Improvement of schedulability bound by task splitting in partitioning scheduling
Frédéric Faubertea, Serge Midonnet, Laurent George

To cite this version:
Frédéric Faubertea, Serge Midonnet, Laurent George. Improvement of schedulability bound by task splitting in partitioning scheduling. 1st International Real-Time Scheduling Open Problems Seminar (RTSOPS’10), Jul 2010, Brussels, Belgium. pp.20-21. hal-00620347

HAL Id: hal-00620347
https://hal-upec-upem.archives-ouvertes.fr/hal-00620347
Submitted on 19 Mar 2013

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.
Improvement of schedulability bound by task splitting in partitioning scheduling

Frédéric Fauberteau  
Université Paris-Est  
LIGM, UMR CNRS 8049  
Email: fauberte@univ-mlv.fr

Serge Midonnet  
Université Paris-Est  
LIGM, UMR CNRS 8049  
Email: midonnet@univ-mlv.fr

Laurent George  
ECE, LACSC  
37, quai de Grenelle, 75015 Paris, France  
Email: lgeorge@ieee.org

Abstract

We focus on the class of static-priority partitioning scheduling algorithm on multiprocessor. We are interested in improving the schedulability of these algorithms by splitting the tasks which cannot be successfully allocated on processors.

1. Context

Unless \( P = NP \), no polynomial time algorithm exists to solve the MULTIPROCESSOR-TASK-PARTITIONING problem. Indeed, this problem can be transformed from BIN-PACKING problem which is NP-hard in the strong sense. Fortunately, there are several heuristics which solve BIN-PACKING problem and they may be adapted to produce partitioned schedulings. For instance, the algorithm FBB-FDD is based on First-Fit Decreasing [1] and the algorithm RM-DU-NFS is based on Next-Fit [2].

We are interested in proposing a partitioning algorithm which is robust to task Worst Case Execution Time (WCET) overruns faults with the same objective as in [3]. This partitioning tends to maximize the capacity of the system to handle the WCET overruns by offering the maximum Allowance to each faulty task of the system. The Allowance of a task is the execution duration which can be added to its WCET such that all the deadlines in the system are met [4].

We have shown in [5] that the Worst-Fit heuristic offers good results in order to maximize the robustness of a partitioned system. Worst-Fit selects the least loaded processor (in terms of utilization) which can accept a task. Unfortunately, the performances of Worst-Fit in terms of schedulability are less efficient than those of the heuristics widely used in partitioning algorithms (First-Fit Decreasing/Best-Fit/Next-Fit). In order to improve the schedulability bound of Worst-Fit and potentially those of the other heuristics, we propose to split the tasks which cannot be allocated on a single processor. Contrary to the portioned scheduling approach proposed in [6] and [7], we don’t migrate a job from a processor to another during its execution but we allocate jobs on different processors such that the migrations occur at job boundaries [8].

2. Task splitting approach

Description. Let \( \Pi \) be a multiprocessor composed by \( m \) identical processors denoted by \( \Pi = \{\pi_1, \ldots, \pi_m\} \). Let \( \tau \) be a set of tasks made of \( n \) periodic (or sporadic) real-time tasks denoted \( \tau = \{\tau_1, \ldots, \tau_n\} \). Each task is characterized by its WCET, its deadline and its period (or minimum interarrival time). We consider a partitioning algorithm denoted \( A \) which allocates the tasks of \( \tau \) on \( \Pi \). \( A \) is designed as follows:
- \( \tau \) is sorted according to a decreasing order policy (for instance Decreasing Utilization),
- a heuristic chooses the processor on which a given task should be allocated,
- a schedulability test is used to decide whether the task can be allocated on the processor chosen by the heuristic.

The resulting algorithm \( A \) produces a partition of \( \tau \) on \( \Pi \). If \( A \) fails to allocate the task \( \tau_i \), the set of pending tasks \( \{\tau_1, \ldots, \tau_n\} \) is unallocated.

In order to schedule these unallocated tasks, two approaches can be considered: portioned scheduling and scheduling at job boundaries (i.e. restricted migration). To illustrate the first one, we show in Figure 1 four tasks...
We consider the case where a partitioning algorithm doesn’t succeed to partition the set $\tau$ of tasks. Our aim is to build an algorithm which can split the pending tasks in subtasks such that a feasible partition of $\tau$ can be found. If no pending task can be split, already allocated tasks can be split until a feasible partition is found. The problem is to build an algorithm which finds a valid splitting scheme if such a scheme exists such that a feasible partition can be found.

Open problem. We consider the case where a partitioning algorithm doesn’t succeed to partition the set $\tau$ of tasks. Our aim is to build an algorithm which can split the pending tasks in subtasks such that a feasible partition of $\tau$ can be found. If no pending task can be split, already allocated tasks can be split until a feasible partition is found. The problem is to build an algorithm which finds a valid splitting scheme if such a scheme exists such that a feasible partition can be found.

Insights. We want to build an algorithm which splits the tasks such that the migration occurs at job boundaries. A first approach is to build a static-priority restricted migration scheduler. This scheduler allocates jobs of the tasks when they are activated on a ready processor according to a static-priority scheme. By logging the allocations made by this scheduler, we can build our splitting scheme. Notice that, this approach can be complex because it may require the consideration of an interval of study which can be exponential. Moreover, it is not obvious to decide on which processor a task must be activated to minimize the number of subtasks.

References


