Using Exact Feasibility Tests for Allocating Real-Time Tasks in Multiprocessor Systems*

Sergio Sáez, Joan Vila and Alfons Crespo
Universidad Politécnica de Valencia, DISCA
Camino de Vera 14, 46071 Valencia, Spain
Phone: +34 6 387 9577 Fax: +34 6 387 7579
{ssaez,jvila,alfons}@disca.upv.es

Abstract

This paper introduces improvements in partitioning schemes for multiprocessor real-time systems which allow higher processor utilization and enhanced schedulability by using exact feasibility tests to evaluate the schedulability limit of a processor. The paper analyzes how to combine these tests with existing bin-packing algorithms for processor allocation and provides new variants which are exhaustively evaluated using two assumptions: variable and fixed number of processors. The problem of evaluating this algorithms is complex, since metrics are usually based on comparing the performance of a given algorithm to an optimal one, but determining optimals is often NP-hard on multiprocessors. This problem has been overcome by defining lower or upper bounds on the performance of an optimal algorithm and then defining metrics with respect this bounds. The evaluation has shown that the algorithms exhibit extremely good behaviors and they can be considered very close to the maximum achievable utilization. It is also shown that dynamic priority policies produces significantly better results than fixed priorities policies when task sets require high processor utilizations.

1. Introduction

Real-time systems are one of the fields of computing where major benefits are expected from the increasing availability of multiprocessor technology. However, taking full advantage of multiprocessor's capabilities for real-time computing has shown to be difficult. This is mainly due to the fact that scheduling algorithms for multiprocessors are significantly more complex than for uniprocessors. Theoretical results in this area show that almost all real-time multiprocessing scheduling is NP-hard [5, 9]. For that reason, looking for optimals to solve the general case makes no sense and research rather focuses on solving some restricted cases or finding a good trade-off between performance and computational cost. In the last case, performance metrics and evaluation of algorithms play an important role.

There are two main approaches for scheduling real-time tasks on a multiprocessor system: global scheduling and partitioning of tasks. In the *global* scheme there is only one (multiprocessor) scheduler for all processors which assigns each occurrence of a task to a (possibly different) processor for execution. Some multiprocessor schedulers even allow tasks to be preempted and moved to a different processor before they complete. This is called *migration* and it is assumed to have very low cost on shared memory multiprocessors. In contrast, in the *partitioning* scheme tasks are pre-allocated to processors by a *task assignment algorithm* and all occurrences of a task are executed on the same processor. In this approach, every processor has its own *local scheduler* that determines the schedule for each processor using a monoprocessor scheduling policy. This paper concentrates on the analysis of heuristics for partitioning schemes. A similar analysis of global schemes has been previously presented in [8].

The goal of a partitioning scheme is to make an assignment of tasks to processors such that all tasks meet their deadlines. The partitioning scheme is mainly used with static workloads, that is, when the system consists of a set of tasks which are known in advance. That makes possible to make task assignment off-line. This approach has several advantages over the global scheme. On one hand, it has less run-time overhead since allocation is only done once per task (not on each occurrence of a task) and it can be made off-line. On the other hand, it is less complex since it is only required to provide a partitioning of the tasks; not a complete schedule.

*This work was supported by the Spanish Government Research Office (CICYT) under grant TAP97-1164-C03-03 and by Generalitat Valenciana under grant GV-C-CN-05-058-96

In partitioning schemes there is a clear distinction between the *task assignment algorithm* and the run-time *scheduling algorithm*. However, both algorithms are not completely independent, since the limit on the number of tasks that can be assigned to a given processor (processor utilization) is determined by the scheduling algorithm. Every time a new task has to be assigned to a processor a *schedulability test* has to be used to check if its timing requirements can be guaranteed on that processor. This test is completely dependent of the scheduling algorithm that is being used. Examples of such tests are the utilization test [6] and the completion time test [4] for fixed priority scheduling algorithms, and the test by Baruah *et al.* [1] or its optimization by Ripoll *et al.* [7] for dynamic priority scheduling algorithms.

The algorithms for task assignment considered in this paper are variants of well-known heuristics for solving the bin-packing problem. In this problem, each bin has a maximum capacity and boxes placed in the bins require some of this capacity. When the problem is translated into real-time scheduling terms, real-time tasks are regarded as a set of boxes while processors as a set of bins. The goal is either maximizing the number of tasks which can be scheduled with a given fixed number of processors, or minimizing the number of processors required to schedule a given set of tasks.

Much work on partitioning schemes deals with the problem of minimizing the number of processors to achieve a feasible schedule for a given set of tasks. There are some theoretical results [3, 2] that evaluate worst case bounds for $\lim_{N_{opt}\to\infty} N/N_{opt}$ where N_{opt} is the number of processors required by an optimal algorithm. However, these results are of limited value in real-time computing systems since they are only available as asymptotic bounds with very large number of processors ($N_{opt} \to \infty$).

This paper addresses two different analysis of bin-packing heuristics: fixed and variable number of processors. In the first analysis the main goal is to estimate the utilization bounds that can be achieved with different bin-packing algorithms using a fixed number of processors. In the second analysis the goal is to compare the number of processors required to schedule a task set, but considering a finite number of processors instead of asymptotic limits. That implies avoiding the concept of N_{opt} and defining new metrics to evaluate the algorithms.

Two common assumptions in previous analysis were the use of the Rate Monotonic (RM) algorithm as the underlying scheduling policy [2] and the test by Liu and Layland [6] (or some optimizations) to evaluate the maximum workload that can be assigned to each processor. These assumptions imply underestimating the schedulability limit of a processor because the above test is only a sufficient condition and leaves some unused processor time.

This paper extends previous results by considering deadline algorithms in their static and dynamic versions: *Deadline Monotonic* (DM) and the *Earliest Deadline First* (EDF) and tighter schedulability conditions to evaluate the schedulability limit of a processor. Results show that the use of these assumptions considerably improves the performance of the bin-packing algorithms and the overall system.

The remainder of this paper is structured as follows. Section 2 shows how to extend bin-packing algorithms for partitioning schemes using DM and EDF algorithms and exact feasibility test. Section 3 introduces new metrics for evaluating multiprocessor scheduling algorithms based on estimating bounds on optimals rather than in finding optimals which are often NP-hard. Section 4 presents several evaluations of bin-packing algorithms assuming variable and fixed number of processors. It also provides estimations on lower bounds on the minimum number of processors required to schedule a given set of tasks.

2. Using exact feasibility tests for allocating tasks to processors

The first goal we address in this paper is to maximize the number of tasks which can be scheduled with a fixed number of processors using different bin-packing algorithms for task assignment and deadline algorithms as scheduling policy. As it has been introduced above, the bin-packing problem is defined as a set of bins (processors) with a maximum capacity and a set of boxes (tasks) with some capacity requirements, here referred as *volume*, that must be fit into the bins.

If we denote a task set $\mathcal{T}$ as a set of tuples (C_i, P_i, D_i), where C_i is the worst case execution time of task i, and P_i is its period and D_i is its deadline, the *volume* of a task can be defined as its utilization factor, i.e., $v_i = C_i/P_i$.

Previous works in partitioning schemes addressed the bin-packing problem assuming sets of periodic tasks with deadlines equal to periods and the Rate Monotonic algorithm as the scheduling policy. The test used to check if there is enough room for a new task on a given processor is the sufficient condition by Liu and Layland that could be regarded as a *volume test*, since it only places requirements on the volume (utilization) of a box (task) to fit it into a bin (processor).

The main goal of this section is how to relax the $D_i = P_i$ restriction. The proposed solution is based on requiring an additional requirement to the volume test (utilization test). This requirement makes the bin-packing problem considerably harder and yields a new class of algorithms called restricted bin-packing algorithms. The additional requirement can be introduced as a *shape test*: a box is not only required to have a smaller volume than the available one, but also to fit in a hole of a given geometry. The shape of a box is

not as easy to define as its volume and requires complex algorithms to determine the shape compatibility of a box and a hole. In the real-time scheduling problem checking the compatibility of a task to a hole implies using schedulability tests which depend on the scheduling policy used at each processor. The tests used in this paper are exact feasibility tests: the *completion time test* [4] for fixed priority scheduling algorithms, and *initial critical instant* test (ICI test) [7] for dynamic priority scheduling algorithms.

Summarizing, when the task assignment algorithm tries to fit a new task into a processor, it begins by checking if the available processor utilization is greater than the task utilization factor (volume test). If this is true, then it executes the exact feasibility test on the set of previously allocated tasks plus the new one (shape test). If this also holds, then the task can be allocated to that processor. Due to the complexity of these tests, the new approach to solve the scheduling problem is only suitable for small scale multiprocessor systems. However, this is not great restriction, since real-time control systems use this kind of platforms.

An additional problem introduced by the shape requirement is the problem of ordering the boxes before the allocation process begins. One of the most useful orders is by decreasing capacity requirements. However, this concept is not well defined when shapes are considered: how can it be stated that a box has less capacity requirements than another, if they are different shapes?. A possible criteria proposed in this paper is to consider a *shape factor* defined as D_i/P_i. This factor indicates how restrictive a task shape is, since the lower the task shape factor is, the harder it is to schedule. Previous works on the analysis of partitioning schemes assume all shape factors are equal to 1. Some other criteria to compare this capacity are proposed below, and later validated through extensive simulations. A comparison among all possible algorithms is performed. Also several new metrics are defined in order to evaluate these new algorithms.

3. Metrics and bounds for multiprocessor scheduling algorithms

The problem of evaluating multiprocessor scheduling algorithms is complex, since metrics are usually based on comparing the performance of a given algorithm to an optimal one, but determining optimals is often NP-hard on multiprocessors. This paper overcomes this problem by defining lower bounds or upper bounds on the performance of an optimal algorithm and then defining metrics with respect this bounds. This section starts revising the concept of optimal multiprocessor scheduling algorithm and then it proposes a definition of optimal multiprocessor which leads to the definitions of two bounds: a lower bound for the minimum number of processors required to scheduled a given task set and an upper bound on the utilization which can be obtained with a fixed number of processors.

Given a set of tasks, a schedule is said to be *feasible* if the execution of each task can be completed before its deadline (using some algorithm). If such a schedule exists, the set of tasks is said to be *feasible*. A feasible schedule is defined to be *minimal* if there is no feasible schedule utilizing fewer processors. According to this, an *optimal multiprocessor scheduling algorithm* is an algorithm that for any set of tasks finds a minimal schedule.

An interesting and attractive metric related with the above concept of optimal is the ratio N/N_{opt}, where N is the number of processors required to schedule a task set by a given algorithm, and N_{opt} is the number of processors required by an optimal task assignment algorithm. Unfortunately, since finding an optimal task assignment with a partitioning scheme is NP-hard [5], so is finding N_{opt}.

The approach in this paper, is to develop a set of practical lower bounds that can act as a good approximation for N_{opt}, but unlike this, they can be easily calculated and serve to define new metrics for evaluations. In this sense, two new bounds are introduced: N_u, defined as the lower bound as required by the total utilization of a task set, and N_{OM}, defined as the number of processors needed by an "optimal multiprocessor". It will be shown that:

$$N_u \leq N_{OM} \leq N_{opt}$$

The number of processors required by the total utilization of a task set, denoted as N_u, can be calculated as:

$$N_u = \left\lceil \sum_{T_i \in \mathcal{T}} \frac{C_i}{P_i} \right\rceil$$

To show that N_u is a lower bound for N_{opt} is straightforward from its definition. Moreover, it will be shown that it is not too accurate with some kind of load templates that are characterized in this paper. To introduce a more accurate lower bound, the concept of optimal multiprocessor will be introduced next.

An *optimal multiprocessor* (with N processors), denoted as OM^N, is defined as a monoprocessor that is N-times faster than one of the processors of the corresponding homogeneous multiprocessor. This means that the computations executed by the optimal multiprocessor in a unit of time, require N units of time in any of the processors of the multiprocessor.

The optimal multiprocessor has an interesting property: if a task set is schedulable on a multiprocessor, it is also schedulable on the corresponding optimal multiprocessor. The converse, as expected, does not hold. This property can be easily proved by showing that the OM^N can emulate any feasible schedule of a N-processor, i.e., any feasible schedule of the N-processor can be mapped into a corresponding

feasible schedule on the OM^N. A technique for making this mapping simply consists of dividing a time unit q of the multiprocessor in N time units $q_1, q_2, \cdots, q_N$ of the OM^N and scheduling in q_i the same load that processor P_i of the N-processor. This is obviously possible from the definition of OM^N. An example of this technique is depicted in figure 1. To prove that the converse is not true consider a task set with a unique task with $Ci = 1$ and $Di = 1$, where this values are expressed in time units of the OM^N. This task would require a computing time of N on the corresponding N-processor that would violate the deadline constraint, since they cannot be executed in parallel.

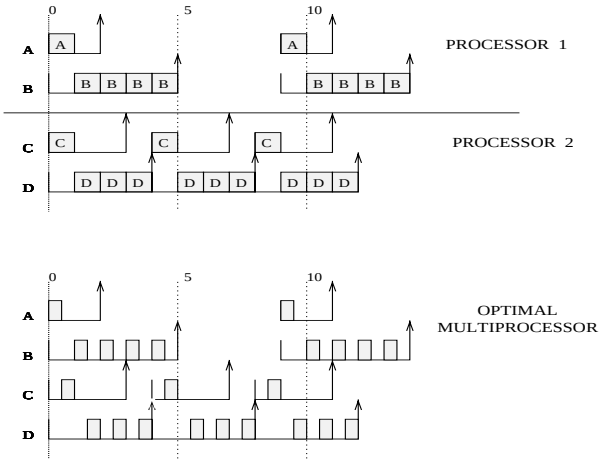

Figure 1. OM^2 optimality example

Using this definition of an optimal multiprocessor, a new lower bound N_{OM} is presented. In order to calculate N_{OM}, it is needed a feasibility test that specifies when a feasible schedule of a set of tasks $\mathcal{T}$ exists over OM^N. Since OM is simply a monoprocessor, well-known feasibility tests can be used.

As it has been showed above, any schedule performed over a real multiprocessor has a direct map over OM. So, if a feasibility test shows that no feasible schedule for $\mathcal{T}$ exists over OM^N, then no scheduler algorithm can find a feasible schedule for a real N-processor. According to this, if N_{OM} is defined as the minimum N for which a feasibility test reports $\mathcal{T}$ as schedulable over OM^N, then N_{OM} is a lower bound on the number of processors needed by an optimal scheduler, i.e., it is a lower bound of N_{opt}.

Next we introduce an upper bound on the utilization which can be obtained with a fixed number of processors N. In order to introduce this bound, consider a particular task set $\mathcal{T}$ where tasks are totally ordered in a sequence according to some arbitrary criteria. Let $\mathcal{T}_A$ be a prefix of $\mathcal{T}$ such that it consists of the initial subsequence of tasks of $\mathcal{T}$ that can be successfully scheduled by some algorithm A. Let $\mathcal{T}_{opt}$ the subsequence of tasks which can been successfully scheduled by an optimal scheduler. It holds that $\mathcal{T}_A \subseteq \mathcal{T}_{opt}$, for any algorithm A. This is easy to prove by self-contradiction. Assume that some algorithm A would be able to schedule more than $|\mathcal{T}_{opt}|$ tasks using N processors. That would mean that the optimal scheduler would need more than N processors to schedule the $\mathcal{T}_A$ set; and this is a contradiction with the definition of optimal scheduler, since N_{opt} is the minimum number of processors where a feasible task set can be scheduled. Once this is proved, U_{opt} for N processors and a given task set $\mathcal{T}$ can be defined as:

$$U^N_{opt}(\mathcal{T}) = \frac{\sum_{T_i \in \mathcal{T}_{opt}} \frac{C_i}{P_i}}{N}$$

that is, the average processor utilization obtained by an optimal scheduled for the $\mathcal{T}_{opt}$ task set. This new metric is an upper bound of $U^N_A(\mathcal{T})$ for any generic algorithm A. In other words, $U^N_{opt}(\mathcal{T})$ is an upper bound on the performance of any assignment algorithm, and therefore, it can be used to estimate how good an assignment algorithm is (e.g. using a metric such as $U^N_A(\mathcal{T})/U^N_{opt}(\mathcal{T})$). Obviously, $U^N_{opt}(\mathcal{T})$ is as harder to obtain as N_{opt}. For this reason, the upper bound $U^N_{OM}(\mathcal{T})$, based on the above definition of optimal multiprocessor, will be used instead in order to define useful metrics. The relation between bounds for processor utilization is:

$$U^N_{OM}(\mathcal{T}) \geq U^N_{opt}(\mathcal{T}) \geq U^N_A(\mathcal{T})$$

The above metrics N_{OM} and $U^N_{OM}(\mathcal{T})$ will be used in the next section to evaluate the restricted bin-packing algorithms proposed in this paper.

4. Experimental evaluation

This section presents several experimental results about partitioning schemes for multiprocessor scheduling. The first experiment provides estimations on lower bounds (N_u and N_{OM}) on the minimum number of processors required to schedule a given set of tasks. Next, it introduces an evaluation in order to decide which is the combination of bin-packing algorithms and ordering criteria which provides higher processor utilizations. Then, it evaluates the goodness of the best bin-packing algorithms comparing the number of processor they require to schedule a given set of tasks versus N_{OM}. Finally, it presents an experiment with a fixed number of processor to evaluate the probability of getting a processor utilization of 90% using the best bin-packing algorithms, and comparing their results with an optimal multiprocessor OM. Previously, the next subsection introduces how to characterize the workload for the experiments and how to generate it.

4.1. Synthetic workload generation

The task model that is used in this paper describes a real-time task T_i using the tuple (C_i, D_i, P_i), as previously described. In order to generate synthetic task sets to evaluate the performance of proposed algorithms, an equivalent task description will be used. This new description depicted in figure 2, defines a task T_i as the tuple $(C_i,$ $\Delta D_i,$ $\Delta P_i)$, where C_i means the same as before, and former characteristics are calculated as: $D_i = C_i + \Delta D_i$ and $P_i = C_i + \Delta D_i + \Delta P_i$.

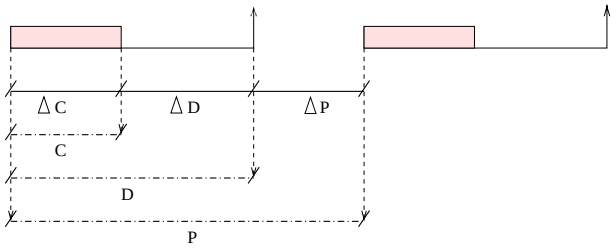

Figure 2. Real-time task alternative description models

This new task description model allows to generate synthetic workloads using a simple algorithm that generates the tasks according to the expressions:

- $C_i = 1 + Unif(MaxC - 1)$
- $D_i = C_i + Unif(MaxD)$
- $P_i = D_i + Unif(MaxP)$

where $MaxC$, $MaxD$ and $MaxP$ are the maximum values of $C_i, \Delta D_i$and ΔP_irespectively, and $Unif(x)$ returns an integer random number in the $[0, x]$ interval following a *uniform distribution.*

4.2. Lower bounds on the minimum number of processors

The first experiment evaluate values of N_u and N_{OM} as a function of $MaxD$ and $MaxP$, and compares N_{OM} accuracy versus N_u. Figure 3 shows the values of N_u and N_{OM} while 4 represents the ratio N_{OM}/N_u. These pictures have been obtained using task sets of 32 tasks, and the following generation pattern: $MaxC$ fixed to 20 and $MaxD$ and $maxP$ varying in the interval $[10, 80]$.

The figures show that N_u is significantly lower than N_{OM} and the N_{OM}/N_u ratio is higher with tasks whose utilization factor is low ($\Delta P \uparrow$) and when they have restrictive deadlines ($\Delta D \downarrow$). This means that, in the above conditions, the number of processors required by a real multiprocessor is significantly higher than the processor required by an optimal multiprocessor or by the utilization.

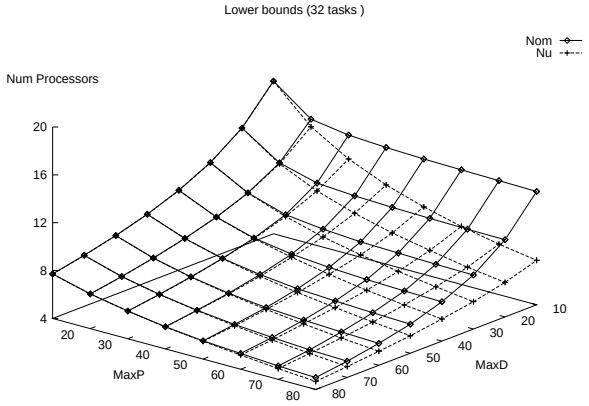

Figure 3. N_u and N_{OM} values

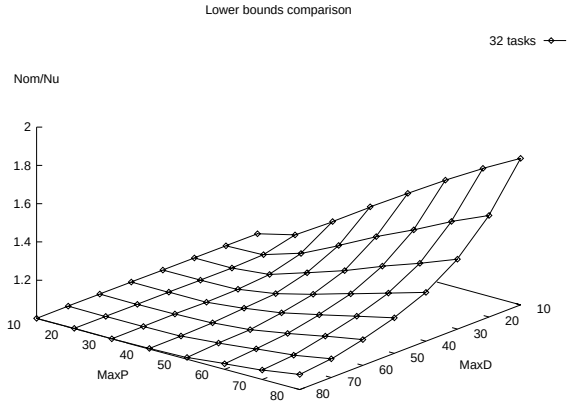

Figure 4. N_{OM}/N_u ratio

4.3. Selection of the best bin-packing algorithm for partitioning schemes

The aim of this experiment is to determine the best combination of bin-packing algorithm and task ordering (order in which tasks are processed). The evaluation process analyzes the number of processors they require to schedule a given set of tasks versus N_{OM}. This task is not easy to solve from the analytical point of view, so this section proposes an exhaustive evaluation of all possible combinations for two cases: static priorities and dynamic priorities.

The considered bin-packing algorithms are: *first fit, next fit, best fit* and *worst fit*, that will referred as FF, NF, BF and WF from now on. On the other hand, possible task orderings are based on task characteristics. The considered ones are: deadline, D_i, period, P_i, utilization factor, $UF_i = C_i/P_i$, weight factor, $WF_i = C_i/D_i$, shape factor, $SF_i = D_i/P_i$, and two combinations of these ones: $WS_i = (C_iP_i)/D_i^2$, and $WU_i = C_i^2/(D_iP_i)$. The orderings defined by these factors can be considered as increasing or decreasing, resulting in a total set of orderings $\mathcal{O} = \{ID, IP, IUF, IWF, ISF, IWS, IWU, DD, DP, DUF, DWF, DSF, DWS, DWU\}$, where the initial stands for *I*ncreasing or *D*ecreasing order, and the rest specifies the ordering factor. A complete partitioning scheme will be referred as *PP-AA:OO*, where *PP* is the scheduling policy, *AA* is the bin-packing algorithm and *OO* the selected order from $\mathcal{O}$, e.g., *DM-FF:ID* means the *first fit* bin-packing algorithm, where the task set has been ordered by increasing deadlines and the scheduling policy is *deadline monotonic*.

The set of bin-packing algorithms and considered orderings results in a 56 possible combinations for each scheduling policy: DM and EDF. In order to evaluate all these partitioning schemes, extensive simulations have been performed, and their results are shown next.

Task generation algorithm and evaluation process

The task set used to evaluate the proposed algorithms has been generated using the synthetic workload generation process described above. The evaluation process is simple:

1. A set of generation parameters are established and the task set emptied,
2. a new task is generated and added to the current task set,
3. the algorithm to be evaluated tries to schedule the new task set with the user defined restrictions (number of processors, ...),
4. **if** the task set is successfully scheduled

   then a new task is added and the process repeated from point 2.

   else the algorithm metrics are calculated, the task set emptied, and another algorithm is evaluated using current generation parameters;
5. this process is repeated with all considered algorithms for several hundred times using the same generation parameters, but changing the random number generator seeds.
6. when all the experiments with a given generation parameters are done, these are changed until all previously set parameters are visited.

Using this simple method, the 112 proposed partitioning schemes have been evaluated, fixing $MaxC$ parameter to 20 and varying the $MaxD$ and $MaxP$ parameters in the interval $[10, 80]$ using steps of 10 units in both cases. The number of used processors have also been varied from 4 to 16 using steps of 4. For every possible parameter combination, 500 task sets have been generated.

The metric used to evaluate the algorithms is $U_A^N(\mathcal{T})$. The resulting values of this metric for all parameter configurations have been averaged in a single value, for each algorithm. This value provides an average processor utilization for each proposed algorithm. If this average value is used as metric to compare these partitioning schemes, the result shows the best algorithms are: *DM-FF:DWU* and *EDF-FF:DWF*. The interpretation of these results deserves some observations: on one hand, results could be affected to some extent by the evaluation process and the generation pattern and, on the other, the fact they are the best in the average case does not ensure they will perform best in some particular case.

In order to improve the behavior of the basic partitioning schemes, a simple method to construct complex algorithms is next suggested.

Sequential forward search constructions

The basic idea to construct more reliable schemes is very simple, and relies on the heuristic called sequential forward search. The method consists of constructing a stack of algorithms, which behaves as a unique algorithm. The steps that should be performed to construct such a stack are the following:

1. get the scheme that obtains the maximum average value as the first component of the stack.
2. add in every step the algorithm that allows the *stack* to obtain higher values of the used metric.

This method can be applied as many times as required for obtaining a good performance, stopping when the increase of performance is insignificant.

Using this method, two complex partitioning schemes have been constructed, one for each scheduling policy: *DM-SFS* and *EDF-SFS*. These schemes are composed by a stack

of eight algorithms each. An experimental evaluation shows their performance achieves almost 99.8 % of the performance all the possible combinations together, that is, these 8 algorithms covers the 99.8 % results of the set of 56 possible schemes, for each policy. Therefore, the performance of these new schemes is close to the limit performance that can be obtained using restricted bin-packing algorithms so they are ideal candidates for evaluation. These schemes allow to increase performance and reliability at a reasonable cost-performance ratio.

The *DM-SFS* and *EDF-SFS* schemes will be used for the rest of this experimental evaluation as representatives of fixed and variable priority assignment schemes.

4.4. Experiments with a variable number of processors

The study about the number of processors required to schedule a given task set is a metric to compare the *DM-SFS* and *EDF-SFS* schemes and to give an idea to the system designer about the power or number of processors required in the hypothetical case this would be a design factor.

This evaluation is performed using the metric N/N_{opt} introduced above, but using N_{OM} as a lower bound of N_{opt}, i.e., the used metric is: N/N_{OM}. In this expression, N is the number of processors a given algorithm needs to allocate a task set, and N_{OM} is the number of processors the optimal multiprocessor needs for the same task set.

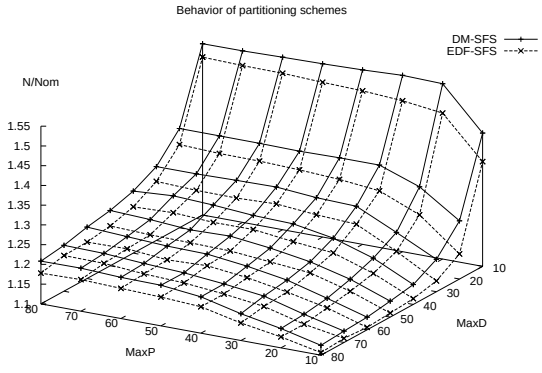

Figure 5. N/N_{OM} (32 tasks)

An extensive simulation has been performed, using the synthetic generation procedure described above, but with a fixed number of tasks. The task sets have a fixed $MaxC$ parameter of 20 and their cardinality is 32. The $MaxD$ and $MaxP$ parameters vary in the interval $[10, 80]$ using steps of 10 units in both cases. For every possible parameter combination 500 task sets have been generated. The results of this experiment are shown in figure 5. It can be observed that the proposed schemes behave reasonably well compared to OM, i.e., the N/N_{OM} factor is close to 1, with an obvious advantage for dynamic priorities scheme *EDF-SFS*. The only exceptional cases occur when the task sets are composed of tasks with restrictive deadlines ($MaxD \downarrow$), i.e., tasks with weight factors rounding 0.66. This is due to fact that the *optimal multiprocessor* can parallelize the code, reducing task weight and making tasks easier to schedule, while any other algorithm should try to schedule the tasks as they really are. Since this problem also affects the optimal assignment scheme, it could be assumed the real factor N/N_{opt} will be lower than the one showed in these cases.

The presented results are similar to those previously presented by Burchard *et al.* [2], but eliminating the $D_i = P_i$ restriction and allowing to use dynamic priority policies.

4.5. Experiments with a fixed number of processors

The evaluation of a system with a fixed number requires different metrics to the ones used in previous experiments. The metric proposed for this case is the expression $U_A^N(\mathcal{T})$ introduced in section 3, indicating how much work can be fitted into the system processor before they report the load as not schedulable.

The simulation parameters and evaluation process are the same that are used in the evaluation process for selecting the best bin-packing algorithm (section 4.3), but without averaging all the result values in a single one. The figure 6 depicts the experiment results, when eight processors are used. Every point shows $U_{DM-SFS}^N(\mathcal{T})$, $U_{EDF-SFS}^N(\mathcal{T})$ and $U_{OM}^N(\mathcal{T})$ values, where task set $\mathcal{T}$ has been generated with the $MaxD$ and $MaxP$ that characterize every point.

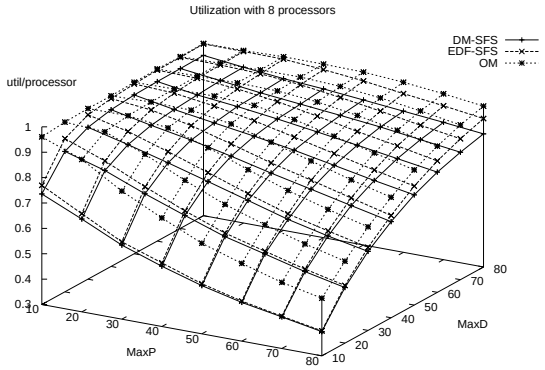

Figure 6. Utilization with 8 processors

It can be observed that the utilizations achieved with *EDF-SFS* are a little bit higher from those obtained with *DM-SFS*. But the most important result is that both results are close enough to the maximum achievable utilization, depicted by the *OM* curve. Taking into account that the optimal assignment scheme should be between the *OM*

and the *EDF-SFS* curves, the results can be considered good enough to avoid looking for more complex and time-consuming algorithms.

From this results, it can be also estimated the probability of obtaining higher processor utilizations. An example is showed in figure 7. It shows the estimated probability to obtain a feasible assignment when the task sets produce processor utilizations greater than 90%.

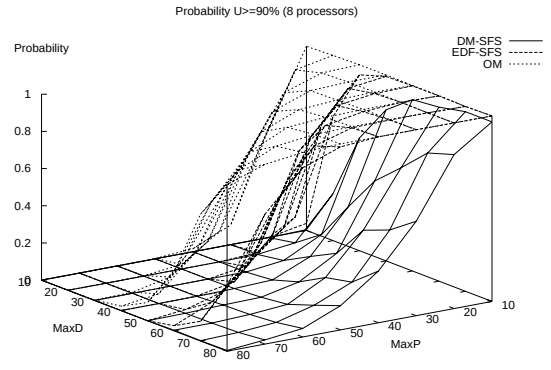

Figure 7. Probability to obtain more than 90% utilization (8 proc)

This result has been obtained from a system with eight processors, and it shows that when a task set $\mathcal{T}$ satisfies the relations:

$$\frac{\sum \Delta D_i}{\sum \Delta P_i} < 1 \ \& \ \sum \frac{C_i}{P_i} \geq 0.9$$

the task set is virtually impossible to schedule using partitioning schemes, and much of them are infeasible for 8 processor systems (reported as *not schedulable* by the optimal multiprocessor). It can be also observed that the problem of finding a feasible assignment using fixed priorities is much harder than using dynamic ones.

5. Conclusions

This paper has introduced new partitioning schemes for multiprocessor real-time systems based on bin-packing algorithms, using both scheduling policies: fixed and dynamic priorities. The new schemes, called *DM-SFS* and *EDF-SFS*, relax the restriction $D_i = P_i$ allowing higher processor utilization and enhanced schedulability. The problem of evaluating this algorithms is complex, since metrics are usually based on comparing the performance of a given algorithm to an optimal one, but determining optimals is often NP-hard on multiprocessors. This problem has been overcome by defining lower or upper bounds on the performance of an optimal algorithm and then defining metrics with respect this bounds. The evaluation has shown that the algorithms exhibit extremely good behaviors; it can be considered close enough to the maximum achievable utilization. The obtained results also show a clear advantage of dynamic priority policies over fixed priorities policies when task sets require high processor utilizations.

As future work, it can be pointed out the refinement of the model by introducing new requirements on the feasibility condition as are shared resources.

References

[1] S. Baruah, L. Rosier, and R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, pages 301–324, Dec. 1990.

[2] A. Burchard, J. Liebeherr, Y. Oh, and S. Son. New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Transactions on Computers*, 44(12):1429–1442, Dec. 1995.

[3] S. Davari and S. Dhall. An on line algorithm for real time tasks allocation. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 194–200, 1986.

[4] J. Lehoczky, L. Sha, and Y. Ding. The rate-monotonic scheduling algorithm: Exact characterization and average behavior. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 166–171, 1989.

[5] J.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

[6] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *J. ACM*, 20(1):46–61, Jan. 1973.

[7] I. Ripoll, A. Crespo, and A. Mok. Improvements in feasibility testing for real-time tasks. *Real-Time Systems*, 11:19–39, 1996.

[8] S. Sáez, J. Vila, and A. Crespo. Dynamic scheduling solutions for real-time multiprocessor systems. *Control Engineering Practice*, 5(7):1007–1013, Jul. 1997.

[9] J. Stankovic, M. Spuri, M. D. Natale, and G.Buttazzo. Implications of classical scheduling results for real-time systems. *IEEE Computer*, pages 16–25, Jun. 1995.