An off-line variable-size bin packing for EDF

Alejandro Masrur Sebastian Drössler Georg Färber
Institute for Real–Time Computer Systems
Technische Universität München, Germany
{Alejandro.Masrur, Sebastian.Droessler, Georg.Faerber}@rcs.ei.tum.de

Abstract

This paper presents a bin-packing algorithm for the allocation of tasks to identical processors. For this purpose, we consider the case of independent periodic real-time tasks scheduled under EDF, whose deadlines may not be equal to periods. The presented bin-packing algorithm combines the feasibility condition of Devi [5] with the First Fit bin-packing heuristic. The result is an off-line variable-size bin packing which might deliver better results, as shown with an example, than the First Fit Decreasing heuristic on the base of the density condition. Finally, we discuss some interesting issues that rise in relation to the presented algorithm.

1 Introduction

In this paper, we focus on determining the number of processors needed for a multiprocessor system to be feasible. For this purpose, we make use of bin-packing algorithms, which allocate tasks sequentially according to a given heuristic criterion. There are many possible bin-packing heuristics for allocating tasks to processors. Among the most popular ones, we can mention the First Fit (FF) and First Fit Decreasing (FFD) heuristics.

The FF heuristic consists in assigning tasks to the first available processor that can guarantee their correct executing, i.e. without time overflow. Those processors, whose capacity is still not completely assigned, will be considered as available. These are verified in order of increasing index; if none of them is able to feasibly execute the task being currently allocated, a new processor will be necessary. This process is repeated until all tasks are allocated.

A bin-packing algorithm based on the FF heuristic is referred as an on-line algorithm. This is because it does not require any previous knowledge on tasks that are to be allocated. In contrast, the FFD heuristic is an off-line heuristic, because it does require previous knowledge on tasks. The only difference between FF and FFD is that in the latter, tasks must be sorted in non-increasing order of processor demand, i.e. all tasks must be previously known.

The first possible approach for bettering these algorithms is to improve the feasibility test used for the allocation procedure. Other possibility is to improve the allocation heuristic itself. In this paper, we explore the first possibility for tasks scheduled under EDF.

There are different ways to perform a feasibility test for for real-time tasks scheduled under EDF. In [9], Liu proposes the following feasibility condition: $\sum_{i=1}^{n} \frac{e_i}{min\{p_i,d_i\}} \leq 1$. Where d_i is the relative deadline of the i-th task, p_i represents its period, e_i the corresponding worst-case execution demand, and n the number of tasks. Only if $d_i \geq p_i$ for all $i \leq n$, this inequality, known as *density condition*, will constitute a necessary and sufficient condition for the feasibility of tasks scheduled under EDF [3].

If any deadline is less than the period of the corresponding task, this inequality constitutes a sufficient but not necessary condition. Another sufficient but not necessary condition was presented by Devi in [5] for the case where d_i can also be less than p_i for any $i \leq n$. Devi's condition was proven to be tighter than the density condition.

Finally, more complex algorithms will be necessary to perform an exact feasibility test when d_i is also allowed to be less than p_i [3]. Nevertheless, for the reason that allocating real-time tasks on multiprocessor systems is an NP-hard problem [6], it is preferable to use sufficient but not necessary feasibility conditions than exact feasibility tests. Otherwise the resulting allocation algorithm will be impractical, even if the allocation itself is performed by an approximation algorithm like a bin-packing algorithm.

Generally, when allocating real-time tasks scheduled under EDF on multiprocessor systems, most approaches force all deadlines to be equal to the corresponding periods. In this way, the density condition becomes sufficient and necessary and only the allocation heuristic itself would be eligible to be bettered.

On the other hand, if $d_i < p_i$ holds for any task T_i, Devi's condition outperforms this latter [5]. Consequently, it would be logical to use Devi's condition instead of the density condition for allocating tasks to processors. Unfortunately, Devi's condition cannot be directly implemented into a bin-packing algorithm and some manipulation will be necessary.

2 Task model and notation

For the purpose of this paper, we make use of the periodic task model of Liu and Layland [8]. That is, we consider a set τ of periodic real-time tasks, which are fully preemptable and independent. Like in [8], we assume that no task may suspend itself and that all overhead, caused by the scheduler and context switches, is negligible.

Each task T_i is characterized by its period of repetition p_i, its relative deadline d_i and its worst-case execution demand e_i. As already mentioned, relative deadlines are not forced to be equal to corresponding task periods, $d_i \leq p_i$ as well as $d_i > p_i$ may hold for all $i \leq n$, where n is the number of tasks in τ. The processor demand of any task T_i is given by $u_i = \frac{e_i}{p_i}$. The so-called density of task T_i will be given by $\frac{e_i}{min\{d_i,p_i\}}$. We assume that $\frac{e_i}{min\{d_i,p_i\}} \leq 1$ holds for every task T_i in τ, otherwise T_i itself will be infeasible.

If k tasks are allocated to a processor P_m the processor utilization of P_m will be given by $U_m = \sum_{i=1}^{k} u_i$. As tasks are scheduled under EDF, which was proven to be optimal [4], U_m cannot be greater than 1 for all possible m—otherwise the allocation is not feasible.

Finally, we consider processors to be identical, i.e. all processor have the same computing capacity. Consequently, a task T_i will take the same time to run independently of the processor on which it runs.

3 Preliminary discussion

The feasibility condition presented by Devi in [5] consists in verifying the following inequality for all k for which $1 \leq k \leq n$ holds:

$$\sum_{i=1}^{k} \frac{e_i}{p_i} + \frac{1}{d_k} \sum_{i=1}^{k} \left(\frac{p_i - min\{p_i, d_i\}}{p_i} \right) \cdot e_i \leq 1$$

Devi's condition requires all tasks to be sorted in non-decreasing order of relative deadlines. So if the index i is less than j, the relative deadline d_i will be less than or equal to d_j. This presorting requisite makes Devi's condition ineligible for on-line allocation algorithm, however, it makes possible to design an off-line algorithm on its base.

Reordering Devi's inequality and replacing the subindex k for $last$, we get the following expression:

$$\frac{1}{d_{last}} \sum_{i=1}^{last} \left(\frac{p_i - min\{p_i, d_i\}}{p_i} \right) \cdot e_i \leq 1 - \sum_{i=1}^{last} \frac{e_i}{p_i} \qquad (1)$$

For the purpose of simplifying next expressions, we represent the left and the right member of inequality 1 by L and C respectively. Consequently, we can rewrite inequality 1 as follows:

$$L_m \leq C_m \qquad (2)$$

Where the index m indicates the *bin* in the bin-packing terminology, i.e. the processor where tasks are allocated. For every processor P_m, whose capacity has still not been completely allocated, it is necessary to keep the values L_m, C_m, and d^m_{last}. This latter represents the deadline of the last task that was allocated to processor P_m. A processor, whose capacity has still not been fully allocated, will be called *open processor*.

4 The allocation algorithm

We term by T_{next} the task that is to be allocated as next. As we make use of the FF strategy for the packing algorithm, T_{next} will be allocated to the processor with the lowest index that can execute it without missing any deadline. So all open processors will be inspected in increasing order of index and T_{next} will be allocated to the first one of them for which the following inequality holds:

$$\frac{L_m \cdot d^m_{last} + \rho_{next} \cdot u_{next}}{d_{next}} \leq C_m - u_{next} \qquad (3)$$

Where $\rho_{next} = p_{next} - min\{p_{next}, d_{next}\}$. If inequality 3 holds for processor P_m, the processor's parameters must be updated as follows:

$$\begin{aligned} C_m &:= C_m - u_{next} \\ L_m &:= \frac{L_m \cdot d^m_{last} + \rho_{next} \cdot u_{next}}{d_{next}} \\ d^m_{last} &:= d_{next} \end{aligned}$$

Otherwise, if inequality 3 does not hold for any open processor or there is no open processor at the time, a new processor will be necessary. Each time a new processor is added to the list of open processors, its parameters are set according to the following:

$$\begin{aligned} C_m &:= 1 - u_{next} \\ L_m &:= \frac{\rho_{next} \cdot u_{next}}{d_{next}} \\ d^m_{last} &:= d_{next} \end{aligned}$$

A processor will be *closed*, i.e. deleted from the open processors' list, whenever the right and the left member of inequality 3 are equal.

Inequality 3 can also be reordered as follows, where $\sum_{i=1}^{last} u_i$ is processor utilization of P_m, i.e. only tasks allocated to P_m are summed up:

$$\sum_{i=1}^{last} u_i + u_{next} \leq 1 - \frac{L_m \cdot d_{last}^m + \rho_{next} \cdot u_{next}}{d_{next}} \quad (4)$$

As the right member of inequality 4 varies (decreases) depending on T_{next}'s parameters, the resulting allocation algorithm is a variable-size bin-packing algorithm. A similar phenomenon can be also observed for fixed-priority schedulings [9].

Figure 1 shows the pseudo code for the allocation algorithm on the base of Devi's condition. The algorithm's output is a listing of processors with the corresponding tasks that were allocated to them. This latter was not explicitly implemented in pseudo code, but it does not present a major difficulty.

4.1 Complexity

Because of our assumption $\frac{e_i}{min\{p_i,d_i\}} \leq 1 \forall i \leq n$, it is possible for the needed number of processors m to be equal to but not greater than the number of tasks n. Consequently, the complexity of the algorithm from figure 1 is $O(n^2 \cdot log \quad n)$. A FF bin-packing algorithm based on the density condition will present a complexity $O(n^2)$. Recall that we pursue to determine an as small as possible number of processors that guarantees the feasibility, i.e. the number of processors is not fixed. On the other hand, if we were to determine if the task set is feasible on a fixed number of processors, the resulting complexity would be $O(n)$ for FF based on the density condition and $O(n \cdot log \quad n)$ for FF based on Devi's condition.

Applying Devi's condition together with the FFD bin-packing heuristic, where tasks are sorted by non-increasing order of density, will result in a complexity with the form $O(n^3 \cdot log \quad n)$. This latter is because sorting tasks according to non-decreasing deadlines might not result in the same sequence of tasks than sorting them by non-increasing density. As a consequence, each time T_{next} is tried to be allocated to an open processor P_m, all other task that were already allocated to P_m and whose deadlines are greater than d_{next} must be afresh verified.

4.2 Example

In this section, we show by means of an example that the algorithm from figure 1 may present a better allocation than the FFD algorithm based on the density condition.

In figure 2, an exemplary task set is sorted by non-increasing density. This corresponds the presorting requisite of FFD. For this case, the FFD packing needs three processors to feasibly allocate this task set. The resulting allocation is shown in figure 3.

```
for $i \leq \ 1$
 sort $T_i$ in non-decreasing order of $d_i$;
end

while SortedTasks $\neq \emptyset$
 get $T_{next}$;

 while OpenProcessors $\neq \emptyset$
  get $P_m$;

  if $\frac{N_m \cdot \ d_{last}^m + \rho_{next} \cdot \ u_{next}}{d_{next}} < C_m - u_{next}$
   update $P_m$'s parameters;

  elseif $\frac{N_m \ \cdot \ d_{last}^m + \rho_{next} \cdot \ u_{next}}{d_{next}} = C_m - u_{next}$
   close $P_m$;

  elseif last processor in OpenProcessors
   add new processor;

   if $\frac{\rho_{next} \cdot \ u_{next}}{d_{next}} = 1 - u_{next}$
    close $P_m$;
   end
  end

  delete $T_{next}$ from SortedTasks;
 end
end
```

Figure 1. Pseudo code

In contrast, the algorithm on the base of Devi's condition requires the presorting of figure 4. As shown in figure 5, this latter algorithm needs only two processors to feasibly allocate this exemplary task set.

$\mathbf{T_i}$	$\mathbf{e_i}$	$\mathbf{d_i}$	$\mathbf{p_i}$	$\mathbf{\frac{e_i}{min\{p_i,d_i\}}}$
T_1	7	10	20	0.7
T_2	2	5	8	0.4
T_3	2	5	10	0.4
T_4	1.9	7	11	0.27
T_5	3	20	30	0.15
T_6	6	40	50	0.15

Figure 2. Sorted by non-increasing density

5 Concluding remarks

We presented a bin-packing algorithm for allocating tasks scheduled under EDF to identical processors. This algorithm bases on Devi's condition and on the First Fit heuristic. Because of the presorting requisite of Devi's condition, combining this feasibility condition with the

Processor	***Allocated tasks***
P_1	T_1, T_4
P_2	T_2, T_3, T_5
P_3	T_6

Figure 3. FFD and density condition

$\mathbf{T_i}$	$\mathbf{e_i}$	$\mathbf{d_i}$	$\mathbf{p_i}$	$\frac{\mathbf{e_i}}{\mathbf{min\{p_i,d_i\}}}$
T_2	2	5	8	0.4
T_3	2	5	10	0.4
T_4	1.9	7	11	0.27
T_1	7	10	20	0.7
T_5	3	20	30	0.15
T_6	6	40	50	0.15

Figure 4. Sorted by non-decreasing deadline

FF bin-packing heuristic results in an off-line allocation algorithm—even if the FF heuristic was initially intended for on-line algorithms. The purpose of the presented algorithm is to find an as small as possible number of processors that guarantees the feasibility of a given task set. In other words, the number of processors is not a fixed parameter.

This bin-packing algorithm on the base of Devi's condition requires three parameters to be kept for each bin or processor. We called these parameters L_m, C_m and d^m_{last}. This differs from known bin-packing algorithms, where only the remaining capacity is needed to be kept for each processor or bin. Additionally, the size of bins, i.e. the available processor capacity, varies depending on T_{next}'s parameters, what leads to a variable-size bin packing. We have termed by T_{next} the task that is to be allocated as next.

The resulting complexity is degraded from $O(n^2)$ of the original FF bin packing to $O(n^2 \cdot log \quad n)$, where the number of processor is limited from above by n, i.e. by the number of tasks—this latter is because $\frac{e_i}{min\{p_i,d_i\}} \leq 1 \forall i \leq n$. Moreover, applying the FFD bin-packing strategy will result in a even higher complexity with the form $O(n^3 \cdot log \quad n)$. This latter is because it is not possible to guarantee that sorting tasks as required by Devi's condition also satisfies the presorting requisite of the FFD strategy.

The algorithm presented in this paper opens some issues. For example, it would be interesting to thoroughly compare the FFD bin-packing algorithm based on the density condition with the one presented in this paper. In principle, both algorithms have the same complexity and both are off-line algorithms. Nevertheless, the FFD bin packing is known to guarantee an asymptotic worst-case performance ratio $R^{\infty}_{FFD} \sim 1.22$, whereas the one of FF bin packing is known to be $R^{\infty}_{FF} \sim 1.7$ [7]. This performance ratio expresses how far from optimum the worst-case allocation is as the number of tasks tends to infinity. It would be interesting to analyze for which task configurations the FF strategy combined with Devi's feasibility condition delivers better results than the FFD strategy based on the density condition.

Processor	***Allocated tasks***
P_1	T_2, T_3, T_4
P_2	T_1, T_5, T_6

Figure 5. FF and Devi's condition

Finally, Albers and Slomka proved in [2] that Devi's condition is a particular case of their superposition approach presented in [1]. Albers and Slomka presented an approximate feasibility test in [1], which verifies the first k deadlines of each task in τ. This number k of deadlines that are verified can be adjusted. A small k results in a simpler but less accurate feasibility test. In contrast, if k is large, the resulting feasibility test is more complex but more accurate. The question is which benefits could be obtained implementing bin-packing algorithms on the base of this latter feasibility test. It would be also interesting to research into methods to adjust this trade-off parameter k for particular task sets.

References

[1] K. Albers and F. Slomka. An event stream driven approximation for the analysis of real–time systems. Proceedings of the 16th Euromicro Conference on Real–Time Systems, pages 187–195, June 2004.

[2] K. Albers and F. Slomka. Efficient feasibility analysis for real–time systems with edf scheduling. Proceedings of the Date 05 Conference, March 2005.

[3] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard–real–time sporadic tasks on one processor. Proceedings of the Real–Time Systems Symposium, pages 182–190, December 1990.

[4] M. Dertouzos. Control robotics: The procedural control of physical processes. Proceedings of the IFIP Congress, pages 807–813, 1974.

[5] M. Devi. An improved schedulability test for uniprocessor periodic task systems. Proceedings of the 15th Euromicro Conference on Real–Time Systems, 2003.

[6] M. Garey and D. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., 1979.

[7] D. Johnson. Near–Optimal Bin Packing Algorithms. PhD Thesis, Massachusetts Institue of Technology, USA, 1973.

[8] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in hard real–time environments. Journal of the Association for Computing Machinery, 20(1):40–61, 1973.

[9] J. Liu. Real–Time Systems. Prentice Hall, 2000.