \leq n})$ be a task set and τ_x and τ_y be two tasks of $\mathcal{S}$ such that $D_x \leq D_y$. We say that τ_{xy} is a* valid cluster *if and only if:*

1. $L_x \geq C_y$
2. *The task set obtained after clustering is schedulable*

In industrial practices, functionalities of different periods are sometimes mapped together, especially when these functionalities interact a lot, to minimize communication as explained in [28]. This possibility makes the clustering more complex because it requires to manage scheduling inside a cluster. For this reason, we do not deal with this option in this paper. Nevertheless, we could relax this assumption via, e.g., hierarchical scheduling [17].

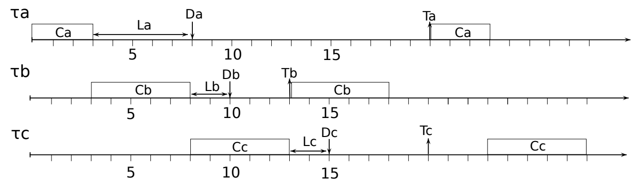
The laxity test is just an optimization. It is redundant with the schedulability test but it is simpler to check (constant time). Laxity is depicted in Subfigure 2(a).

A schedulable system might become non schedulable after clustering, as illustrated in Figure 2. Indeed, we notice in Subfigure 2(b) that the task τ_b misses its first deadline after the clustering of tasks τ_a and τ_c. Thus, we must check the resulting task set schedulability after clustering.

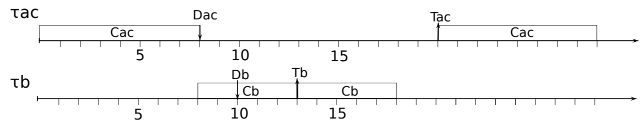
3 Checking cluster schedulability

Conditions 1 of the Definition 4 can be checked trivially in constant time. Nevertheless, condition 2 is more complex. Indeed, as we intend to check schedulability of a large number of solutions (i.e. at each step of the clustering process), considering a suitable schedulability test is important.

A schedulability test is called sufficient if all task sets considered schedulable by the test are actually schedulable. In the same manner, a schedulability test is called



(a) Initial schedulable system of tasks τ_a,τ_b and τ_c under DM.



(b) Resulting unschedulable system after clustering of tasks τ_a and τ_c.

Figure 2: Influence of task clustering on system schedulability.

necessary if all task sets considered unschedulable by the test are in fact unschedulable. Schedulability tests that are both sufficient and necessary are referred to as exact.

In this section, we review existing schedulability tests that can be used for clustering under DM and EDF scheduling policies. We only consider exact or sufficient tests insuring that the task sets obtained after clustering are schedulable. Indeed, applying sufficient tests means that we might not get the minimum number of clusters but we are sure to obtain a valid clustering. Notice that we work with synchronous (with offsets equal to zero) task sets that have constrained deadlines (i.e. with $D_i \leq T_i$).

3.1 Exact schedulability tests

Authors in [9] distinguishe two types of tests: *Boolean schedulability tests* and *response time tests*. On the one hand, Boolean tests give a Boolean answer, determining only whether a task set is schedulable or not, for instance with processor demand analysis (PDA) as the Quick convergence Processor-demand Analysis (QPA) [31]. On the other hand, exact tests based on response time analysis (RTA) provide worst response time for each task. The response time of a task is the time elapsed between its release and the time when it finishes its job.

Deadline Monotonic RTA [14, 3] of a task τ_i is based on the concept of level-i *busy period*. The level-i busy period is the maximum continuous time interval during which a processor executes tasks of higher or equal priority to the priority of the considered task τ_i, until τ_i finishes its active job. Then, the computation of the worst response time for each task τ_i is based on the length of level-i busy period. RTA for DM can be performed with a pseudo-polynomial time algorithm.

Earliest Deadline First Contrary to fixed-task priority (FP) systems, the worst response time is not necessarily found on the first processor busy period in a task set scheduled by EDF [30]. Thus, computing RTA for EDF is more complex and has an exponential complexity.

3.2 Sufficient schedulability conditions

In order to reduce the complexity of the computations, we also considered linear sufficient schedulability tests. Audsley [4] and Devi [10] propose sufficient but not necessary schedulability tests, respectively for DM and EDF in $\mathcal{O}(n)$ complexity. As far as we know, there are no more efficient tests for DM and EDF in linear complexity. The first results show that the test for DM behaves well for clustering and better than that of EDF. Those two sufficient tests actually provide an approximate worst response time for each task. They can be considered an approximate RTA analysis.

4 Minimizing the number of tasks

In this section, we detail our approach for minimizing the size of the initial task set by successive clusterings. Due to size of the search space, we rely on a heuristic instead of an exact algorithm.

4.1 Search space

Our problem consists in finding a partition of the task set that is schedulable and with a minimum number of subsets. A partition of a set $\mathcal{X}$ is a set of nonempty subsets of $\mathcal{X}$ such that every element n in $\mathcal{X}$ is in exactly one of these subsets. The number of partitions of a set is the Bell number [24]. The Bell number is exponential with respect to the size of $\mathcal{X}$ and can be computed by the following recurrence relation:

$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$ with $B_0 = 1$

As we only cluster tasks with identical periods, the search space can be restricted to $\prod_{i=0}^{m} B_{n_i}$ where B_{n_i} is the Bell number of set i of n tasks with equal periods and m is the number of sets. Nevertheless, this number remains exponential. To give a better idea of the size of the search, notice that for instance, $B_{500} \simeq 10^{844}$.

4.2 Partitions enumeration

A naive solution might be to conduct an exhaustive search among all partitions of the initial task set, e.g. by applying partitions generation algorithms [2, 20]. Nonetheless, our first experimentations show that, even using sufficient linear tests, this solution is not achievable due to the exponential number of partitions to explore. For instance, experiments conducted on a 2.3GHz Intel Core i7 quad-core with 4GByte memory, from an initial set of 20 tasks, lead to more than several days of computation. Thus, we think that it is necessary to limit the search space by applying a heuristic.

Our technique is derived from a simple recursive method found in Section 17.1 of [2]. For instance, for the set $\{\{A\},\{B\},\{C\}\}$ we generate the following 3 partitions in a first step:

$\{\{A\},\{B,C\}\}$
$\{\{A,C\},\{B\}\}$
$\{\{A,B\},\{C\}\}$

We apply recursively this principle for each partition generated until we obtain a partition with a unique element. This situation corresponds to having all tasks regrouped in a single cluster. This enumeration produces a tree as illustrated in Figure 3. Notice that this recursive algorithm generates many duplicates. For example, we can observe in the Figure 3 that the partition $\{\{A\},\{B,C,D\}\}$ appears twice. However, our heuristic always selects a single child by recursive call so we do not encounter duplicates.

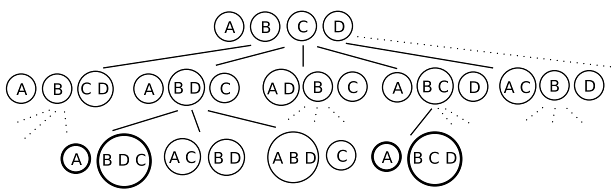


Figure 3: Recursive generation of partitions.

4.3 Heuristic

We start from an initial task set where each task is considered a cluster with one element, we gradually try to group more and more clusters together to minimize the cardinality of the task set. At each step, we try to group one cluster with another and we have, several candidates that fullfilled conditions 1 and 2. As some possiblities are better than the others, we must select the best candidate. This can be achieved by a heuristic cost (or evaluation) function that estimates which candidate will the most likely lead to the best clustering. We propose to achieve task clustering using classic heuristics based on cost functions.

4.3.1 Cost functions

We need a schedulability test to determine a valid task clustering because grouping tasks makes the resulting task set more and more difficult to schedule. Moreover, we need a relevant heuristic cost function to determine the best candidate for the clustering. We want a schedulability test that exhibits some features that might allow us to compare the potential of two task sets. Therefore, in this section, we explore the compatibility of the tests presented in Section 3 with a heuristic based on a cost function.

Boolean exact tests only give a Boolean answer on the schedulability of a task set. Thus, they do not exhibit any clear feature that could be considered a heuristic cost function.

On the one hand, exact tests based on RTA gives worst response times for each task. On the other hand, sufficient tests for DM and EDF presented below are based on a pessimistic approximation of the RTA. Considering a task τ_k with its worst response time denoted R_k, the closer to 1 $\frac{R_k}{D_k}$ is, the less we have margin to group the task τ_k with another. Thus, we can use the sum of each task response time divided by its respective deadline as heuristic cost function in both cases. Then, we have a heuristic cost function $h(\mathcal{S})$, such that

$$h(\mathcal{S}) = \sum_{k=0}^{|\mathcal{S}|} \frac{R_k}{D_k}$$

The RTA for EDF has an exponential complexity and experiments show that the test is not practicable (it takes more than several days of computation for 20 tasks). However, even though the RTA for FP has a pseudo-polynomial complexity, experiments show that run-time is not significantly slower than run-time with sufficient test under DM when the former gives an exact answer.

As a consequence, we can use the exact test based on RTA for DM and the sufficient test for EDF to achieve the best task clustering possible in a reasonable time.

4.3.2 Algorithm

Several heuristics based on a cost function exist such as greedy best-first search (greedy BFS), A* algorithm, simulated annealing, etc. We do not aim in this paper at comparing their different performances but at proposing a tractable solution. We moved towards a heuristic based on greedy BFS [25] detailed in Algorithm 1. The choice of the heuristic (as BFS here) is not central in this work. The main idea is the heuristic cost function that may also be applied with other heuristics, as those cited above. In this algorithm, we recursively enumerate partitions as explained in Section 4.2. At each recursive call, we choose the most promising local child (partition generated as in Section 4.2) according to a heuristic cost function as those presented in Section 4.3.1

Lemma 1. *The complexity of Algorithm 1 with linear tests is $\mathcal{O}(n^4)$ and pseudo-polynomial with pseudo-polynomial tests (RTA for DM).*

Proof. The number of children (or direct successors) generated by the technique described in Section 4.2 from a partition of i elements is equal to $i \times (i-1)/2$. We only explore one among all visited children at each step with our greedy heuristic. Thus, the maximum number of visited partitions is equal to $\sum_{i=0}^{n} \frac{i \times (i-1)}{2}$. This sum corresponds to the sum of the first n triangular numbers (also called tetrahedral numbers) and its closed-form expression is $f(n) = \frac{n(n+1)(n+2)}{6}$ [29]. Hence, this sequence complexity is $\mathcal{O}(n^3)$. We apply a sufficient schedulability test in $\mathcal{O}(n)$ complexity (whether with DM or EDF) on each visited partition, so the heuristic complexity is $\mathcal{O}(n^3) \times \mathcal{O}(n) = \mathcal{O}(n^4)$. In a similar way, applying schedulability tests with a pseudo-polynomial complexity gives a pseudo-polynomial complexity to the whole algorithm. □

Algorithm 1 Automated task clustering algorithm
Function clustering(S)

Require: $\mathcal{S} = (\{\tau_i\}_{1 \le i \le n})$: initial set of tasks in ascending deadline order

```
minSumTests ← n + 1
minSet ← null
for i = n − 1 to 0 do //find the best child
  for j = i − 1 to 0 do
    if T_i == T_j then
      if C_i + C_j ≤ min(D_i, D_j) then //laxity
        S' ← {S \ {τ_i, τ_j}} ∪ τ_ij
        if schedulable(S') then
          if h(S) < minSumTests then
            minSumTests ← h(S)
            minSet ← S'
          end if
        end if
      end if
    end if
  end for
end for

if minSet ≠ null then
  return clustering(minSet) //continue with best child
else
  return S
end if
```

5 Experimental results

5.1 Task set generation

We chose the following model to generate random task sets:

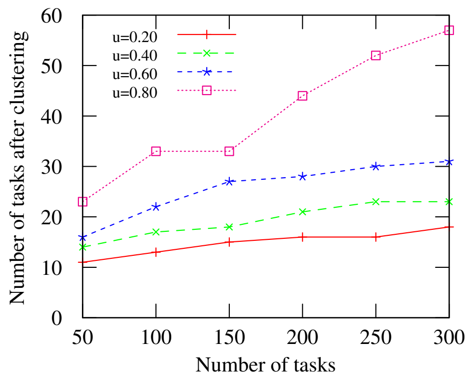
- U_i: each task utilization ($\frac{C_i}{T_i}$) is computed following the classic UUnifast [6] method. We denote as u the overall utilization factor of the processor.
- T_i: each task period is uniformly distributed between a set of 10 coprime periods. We observed that in industrial real-time embedded systems, the number of different tasks periods is usually limited (most often less than 10).
- $C_i = T_i \times U_i$
- $D_i = round((T_i - C_i) \times rand(d1, d2)) + C_i$ with $0 \le d1 \le d2$. This computation comes from [12] and use the following functions: $rand(d1, d2)$ which returns a pseudo-random real number uniformly distributed in the interval $[d1, d2]$ and $round(x)$ which returns the closest integer to x. We notice that $d1 = d2 = 1$ corresponds to implicit deadlines and $d1 \le d2 = 1$ to constrained deadlines.

5.2 Results

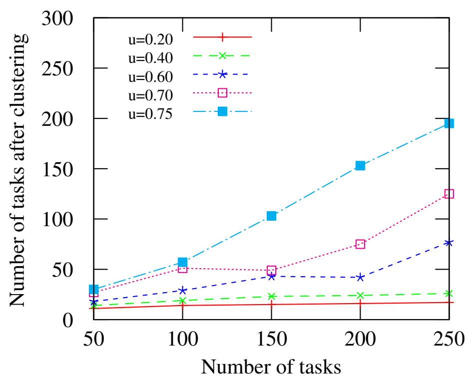
Unfortunately, as mentioned in Section 4.2, we cannot compare our heuristic with an optimal solution because the task clustering is not achievable with an exhaustive search among all partitions. Instead, we study how our heuristic behaves with various task set parameters (for example, deadline bounds).

We have implemented the heuristic in Scala [19]. Task sets range from 50 to 300 tasks by step of 50 tasks. Maximum utilization factor is fixed at 0.80 for DM and at 0.75 for EDF. Indeed, our tests show that there are only few schedulable task sets (according to the tests used) generated above those values. We only take into account task sets that are initially schedulable. We compute average results by executing several times the heuristic on randomly generated task sets with the same parameters.

We observe in Figure 4(a) that the technique is efficient under DM. Indeed, the number of tasks obtained after clustering is approximately linear in the number of tasks



(a) Task clustering under DM.

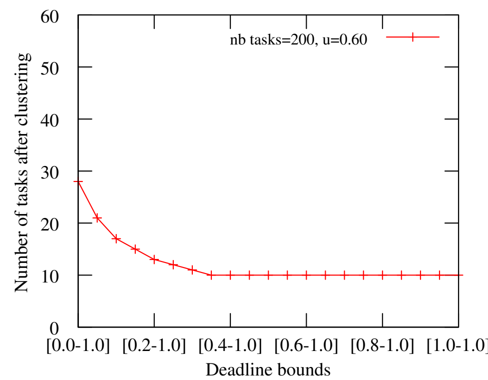


(b) Task clustering under EDF.

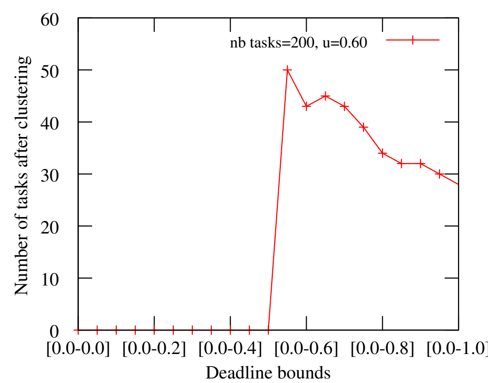
Figure 4: Results of task clustering.

and the slope of the curve is rather limited. However, results under EDF test in Figure 4(b) are not as satisfying. Clustering is less efficient, especially when the utilization goes over 0.6. This difference probably comes from the fact that the clustering affects more the test under EDF than the test under DM. Finally notice that, the higher the utilization factor is, the less the tasks are clustered.

Figure 5(a) and Figure 5(b) present the clustering depending on deadline bound variations with DM. For instance, $[0.4-1.0]$ on the horizontal axis means that the deadline is chosen between 40% and 100% of the period minus the execution time. We can see in Figure 5(a) that the number of clusters is minimal (equal to the number of different periods) when the deadline lower bound is about 40% of the period minus the execution time. Figure 5(b) shows that no clustering is possible before the upper bound gets to around 60%. Above that bound, the efficiency of the clustering improves steadily (the number of clusters decreases).



(a) Variation of deadline lower bound.

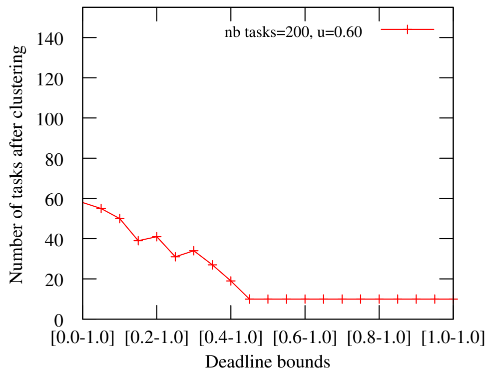


(b) Variation of deadline upper bound.

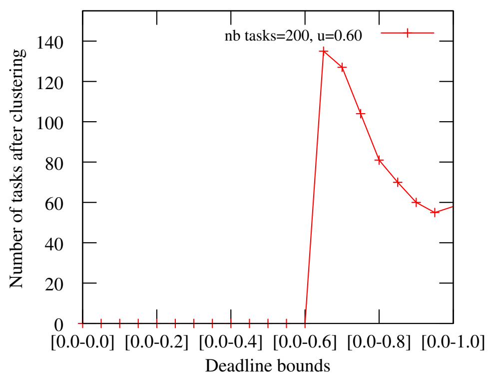
Figure 5: Task clustering with DM: impact of deadline bounds.

Figure 6(a) and Figure 6(b) present the clustering for the same deadline variations with EDF. The overall trends of the curves are similar, though the clustering is overall less efficient.

These results show that the deadline bounds have the most significant impact on the clustering (even more than the number of tasks for DM). Both with DM and EDF, the clustering is the most efficient with deadlines bounds



(a) Variation of deadline lower bound.



(b) Variation of deadline upper bound.

Figure 6: Task clustering with EDF: impact of deadline bounds.

in the interval [0.5,1]. Indeed, the closer deadlines are to the period, the more margin is left for the clustering. The clustering is even maximal in that interval because we get as many tasks as the number of different periods, both for DM and EDF. Notice that according to further experiments, this remains true for a higher number of distinct periods.

6 Conclusion and future work

We proposed a heuristic to automatically reduce a large set of independent tasks to a smaller set, while preserving the schedulability of the task set. The current assumption that tasks are independent will be lifted in future work. The present work is meant to lay the foundations of automated task clustering, which, as far as we know, has not been studied formally before.

Experimental results point out that under some ranges of deadline bounds, the clusterings are maximal (i.e. the number of tasks equals the number of periods). As these ranges are actually realistic, it would be interesting to try to formally prove that we can always reach maximal clusterings for these bounds. Such a property would allow to directly gather all the tasks with the same periods without using any clustering algorithm.

References

[1] A. Ahmadinia, C. Bobda, and J. Teich. Temporal task clustering for online placement on reconfigurable hardware. In *Field-Programmable Technology (FPT), 2003. Proceedings. 2003 IEEE International Conference on*, pages 359 – 362, Dec. 2003.

[2] J. Arndt. *Matters Computational: Ideas, Algorithms, Source Code*. Springer, 2010.

[3] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[4] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. *Deadline monotonic scheduling*. 1990.

[5] AUTOSAR. *RTE Standard Specifications*.

[6] E. Bini and G. Buttazzo. Biasing effects in schedulability measures. In *16th Euromicro Conference on Real-Time Systems, 2004. ECRTS 2004. Proceedings*, pages 196 – 203, July 2004.

[7] F. Boniol, P.-E. Hladik, C. Pagetti, F. Aspro, and V. Jégu. A framework for distributing real-time functions. In *Proceedings of the 6th international conference on Formal Modeling and Analysis of Timed Systems*, FORMATS '08, pages 155–169. Springer-Verlag, 2008.

[8] A. Curic. *Implementing Lustre Programs on Distributed Platforms with Real-time Constrains*. PhD thesis, University Joseph Fourier, Grenoble, 2005.

[9] R. I. Davis, A. Zabos, and A. Burns. Efficient exact schedulability tests for fixed priority real-time systems. *Computers, IEEE Transactions on*, 57(9):1261–1276, 2008.

[10] U. Devi. An improved schedulability test for uniprocessor periodic task systems. In *Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on*, pages 23 – 30, july 2003.

[11] J. Forget. *A Synchronous Language for Critical Embedded Systems with Multiple Real-Time Constraints*. PhD thesis, Université de Toulouse, 2009.

[12] J. Goossens and C. Macq. Limitation of the hyperperiod in real-time periodic task set generation. In *In Proceedings of the RTS Embedded System (RTS'01*, pages 133–147, 2001.

[13] L. Guodong, C. Daoxu, W. Daming, and Z. Defu. Task clustering and scheduling to multiprocessors with duplication. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, page 8 pp., Apr. 2003.

[14] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[15] E. Lee. The problem with threads. *Computer*, 39(5):33–42, 2006.

[16] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance evaluation*, 2(4):237–250, 1982.

[17] G. Lipari and E. Bini. A methodology for designing hierarchical scheduling systems. *Journal of Embedded Computing*, 1(2):257–269, 2005.

[18] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, January 1973.

[19] M. Odersky, L. Spoon, and B. Venners. *Programming in Scala, 2/e*. Artima Series. Artima Press, 2010.

[20] M. Orlov. Efficient generation of set partitions. *Engineering and Computer Sciences, University of Ulm, Tech. Rep*, 2002.

[21] C. Pagetti, J. Forget, F. Boniol, M. Cordovilla, and D. Lesens. Multi-task implementation of multi-periodic synchronous programs. *Discrete Event Dynamic Systems*, 21(3):307–338, 2011.

[22] M. Palis, J.-C. Liou, and D. Wei. Task clustering and scheduling for distributed memory parallel architectures. *Parallel and Distributed Systems, IEEE Transactions on*, 7(1):46 –55, Jan. 1996.

[23] K. Ramamritham. Allocation and scheduling of precedence-related periodic tasks. *IEEE Trans. Parallel Distrib. Syst.*, 6(4):412–420, Apr. 1995.

[24] G.-C. Rota. The number of partitions of a set. *The American Mathematical Monthly*, 71(5):498–504, 1964.

[25] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*, pages 94–95. Prentice Hall, 2 edition, 2003.

[26] L. Santinelli, W. Puffitsch, C. Pagetti, and F. Boniol. Scheduling with functional and non-functional requirements: the sub-functional approach. *Work-in-Progress Session of ECRTS 2013*, 2:9, 2013.

[27] O. Scheickl and M. Rudorfer. Automotive real time development using a timing-augmented AUTOSAR specification. *Proceedings of ERTS2008*, 4, 2008.

[28] S. Schliecker, J. Rox, M. Negrean, K. Richter, M. Jersak, and R. Ernst. System level performance analysis for real-time automotive multicore and network architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(7):979 –992, July 2009.

[29] N. J. A. Sloane. The On-Line Encyclopedia of Integer Sequences. A000292.

[30] M. Spuri. Analysis of Deadline Scheduled Real-Time Systems. Research report RR-2772, INRIA, 1996. REFLECS Project.

[31] F. Zhang and A. Burns. Schedulability analysis for real-time systems with edf scheduling. *Computers, IEEE Transactions on*, 58(9):1250–1258, 2009.

[32] M. Zhang and Z. Gu. Optimization issues in mapping AUTOSAR components to distributed multi-threaded implementations. In *2011 22nd IEEE International Symposium on Rapid System Prototyping (RSP)*, pages 23 –29, May 2011.