EDF and RM Multiprocessor Scheduling Algorithms: Survey and Performance Evaluation

Omar U. Pereira Zapata, Pedro Mejía Alvarez
CINVESTAV-IPN, Sección de Computación
Av. I.P.N. 2508, Zacatenco, México, D.F. 07000
{opereira, pmejia}@computacion.cs.cinvestav.mx

***Abstract*— In this paper a survey of the best-known real-time scheduling techniques executing on multiple processors is conducted. These techniques solve the problem of scheduling sets of periodic and preemptable tasks, using the Rate Monotonic and Earliest Deadline First scheduling policies. Diverse algorithms are introduced and their performance is compared for the partitioning scheme as well as for the global scheme. Extensive simulation experiments are conducted to analyze and compare the performance of the algorithms.**

I. Introduction

Real-time systems are those in which its correct operation not only depends on the logical results, but also on the time at which these results are produced. These are high complexity systems that are executed in environments such as: military process control, robotics, avionic systems, distributed systems and multimedia. The computational demands of these applications require the use of multiple processors. For example, the DD-21 land destroyer, require the use of 500 to 1000 processors[23]. The use of the real-time systems in multiple processors has extended not only because of the high computing requirements of these applications, but also because of the reduction of costs and the proliferation of design tools in this type of systems.

Nowadays there exists a great number of research work focused on the study of real-time scheduling in one processor [27], [24], [25], [20]. Diverse problems related to real-time scheduling have been widely studied, for example, synchronization [36], aperiodic tasks [37], precedence constraints, overloads, and fault tolerance [16]. Nevertheless, for multiple processors some of these problems are still without solution, or have been solved only in a limited form.

In this paper we analyze and compare the performance of diverse partitioning and global scheduling schemes for multiprocessors using Rate Monotonic (RM) and Earliest Deadline First (EDF).

The rest of this document is organized as follows: In Section II, an overview of scheduling on multiple processor real-time systems is introduced. Section III introduces the system model and the taxonomy of the algorithms used in this paper. In Section IV, the partitioned multiprocessor approach under RM and EDF is introduced and the best-known heuristic algorithms for this approach are described. Also different schedulability conditions used in multiprocessor scheduling are introduced. In Section V, the global multiprocessor approach under RM and EDF is introduced and the best-known heuristic algorithms for this approach are described. In Section VI, an extensive set of simulation experiments are executed to measure the performance of the partitioned and global scheduling algorithms under RM and EDF scheduling on *fixed* and *infinite* number of processors. Finally the conclusions appear in Section VII.

II. Scheduling on Multiprocessor Systems

Meeting the deadlines of a real-time task set in a multiprocessor system requires a scheduling algorithm that determines, for each task in the system, in which processor they must be executed (*allocation problem*), and when and in which order, with respect to other tasks, they must start their execution (*scheduling problem*). This is a problem with a difficult solution, because 1) some research results for a single processor not always can be applied for multiple processors [14], [25], 2) in multiple processors different scheduling anomalies appear [18], [1], [19] and 3) the solution to the allocation problem requires of algorithms with a high computational complexity.

The scheduling of real-time tasks on multiprocessors can be carried out under the *partitioning scheme* or under the *global scheme*. In the partitioning scheme (Figure 1.a) all the instances (or jobs) of a task are executed on the same processor. In contrast, in the global scheme (Figure 1.b), a task can migrate from one processor to another during the execution of different instances. Also, an individual job of a task that is preempted from some processor, may resume execution in a different processor. Nevertheless, in both schemes parallelism is prohibited, that is, no job of any task can be executed at the same time on more than one processor.

On both schemes, the *admission control* mechanism not only decides which tasks must be accepted, but also it must create a feasible allocation of tasks to processors (i.e., on each processor, all tasks allocated must met their deadlines). For the partitioning and global schemes, task sets can be scheduled using static or dynamic schedulers. In any case, the computational complexity associated to the admission control must remain as low as possible, especially for the dynamic case.

It is well known that the Liu and Layland results break down on multiprocessor systems [26]. Dhall and Liu [14] gave examples of task sets for which global RM and EDF scheduling can fail at very low processor utilizations, essentially leaving all but one processor idle nearly all of the time.

Reasoning from such examples is tempting to conjecture that perhaps RM and EDF are not good or efficient scheduling policies for multiprocessor systems. However, at the moment this conclusions have not been formally justified [5].

The partitioning scheme has received greater attention than the global scheme, mainly because the scheduling problem can be reduced to the scheduling on single processors, where at the moment a great variety of scheduling algorithms exist. It has been proved by Leung and Whitehead [25] that the partitioned and global approaches to static-priority scheduling on identical multiprocessors are *incomparable* in the sense that 1) there are task sets that are feasible on m identical processors under the partitioned approach but for which no priority assignment exists which would cause all jobs of all tasks to meet their deadlines under global scheduling on the same m processors, and 2) there are task sets that are feasible on m identical processors under the global approach, which cannot be partitioned into m distinct subsets such that each individual partition is feasible on a single static-priority uniprocessor.

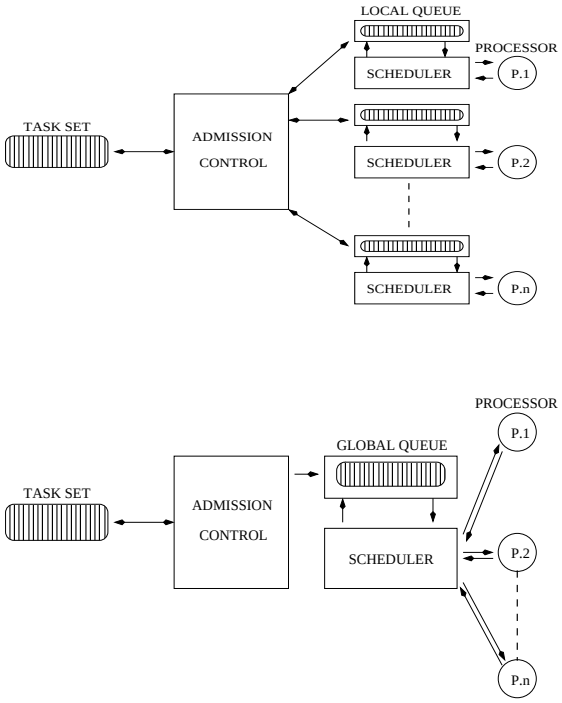

Fig. 1. (a). Partitioning and (b). Global Scheduling Schemes

III. System Model

In this paper, the problem to be studied is to schedule a set of n real-time tasks $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$, on a set of m processors, $P = \{P_1, P_2, ..., P_m\}$. A task is usually a thread or a process within an operating system. The parameters that define a task are: the execution time C_i, the period T_i, and the deadline D_i. We will consider that only periodic and preemptive tasks can execute in the system. Each periodic task, denoted by τ_i is composed of an infinite sequence of *jobs*. The period T_i of the periodic task τ_i is a fixed time interval between release times of consecutive jobs in τ_i. Its execution time C_i is the maximum execution time of all the jobs in τ_i. The period and the execution time of task τ_i satisfies that $T_i > 0$ and $0 < C_i \leq T_i = D_i, (i = 1, \ldots, n)$. $u_i = C_i/T_i$ is defined as the utilization factor of task τ_i. The utilization factor of the set of tasks is the sum of the utilizations of the tasks in the set, $U_{TOT} = \sum_{i=1}^{n} \frac{C_i}{T_i}$. Let α denote the maximum possible utilization of any periodic task in the system, $\alpha = max_{i=1,\ldots,n} \ (C_i \ / \ T_i)$. Following the hard real-time scheme, in this paper we will consider that a job in τ_i that is released at time t must complete at most D_i time units after t, that is, it must complete within the time interval $(t, t + D_i]$.

In the model used in this paper, the following restrictions also apply. The tasks are independent. That is, the arrival of some task is not affected by arrival of any other tasks in the system. The cost of the context switch of the tasks is considered negligible. No resources, other than the CPU, are shared among tasks. The cost of the admission control mechanisms is considered null. The Rate Monotonic and Earliest Deadline First scheduling policies will be considered in this paper. The cost of migration is considered null and every job is allocated to at most one processor at a time.

In this paper, a performance analysis of different scheduling algorithms is carried out for identical multiprocessors (i.e., processors executing at the same speed), where the priorities of the tasks are assigned statically (using RM) and dynamically (using EDF). We will study algorithms that allow *migration* (i.e, global schemes) together with the algorithms that do not allow migration (i.e., partitioned schemes). For the partitioned and the global schemes we will study those algorithms that consider a fixed and an infinite number of processors. Also, for the partitioned scheme we will study off-line and on-line algorithms. A taxonomy of the multiprocessor scheduling algorithms studied in this paper, is illustrated in Figure 2.

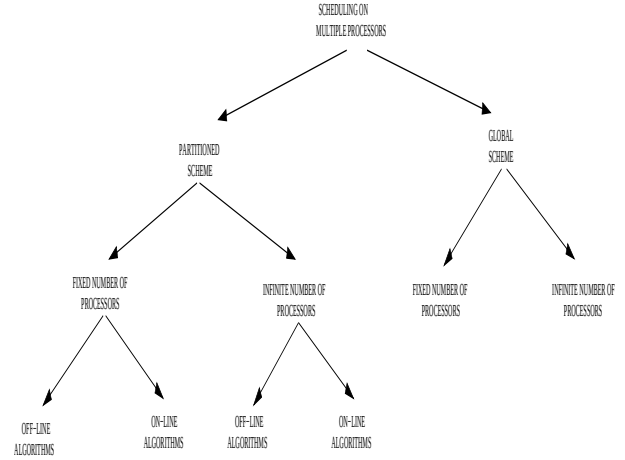

Fig. 2. (a). Taxonomy of Multiprocessor Scheduling Algorithms

IV. Partitioned Scheduling on Multiprocessors

In the multiprocessor partitioning scheme it is necessary to choose the scheduling algorithm on every processor, and the allocation algorithm used to allocate tasks to processors. The allocation problem has been solved assuming a *fixed* or an *infinite* number of processors. It has been demonstrated that the allocation problem is an *NP-Hard* problem [25].

In the *fixed case*, the aim is to find an allocation algorithm and a schedulability test to verify if a given task set is schedulable (or not) on a fixed number of processor [26] .

In the *infinite case*, the problem of allocating a set of tasks to a set of processor is analogous to the *Bin-Packing* problem [11]. In this problem, the processor is the recipient *bin*, whose capacity is given by the utilization bound of the local processor. In the Bin-Packing problem, it is required to put n objects (tasks) with weight w_k (utilization $u_k = w_k$) into the minimum possible number of bins (processors), such that the total weight of the objects on each bin do not exceed its maximum capacity[1] (c).

Since this problem is *NP-Hard*, optimal algorithms for solving this problem cannot even solve this problem in pseudo-polynomial time.

The best-known heuristics found in the literature that solve the allocation problem are[11]:

- **First-Fit (FF)**: FF allocates a new object to a non-empty bin[2] with the lowest index, such that the weight of the new object along with the weights of the objects already allocated to that bin, do not exceed the capacity of the bin. If the new object exceeds the capacity of the bin, then FF allocates the object to an empty bin.
- **Best-Fit (BF)**: If an object cannot be allocated to a non-empty bin then BF puts the object into an empty bin. Otherwise, BF will allocate the object to the non-empty bin with smallest capacity available, in which the object can be allocated. If there is more than one bin with the same capacity, then BF will choose the bin with smallest index.
- **Next-Fit (NF)**: After allocating the first object to the current bin, NF allocates the next object to the same bin, only if it fits into this bin. Otherwise, NF allocates the object to the next empty bin. Note that NF does not check if the object can be allocated in previous bins.
- **Worst-Fit (WF)**: This algorithm is similar than BF, with the difference that WF allocate objects into the bins with the greatest capacity available, in which they can be feasibly allocated.

The performance of the algorithms for the *infinite* processors case will be denoted as follows. Let N_{opt} be the number of processors used by an optimal algorithm, and let $N(A)$ be the number of processors used by algorithm A. The guaranteed performance of the heuristic algorithm A is defined by, $\Re_A = lim_{N_{opt}\to\infty} \frac{N(A)}{N_{opt}}$. Note that, when this value is smaller (close to 1), the solution provided by algorithm A is closer to the optimal solution.

For the partitioned scheme, the allocation of task to processors depends not only on the allocation algorithm itself, but also on the schedulability condition used. The value of c in the bin-packing problem, depend on the schedulability condition. The best-known schedulability conditions for RM and EDF used for multiprocessor scheduling will be described in the next section.

[1]In general, the maximum capacity is given by the schedulability condition used.

[2]In our notation, an empty bin, is a bin with no objects allocated. In contrast, a non-empty bin is a bin with at least one object allocated.

A. Schedulability Conditions for Rate Monotonic

In this section diverse schedulability conditions used in Rate Monotonic multiprocessor scheduling are introduced.

1) Schedulability Condition L&L (Liu & Layland): L&L is a schedulability condition for task sets scheduled under RM, that is based on the utilization of the processor [27]. In this condition, the utilization of the task set is compared with a utilization bound, which depends on the number of tasks in the system. A task set will not miss any deadline if it meets the following condition:

$$\sum_{i=1}^{n} C_i/T_i \leq n(2^{1/n} - 1) \tag{1}$$

Theorem 1: **(Condition L&L)** [27]: If a set of tasks τ is schedulable under RM algorithm, then the minimum utilization achieved is $n(2^{1/n} - 1)$. When $n \to \infty$, then $n(2^{1/n} - 1) \to ln\ 2$.

Condition L&L is *sufficient but not necessary*, therefore, if a set of tasks meet the condition, then all the tasks will meet their deadlines. Nevertheless, it can be the case in which the task set does not meet this condition, and still the task set is schedulable under RM.

2) Schedulability Condition IP (Increasing Period): In condition IP [14], it is required that the periods of the tasks are ordered increasingly, thus the timing constraints of the task set must be known a priori. Condition IP, introduced by Dhall and Liu [14] is used in the implementation of the multiprocessor algorithms Rate Monotonic Next Fit and Rate Monotonic First Fit.

Theorem 2: **(Condition IP)** [14]: Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a set of tasks with $T_1 \leq T_2 \leq \ldots \leq T_n$ and let

$$u = \sum_{i=1}^{n-1} C_i \ / \ T_i \ \leq \ (n-1)(2^{1/(n-1)} \ - \ 1) \tag{2}$$

If the following condition is met,

$$C_n \ / \ T_n \ \leq \ 2(1 + \ \frac{u}{(n-1)})^{-(n-1)} \ - \ 1 \tag{3}$$

then the set of tasks will have a feasible schedule under the RM algorithm. When $n \to \infty$, the minimum utilization of task τ_n approaches to $(2\ e^{-u} \ - \ 1)$.

3) Schedulability Condition PO (Period Oriented): Condition PO, was introduced by Burchard *et al.* [10] in the development of the RMST and RMGT multiprocessor algorithms. This condition considers that it is possible to increase the utilization of a task set if the periods of all tasks are close to each other.

Theorem 3: **(Condition PO)** [10]: Given a set of tasks $\tau = \{\tau_1, \ldots, \tau_n\}$, V_i and β are defined as follows,

$$V_i = log_2\ T_i - \lfloor\ log_2\ T_i\ \rfloor \qquad i = 1, \ldots, n \tag{4}$$

and

$$\beta = max\ V_i - min\ V_i \qquad i = 1, \ldots, n \tag{5}$$

(a) if $\beta < (1 - 1/n)$ and the total utilization satisfies that:

$$U \leq (n-1)(2^{\beta / (n-1)} - 1) + 2^{1-\beta} - 1 \quad (6)$$

then the task set is schedulable on one processor under RM.

(b) if $\beta \geq (1 - 1/n)$ and the total utilization satisfies that:

$$U \leq n(2^{1/n} - 1) \quad (7)$$

then the task set is schedulable on one processor under RM.

Condition 3.b) is similar to condition L&L [27], however, in general condition PO has better performance than L&L when Condition 3.a) is met. Condition PO, in its most simple version, is defined in Corollary 1.

Corollary 1: **(Condition PO)** [10]: Given a set of tasks $\tau = \{\tau_1,\ldots,\tau_n\}$ and given β (defined as in Theorem 3), if the total utilization satisfies that:

$$U = \sum_{i=1}^{n} C_i/T_i \leq max\{ln\ 2, 1 - \beta\ ln\ 2\} \quad (8)$$

then the task set can be feasibly scheduled on one processor under RM.

4) Schedulability Condition UO (Utilization Oriented): Y. Oh *et al.* [33] introduced a schedulability condition based on the utilization of the tasks, in the development of the RM-FFDU multiprocessor algorithm. In this condition, besides taking into consideration the number of tasks, the utilization of the tasks is also considered.

Theorem 4: **(Condition UO)** [33]: Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_{n-1}\}$ be a set of $(n-1)$ tasks feasibly scheduled under RM. A new task τ_n can be feasibly scheduled together with the (n-1) tasks already in the system (on one processor under RM), if the following condition is met.

$$C_n/T_n \leq 2[\prod_{i=1}^{n-1}(1+u_i)]^{-1} - 1 \quad (9)$$

The complexity of this condition is $O(n)$. In this condition tasks do not require a previous ordering.

5) Schedulability Condition RBOUND: Lauzac *et al.* [21] developed the schedulability condition RBOUND for the RM scheduling policy. This condition uses information of the periods of the task set to achieve a high utilization.

In order to apply condition RBOUND to a task set, first it is necessary to transform the original task set to an equivalent task set where the ratio between maximum and minimum periods is less than 2, $r = T_{max}/T_{min} < 2$. Then, the RBOUND condition is applied to the modified task set to verify its schedulability. As shown in Lemma 1, if the transformed task set is feasibly scheduled then the original task set is also feasible. Condition RBOUND is described in Theorem 5. The transformation procedure is made by the algorithm $ScaleTaskSet$, defined in Figure 3.

Lemma 1: [21]: Let τ be a given periodic task set τ, and let τ' be the transformed task set after applying the $ScaleTaskSet$ algorithm to τ. If τ' is schedulable on one processor under RM, then τ is also schedulable.

Theorem 5: **(Condition RBOUND)** [21]: Consider a periodic task set τ, and let τ' be the transformed task set after executing the $ScaleTaskSet$ on τ. If,

$$\sum_{i=1}^{n} C_i/T_i \leq (n-1)(r^{1/(n-1)} - 1) + (2/r) - 1 \quad (10)$$

where $r = T'_n/T'_1$, the task set τ can be feasibly scheduled on one processor under RM.

Corollary 2: **(Condition RBOUND)** [21]: When $n \to \infty$ the minimum achievable processor utilization approaches to $(ln\ r) + (2/r) - 1$.

Condition RBOUND is a sufficient but not a necessary condition.

ScaleTaskSet (In: τ, Out: τ')

```
1. begin
2.     Sort the task set in τ by increasing period
3.     for (i = 1 to n − 1) do
4.             T'_i := T_i 2^⌊ log Tn/Ti ⌋
5.             C'_i := C_i 2^⌊ log Tn/Ti ⌋
6.     Sort the tasks in τ' by increasing period
7.     return (τ')
6. end
```

Fig. 3. Algorithm ScaleTaskSet

6) Exact Schedulability Condition (Le): Lehoczky *et* al. [24] introduced a necessary and sufficient (exact) schedulability condition for task sets executing under RM.

Theorem 6: **(Condition Le)** [24]: Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a task set with n tasks and $T_1 \leq T_2 \leq \ldots \leq T_n$. τ_i can be feasibly scheduled if and only if,

$$L_i = min_{\{t \in S_i\}} (W_i(t) / t) \leq 1 \quad (11)$$

The entire task set can be feasibly scheduled under RM if and only if

$$L = max_{\{l \leq i \leq n\}} L_i \leq 1 \quad (12)$$

where

$S_i = \{ k\ T_j \mid j = 1, \ldots, i;\ k = 1, \ldots, \lfloor T_i/T_j \rfloor \}$,

$W_i(t) = \sum_{j=1}^{i} C_j \lceil t/T_j \rceil$,

Condition Le is an exact condition. It can be observed that its computational complexity is pseudo-polynomial.

B. Schedulability Condition for EDF

In 1973, Liu y Layland [27] proposed a necessary and sufficient utilization bound for algorithm EDF.

Theorem 7: **(EDF)** [27]: If a set τ of n tasks is schedulable according to the Earliest Deadline First algorithm then the least upper bound is 1.

Theorem 7 demonstrates that any set of periodic preemptive and independent tasks is schedulable using EDF if and only if the total utilization of the task set is not higher than 1. Note

that in this condition $D_i = T_i$. Dertouzos [13] demonstrated that EDF is optimal for any periodic and non-periodic task sets.

C. Rate Monotonic under the Partitioning Scheme

In the following sections, we will describe the heuristic static scheduling algorithms executed under RM on multiple processors. We will introduce off-line heuristic algorithms followed by on-line heuristic algorithms.

1) Rate Monotonic Next Fit (RMNF): The Rate Monotonic Next Fit (RMNF) [14] algorithm, illustrated in Figure 4, orders tasks increasingly according to their periods, before the allocation process.

In this algorithm, task τ_i is allocated to processor P_j (step 5 from the algorithm) only if, after using the schedulability condition IP, there is a feasible schedule for the task set (step 4, from the algorithm). Otherwise, task τ_i is allocated to processor P_{j+1}. After task τ_i is allocated, the next step is to increase index i (step 9 from the algorithm). Steps 3 to 9 from the algorithm are repeated until all tasks are allocated. When the algorithms finishes, the total number of processors required for the feasible scheduling of the task set is given by variable j. The computational complexity of algorithm RMNF is $O(n\ log\ n)$ (where n is the number of tasks in the set), and its performance of this algorithm is $2.4 \leq \Re_{RMNF} \leq 2.67$ [14].

From here on, we will assume that in the notation used the allocation of task τ_i to processor P_j consists on increasing the utilization of the processor by, $U_j = U_j + u_i$, where U_j denotes the utilization of processor P_j and u_i denotes the utilization of task τ_i.

Input: Task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$
and a set of processors $\{P_1, \ldots, P_m\}$.
Output: j; number of processors required.

```
1. Sort the tasks by increasing periods
2. i := 1; j := 1;   /* i = i^th task, j = j^th processor */
3. while (i ≤ n) do
4.     if ( u_i ≤ Condition IP) then
5.         U_j := U_j + u_i;
6.     else
7.         U_{j+1} := U_{j+1} + u_i;
8.         j := j + 1;
9.     i := i + 1;
10.return (j);
11.end
```

Fig. 4. Rate Monotonic Next Fit (RMNF) Algorithm

2) Rate Monotonic First Fit (RMFF): In the allocation process, algorithm RMNF verifies the current processor to see if a feasible schedule can be found for a new task (along with the previously allocated tasks). If there is no feasible schedule in that processor, RMNF allocates the task on an idle processor, even though the task could be feasibly scheduled in another non-idle processor previously used. To overcome this waste of processor utilization, RMFF always checks the schedulability of the new task from the first processor to the current one on which the task can be scheduled.

The Rate Monotonic First Fit (RMFF) algorithm [14], illustrated in Figure 5, orders tasks increasingly according to their periods, before the allocation process. To allocate task τ_i, it is required to select the processor with smallest index, such that task τ_i along with previously allocated tasks, finds a feasible schedule using the schedulability condition IP.

In the algorithm, if task τ_i satisfies condition IP (step 4 from the algorithm) then it is schedulable, so the task is allocated to processor P_m (step 7). k_m denotes the number of tasks allocated to processor P_m, and U_m denotes the total utilization of the k_m tasks. If $(i > n)$, its means that all tasks have already been allocated (step 3), then the algorithm finishes, otherwise, the index is increased (step 10), and the algorithm continues with steps 4 to 10. When the algorithms finishes, the total number of processors required for the feasible scheduling of the task set is given by variable j.

The performance of the RMFF algorithm is $2 \leq \Re_{RMFF} \leq (4 \times 2^{1/3})/(1 + 2^{1/3}) \approx 2.23$ [14] and its computational complexity is $O(n\ log\ n)$.

Input: Task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$
and a set of processors $\{P_1, \ldots, P_m\}$.
Output: j; number of processors required.

```
1. Sort the tasks by increasing periods
2. i := 1; j := 1;   /* i = i^th task, j = j^th processor */
3. while (i ≤ n) do
4.     q := 1;
5.     while (u_i > 2(1 + U_q/k_q)^{-k_q} − 1) do
6.         q := q + 1; /* q denotes the processor index*/
7.     U_q := U_q + u_i;   k_q := k_q + 1
8.     if (q > j) then
9.         j := q;
10.    i := i + 1;
11.return (j);
12.end
```

Fig. 5. Rate Monotonic First Fit (RMFF) Algorithm

Input: Task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$
and a set of processors $\{P_1, \ldots, P_m\}$.
Output: j; number of processors required.

```
1. Order tasks increasingly by their periods
2. i := 1; j := 1;   /* i = i^th task, j = j^th processor */
3. while (i ≤ n) do
4.     if (u_i ≤ Condition IP) and
       (2(1 + U_j / k_j)^{-k_j} − 1) is the smallest possible
5.         U_j := U_j + u_i;
6.     else
7.         task τ_i is allocated to an idle processor
8.     i := i + 1;
9. return (j);
10.end
```

Fig. 6. Rate Monotonic Best Fit (RMBF) Algorithm

3) Rate Monotonic Best Fit (RMBF): Algorithm RMBF[35], illustrated in Figure 6, allocates the task to the processor with smallest available utilization, on which it can be feasibly scheduled.

RMBF orders tasks increasingly according to their periods, before the allocation process. In this algorithm, the allocation of task τ_i to processor P_j (step 5 from the algorithm) requires finding the smallest j such that task τ_i together with all the task previously allocated to processor P_j, can be feasibly scheduled according to the schedulability condition IP, and that the value of $2(1 + U_j / k_j)^{-k_j} - 1$ be as small as possible (step 4).

The processor with the smallest available utilization, on which the new task can be feasibly scheduled, will be the one where the task will be allocated.

The performance of RMBF is $\Re_{RMBF} \leq 2 + (3 - 2^{3/2}) / a \approx 2.33$, where $a = 2\,(2^{1/3} - 1)$[35], and its computational complexity is $O(n\ log\ n)$.

4) Rate Monotonic First Fit Decreasing Utilization (RM-FFDU): Algorithm RM-FFDU [33], shown in Figure 7, orders the task set decreasingly according to their utilizations, and follows the First-Fit heuristic to allocate tasks to processors. This algorithm uses the UO schedulability condition.

In this algorithm, task τ_i is allocated to processor P_q (step 7) only if it meets condition UO. Note that u_{qh} denotes the utilization of task τ_h in processor P_q, and k_q denotes the number of tasks allocated to processor P_q.

The performance of RM-FFDU algorithm is $\Re_{RM-FFDU} \leq 5/3$, and its computational complexity is $O(n\ log\ n)$, which is influenced by the ordering of the tasks.

Previous to the RM-FFDU algorithm, Davari and Dhall [12] developed the First Fit decreasing utilization factor algorithm (FFDUF), which is similar to RM-FFDU with the only difference that it uses condition L&L for the admission control. The performance of the FFDUF algorithm is $\Re_{FFDUF} = 2$ and its complexity is $O(n\ log\ n)$.

Input: Task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$
and a set of processor $\{P_1, \ldots, P_m\}$.
Output: j; number of processors required.

```
1. Order tasks nonincreasingly by their utilizations
2. i := 1; j := 1; /* i = i^th task, j = j^th processor */
3. while (i ≤ n) do
4.     q := 1;
5.     while (u_i > 2/ ∏_{h=1}^{k_m} ( u_mh + 1)) do
6.         q := q + 1;
7.     U_q := U_q + u_i;    k_q := k_q + 1;
8.     if (q > j) then
9.         j := q;
10.    i := i + 1;
11.return (j);
12.end
```

Fig. 7. Rate Monotonic First Fit Decreasing Utilization (RM-FFDU) Algorithm

5) Rate Monotonic Small Tasks (RMST): Rate Monotonic Small Tasks (RMST) [10], illustrated in Figure 8, is an algorithm that favors task sets with small utilization factors (i.e., $u_i \leq 1/2$). RMST allocates tasks to processors in a similar way than that of the Next-Fit heuristic. The idea behind algorithm RMST is to partition the task set such that on each processor, variable β (defined in Theorem 3[10]) yields small values.

In RMST, before the allocation, tasks are sorted increasingly according to $V_i = log_2\ T_i - \lfloor\ log_2\ T_i\ \rfloor$ (step 1). Then, task τ_i is allocated to processor P_j only if this task together with the tasks previously allocated to P_j can be feasibly scheduled according to condition PO (steps 6 and 7). Otherwise task τ_i is allocated to an idle processor (steps 9 and 10).

Note that β_j denotes the difference of the V_i values on each processor. Therefore when $(1 - \beta_j\ ln2) \rightarrow 1$, then $\beta_j \rightarrow 0$.

The performance of algorithm RMST is $\Re_{RMST} \leq 1/(1 - \alpha)$ [10], and its computational complexity is bounded by $O(n\ log\ n)$.

Input: Task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$
and a set of processors $\{P_1, \ldots, P_m\}$.
Output: j; number of processors required.

```
1. Obtain V_i := log_2 T_i − ⌊ log_2 T_i ⌋ for all tasks
and sort the task set such that 0 ≤ V_i ≤ ... ≤ V_n < 1
2. i := 1; j := 1; /* i = i^th task, j = j^th processor */
3. V := V_i; β_j := 0; U_j := u_i; i := i + 1;
4. while (i ≤ n) do
5.     β_j := V_i − V; /* β_j value is updated */
6.     if (U_j + u_i ≤ max {ln 2, 1 − β_j ln 2}) then
7.         U_j := U_j + u_i; /* Task τ_i to processor P_j */
8.     else
9.         j := j + 1; V := V_i; β_j := 0;
10.        U_j := U_j + u_i; /* Task τ_i to processor P_{j+1} */
11.    i := i + 1;
12.return(j);
13.end
```

Fig. 8. Rate Monotonic Small Tasks (RMST) Algorithm

6) Rate Monotonic General Tasks (RMGT): The RMGT algorithm [10], summarized in Figure 9, was developed with the purpose of improving the performance of algorithm RMST. This algorithm can be applied to task sets that do not have the restriction of having small utilization factors.

Algorithm RMGT partitions a task set in two groups, such that the utilizations of the first and second group satisfies that $u_i \leq 1/3$ and $u_i > 1/3$ respectively. The allocation of the tasks from the first group is performed using the RMST algorithm. The First-Fit algorithm is used for the allocation of the tasks from the second group, with the restriction that at most two tasks can be allocated to each processor, using the Le schedulability condition [24].

The performance of algorithm RMGT is $\Re_{RMGT} = 7/4$. The allocation of the tasks in the first group has a complexity $O(n_1\ log\ n_1)$, where n_1 is the number of tasks from the first group. Since the First-Fit algorithm can be implemented with complexity $O(n_2\ log\ n_2)$ where n_2 is the number of tasks from the second group G_2, then it can be concluded that the computational complexity of algorithm RMGT is bounded by $O(n\ log\ n)$.

7) RBOUND-MP: Algorithm RBOUND-MP, summarized in Figure 10, uses condition RBOUND as follows. Algorithm $ScaleTaskSet$ (illustrated in Figure 3), is first applied to transform the original task set. Once the task set is transformed it is allocated to processors using the First-Fit heuristic. First-Fit uses the RBOUND condition as an admission control mechanism (i.e., to decide whether a new task can be accepted or not to execute in some processor). Finally, each original task is allocated to the processor where its transformed task was allocated.

Considering a task set with n tasks and m processors for finding a feasible allocation using algorithm RBOUND-MP, the computational complexity of the algorithm is bounded by $O(n(m + log\ n))$, where m denotes the number of tasks and p the number of processors. This is explained as follows. Algorithm $ScaleTaskSet$ requires $O(n\ log\ n)$, while First-Fit requires $O(mn)$ to allocate m transformed tasks to p

Input: A task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$
and a set of processors $\{P_1, \ldots, P_m\}$.
Output: j; number of processors required.

1.Partition the task set in two groups:
$G_1 := \{ \tau_i \mid u_i \leq 1/3\}$; $G_2 := \{ \tau_i \mid u_i > 1/3\}$;
2.Allocate tasks from the first group using RMST.
3.Allocate tasks from the second group as follows:
4.$i := 1$; $U_m := u_i$;
5.$i := i + 1$;
6.Use heuristic First-Fit to find a processor k
that executes only one task (e.g, task τ_w)
and satisfies the following conditions:
$\lfloor T_w/T_i \rfloor (T_i - C_i) \geq C_w$ or $T_w \geq \lceil T_w/T_i \rceil C_i + C_w$ if $T_i < T_w$
$\lfloor T_i/T_w \rfloor (T_w - C_w) \geq C_i$ or $T_i \geq \lceil T_i/T_w \rceil C_w + C_i$ if $T_i \geq T_w$
7.**if** such processor exists **then**
8. $U_k := U_k + u_i$; /* Allocate task τ_i to processor P_k
and set it as fully utilized
9.**else**
10. $U_l := u_i$; /* Allocate task τ_i to an
idle processor with index l. */
11**goto** 5
12Terminate when all tasks from group G_2
are allocated.
13**return**(j) /* j = number of processors required in
sets G_1 and G_2*/
14**end**

Fig. 9. Rate Monotonic General Tasks (RMGT) Algorithm

processors. Finally $O(n)$ is needed for the mapping of the original task set to the processors.

In [21] algorithm RBOUND-MP has been extended to include Fault Tolerance. The performance of RBOUND-MP has not yet been computed.

Input: A set of tasks τ.
Output: number of processors required to schedule τ.

1. ScaleTaskSet(τ)
2. The task set τ' is allocated using FF and
using the schedulability condition RBOUND.
3. Each task from the τ task set is allocated
to the processor at which its transformed task τ'
was allocated
4. Return(number of processors required)

Fig. 10. RBOUND-MP Algorithm

8) Rate Monotonic General Tasks M (RMGT/M): Algorithm RMGT/M [9], summarized in Figure 11, is the on-line version (with on-line task allocation) of algorithm RMGT. Recall that the on-line task allocation does not require that the entire task set is known a priori. Rather, on-line schemes provide procedures for dynamically adding new tasks and deleting existing tasks at any time. In algorithm RMGT/M, M is a parameter denoting the number of processors to which a new task can be allocated. This algorithm is based on schedulability condition PO.

In RMGT/M, each task is allocated to one of a fixed number of M classes. The class membership, say k, of a task τ_i is determined by the following expression.

$$k = \lfloor M \, (log_2 \, (T_i) - \lfloor log_2 \, (T_i) \rfloor) \rfloor + 1 \qquad (13)$$

For each class, the algorithm keeps an index, which corresponds to the current processor $P_{curr()}$ and each processor allocates tasks from the same class. If a new task from class k is added to the task set, then the algorithm first attempts to allocate it to processor $P_{curr()}$ in the k^{th} class (step 3). If this task can be feasibly scheduled on the current processor, then it is allocated (step 4). Otherwise, it is allocated to an idle processor (step 6), which will be called now the current processor. Note that $U_{curr(k)}$ denotes the total utilization of the current processor on class k and $newproc()$ computes the index of an idle processor.

The performance of the RMGT/M is $\Re_{RMGT/M} = \frac{1}{1-(ln\ 2)/M}$, and its computational complexity is $O(n)$.

Input: A new task τ_i.
Output: j; number of processors required to schedule the task set τ.

1. **begin**
2. $k := \lfloor M \, (log_2 \, (T) - \lfloor log_2 \, (T) \rfloor) \rfloor + 1$;
3. **if** $(U_{curr(k)} + u_i \leq 1 - (ln\ 2)/M)$ **then**
4. $U_{curr(k)} := U_{curr(k)} + u_i$;
5. **else if** $(U_{curr(k)} < u_i)$ **then**
6. $curr(k) := newproc()$; $j := j + 1$;
7. $U_{curr(k)} := u_i$;
8. else
9. $x := newproc()$; $U_x := u_i$;
10.$return(j)$;
11. **end**

Fig. 11. On-line Task Allocation using the Rate Monotonic General Tasks M (RMGT/M) Algorithm

9) RMNF-L&L, RMFF-L&L and RMBF-L&L Algorithms: In previous sections, the RMNF, RMFF and RMBF algorithms were described. These algorithms use the schedulability condition IP. A similar scheme to these algorithms is introduced by Oh and Son in [34], with the difference that it uses the L&L schedulability condition.

The RMFF-L&L, RMNF-L&L and RMBF-L&L algorithms can be used as on-line algorithms without ordering of the task set previous to the allocation.

The performance of the RMNF-L&L, RMFF-L&L algorithms is $\Re_{RMNF-L\&L} = 2/ln\ 2$, $\Re_{RMFF-L\&L} \leq 2 + (3 - 2^{3/2}) \ / \ (2\ (2^{1/3} - 1)) \approx 2.33$, respectively [34]. The RMBF-L&L algorithm has the same performance as that of RMFF-L&L. RMNF-L&L has a computational complexity of $O(n)$, while RMFF-L&L and RMBF-L&L have a computational complexity of $O(n\ log\ n)$.

D. EDF under the Partitioning Scheme

In this section, we introduce heuristic algorithms executing EDF under the partitioning scheme. As in RM, a set of algorithms can be developed integrating an allocation scheme (using Bin-Packing heuristics) along with a schedulability bound for EDF.

1) Earliest Deadline First Next Fit (EDF-NF): The first heuristic algorithm we introduce here is the Earliest Deadline First Next Fit (EDF-NF) [31].

In the algorithm, summarized in Figure 12, task τ_i is allocated to processor P_j (step 4 of the algorithm) as long as the total utilization of the processor do not exceed 100% (step 3). Otherwise, task τ_i is allocated to processor P_{j+1}.

According to the notation used in Figure 12, the allocation of task τ_i to processor P_j is done by the increase of utilization on that processor, $U_j = U_j + u_i$, where U_j denotes the utilization of processor P_j, and u_i the utilization of task τ_i.

After task τ_i is allocated, the next step is to allocate the next task, (step 8). Steps 2 to 8 from the algorithm, are repeated until all tasks are allocated. When the algorithm finishes, the total number of processors required for the feasible allocation of the task set, is given by variable j. The computational complexity of the algorithm is $O(n)$.

Input: A task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ and a set of processors $\{P_1, \ldots, P_m\}$.
Output: j; number of processors required.

1. $i := 1$; $j := 1$; /* $i = i^{th}$ task, $j = j^{th}$ processor */
2. **while** $(i \leq n)$ **do**
3. **if** $((U_j + u_i) \leq 1)$ **then**
4. $U_j := U_j + u_i$;
5. **else**
6. $U_{j+1} := U_{j+1} + u_i$;
7. $j := j + 1$;
8. $i := i + 1$;
9. **return**(j);
10. **end**

Fig. 12. Earliest Deadline First Next Fit Algorithm (EDF-NF)

Input: A task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ and a set of processors $\{P_1, \ldots, P_m\}$.
Output: j; number of processors required.

1. $i := 1$; $j := 1$; $k_q = 0$; $(\forall_q)$
2. **while** $(i \leq n)$ **do**
3. $q := 1$;
4. **while** $((U_q + u_i) > 1)$ **do**
5. $q := q + 1$; /* increase the processor index */
6. $U_q := U_q + u_i$; $k_q := k_q + 1$
7. **if** $(q > j)$ **then**
8. $j := q$;
9. $i := i + 1$;
10. **return**(j);
11. **end**

Fig. 13. Earliest Deadline First - First Fit Algorithm (EDF-FF)

2) Earliest Deadline First - First Fit (EDF-FF): In the EDF-FF Algorithm [31], before allocating task τ_i to processor P_j it is required first to find the smallest index j such that task τ_i, along with all tasks previously allocated to this processor, finds a feasible allocation. Algorithm EDF-FF is illustrated in Figure 13. In this algorithm, if task τ_i satisfies the schedulability condition shown in step 4, then task τ_i is schedulable. That is, task τ_i is allocated to processor P_m in step 6. k_m denotes the number of tasks previously allocated to processor P_m, and U_m is the total utilization of the k_m tasks. u_i denotes the utilization of task τ_i. If $i > n$ (if all tasks in the set have already been allocated) then the algorithm stops (step 2), otherwise the task index is increased (step 9), and the algorithm continues with steps 3 to 9. When the algorithm finishes, the total number of processors required for the feasible allocation of the task set, is given by variable j.

The computational complexity of the algorithm is $O(n\ log\ n)$, and its performance is $\Re_{EDF-FF} \leq 1.7$[15].

3) Earliest Deadline First Best Fit (EDF-BF): In algorithm EDF-BF, shown in Figure 14, tasks are allocated to the processor with the smallest available utilization, where the allocation is feasible. In order to allocate task τ_i (step 3 from the algorithm), it is required to find the smallest j (processor index) such task τ_i along with all tasks already allocated to processor P_j yield a feasible allocation, and value $1 - U_j$ is as small as possible. Task τ_i is allocated to processor P_j in step 4, where k_j denotes the number of tasks already allocated to processor P_j and U_j denotes the utilization of the k_j tasks. The number of processors used in this algorithm is given by variable j. The computational complexity of EDF-BF is $O(n\ log\ n)$. and its performance is $\Re_{EDF-BF} \leq 1.7$[15].

Input: Task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ and a set of processors $\{P_1, \ldots, P_m\}$.
Output: j; number of processors required.

1. $i := 1$; $j := 1$; /* $i = i^{th}$ task, $j = j^{th}$ processor */
2. **while** $(i \leq n)$ **do**
3. **if** $((U_j + u_i) \leq 1$) **and**
$(1 - U_j)$ is the smallest possible
4. $U_j := U_j + u_i$;
5. **else**
6. allocate task τ_i to an idle processor.
7. $i := i + 1$;
8. **return**(j);
9. **end**

Fig. 14. Earliest Deadline First Best Fit Algorithm (EDF-BF)

Input: Task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ and a set of processors $\{P_1, \ldots, P_m\}$.
Output: j; number of processors required.

1. $i := 1$; $j := 1$; /* $i = i^{th}$ task, $j = j^{th}$ processor */
2. **while** $(i \leq n)$ **do**
3. **if** $((U_j + u_i) \leq 1$) **and**
$(1 - U_j)$ is the greatest possible
4. $U_j := U_j + u_i$;
5. **else**
6. task τ_i is allocated to an idle processor.
7. $i := i + 1$;
8. **return**(j);
9. **end**

Fig. 15. Earliest Deadline First - Worst Fit Algorithm (EDF-WF)

4) Earliest Deadline First - Worst Fit (EDF-WF): Algorithm EDF-WF is illustrated in Figure 15. In this algorithm, tasks are allocated to the processor with the highest available utilization, where this task can be feasibly allocated.

According to this algorithm, in order to allocate task τ_i (step 3) the smallest j index must be found, such that task τ_i along with all tasks already allocated to processor P_j finds a feasible allocation, and the value of $1 - U_j$ is the smallest possible. Task τ_i is allocated to processor P_j in step 4, where k_j denotes the number of tasks already allocated to processor P_j, and U_j denotes the total utilization of the k_j tasks. The number of processors used in this algorithm is given by variable j.

Note that the available utilization on each processor is $1 - U_j$. The computational complexity of EDF-WF is $O(n\ log\ n)$. The performance of EDF-WF has not yet been computed.

5) EDF-NFD, EDF-FFD, EDF-BFD, EDF-WFD, EDF-NFI, EDF-FFI, EDF-BFI and EDF-WFI Algorithms: When a task set is ordered according to a parameter, before the allocation process, some variants of the heuristic algorithms

are produced. For instance, EDF-NFD (Earliest Deadline First Next Fit Decreasing) is an algorithm where the task set is ordered in a non-increasing fashion according to their utilizations. In contrast in the EDF-NFI algorithm (Earliest Deadline First - Next Fit Increasing) the task set follows a non-decreasing order of their utilizations. The same occurs for algorithms EDF-FF, EDF-BF and EDF-WF. The computational complexity of these algorithms is $O(n\ log\ n)$, which is influenced by the ordering of the tasks. The implementation of these algorithms is restricted to static environments because all the tasks in the set must be known a-priori. A detailed analysis of these algorithms is conducted in [31].

E. Complexity and Performance of the Partitioning Algorithms

In Table I the complexity and performance of the partitioning algorithms is introduced. Note that the algorithms with lowest complexity are RMNF-L&L, RMGT/M, and EDF-NF, while the algorithm with highest complexity is RBOUND-MP. The rest of the algorithms have complexity $O(n\ log\ n)$. The algorithms with best theoretical performance are RM-FFDU, RMST, RMGT, RMGT/M, EDF-FF and EDF-BF.

Algorithm	Condition	Complexity	$\Re_A$
RMNF [14]	IP	$O(n\ log\ n)$	2.67
RMFF [35]	IP	$O(n\ log\ n)$	2.33
RMBF [35]	IP	$O(n\ log\ n)$	2.33
RM-FFDU [33]	UO	$O(n\ log\ n)$	5/3
FFDUF [12]	L&L	$O(n\ log\ n)$	2.0
RMST [10]	PO	$O(n\ log\ n)$	1/1-α
RMGT [10]	PO and Le	$O(n\ log\ n)$	7/4
RBOUND-MP [21]	RBOUND	$O(n(m+log\ n))$	N/A
RMNF-L&L [34]	L&L	$O(n)$	2.88
RMFF-L&L [34]	L&L	$O(n\ log\ n)$	2.33
RMBF-L&L [34]	L&L	$O(n\ log\ n)$	2.33
RMGT/M [9]	PO	$O(n)$	$\frac{1}{(1-\alpha)}+\frac{1}{1-(ln\ 2)/M}$
EDF-FF [15]	$U \leq 1$	$O(n\ log\ n)$	1.7
EDF-BF [15]	$U \leq 1$	$O(n\ log\ n)$	1.7
EDF-WF [31]	$U \leq 1$	$O(n\ log\ n)$	N/A
EDF-NF [31]	$U \leq 1$	$O(n)$	N/A

TABLE I
COMPLEXITY AND PERFORMANCE OF THE MULTIPROCESSOR PARTITIONING ALGORITHMS

F. Utilization Bounds for Partitioned Algorithms on Fixed Number of Processors

Most of the research work about RM and EDF scheduling on multiprocessors focus on searching for heuristic allocation algorithms which are compared to each other using the performance metric $\Re_A = N_A/N_{opt}$. This metric is useful when there is a need for comparing the performance of different allocation multiprocessing algorithms but not to establish the schedulability of the system. There are several reasons [30]:

- In general, the number N_{opt} cannot be obtained in polynomial time.
- Even if N_{opt} were known, the utilization bound derived from the metric is too pessimistic, as is shown by Oh and Baker [32].

Recall that the aim of the previously described allocation algorithms was to find the minimum number of processor, assuming that the number of processors available on the system was *infinite*; an assumption not realistic on any real-time application.

The objective in this section is to introduce well known utilization bounds using RM and EDF scheduling, on which the number of processors is *fixed*. With this purpose, a new parameter m, indicating the number of processors, will be included to the utilization bound.

1) Rate Monotonic Utilization Bounds: Dhall and Liu [14] introduced the first known RM utilization bound for Multiprocessors, with the development of the Rate Monotonic First Fit Multiprocessor algorithm,

Lemma 2: [14]: If $m > 3$ tasks cannot be feasibly scheduled on $m-1$ processors according to the RMFF scheduling algorithm, then the utilization factor of the set of tasks is greater than $m/(1+2^{1/3})$.

Oh and Son[34], developed the RMFF-L&L and RMBF-L&L algorithms and showed that both utilization bounds are similar.

Lemma 3: [34]: If m tasks cannot be feasibly scheduled on $m-1$ processors according to the RMFF-L&L scheduling algorithm, then the utilization factor of the set of tasks is greater than $m/(1+2^{1/2})$.

The utilization bounds developed by Dhall [14] and by Oh and Son [34] for RMFF and RMFF-L&L are shown in Figure 16(a).

Oh and Baker [32] have shown that the First-Fit partitioning algorithm under Rate Monotonic can schedule any periodic task set (τ) on $m \geq 2$ processors with a utilization $U(\tau) \leq m(2^{1/2}-1)$. They also showed that the worst-case achievable utilization $U_{min}(m)$ for a fixed-priority preemptive multiprocessor scheduling algorithm with $m \geq 2$ processors is bounded by

$$m(2^{1/2}-1) < U_{min}(m) \leq (m+1)/(1+2^{1/(m+1)}) \quad (14)$$

López *et al.*[30] obtained tight utilization bounds for RM using the allocation algorithms First-Fit, Best-Fit and Worst-Fit. Their results are shown in the following Theorems.

Theorem 8: [30]: If all the tasks have a utilization under a value α, and ($n > \beta_{RM}\ m$) the utilization bound for RMFF, RMBF, RM-FFI and RM-BFI is proved to be

$$U_{RM-WFI} = \begin{cases} mU_b - (m-1)\alpha & if\ \alpha \leq U_b \\ U_b & if\ U_b < \alpha \end{cases} \quad (15)$$

$$\begin{array}{l} U_{RMFF}(n,m,\beta_{RM}) = \\ (m-1)(2^{1/(\beta_{RM}+1)}-1)\beta_{RM} + \\ (n-\beta_{RM}(m-1))(2^{1/(n-\beta_{RM}(m-1))}-1) \end{array} \quad (16)$$

where $\beta_{RM} = \lfloor \frac{1}{log_2\ (\alpha+1)} \rfloor$, is the maximum number of tasks of utilization factor α which fit into one processor, and n is the number of tasks in the set.

The utilization bounds developed in [32], [30] are shown in Figure 16(b) and 16(c) respectivelly.

Theorem 9: [28]: If ($n > m\beta_{RM}$) the utilization bound RMWF is given by

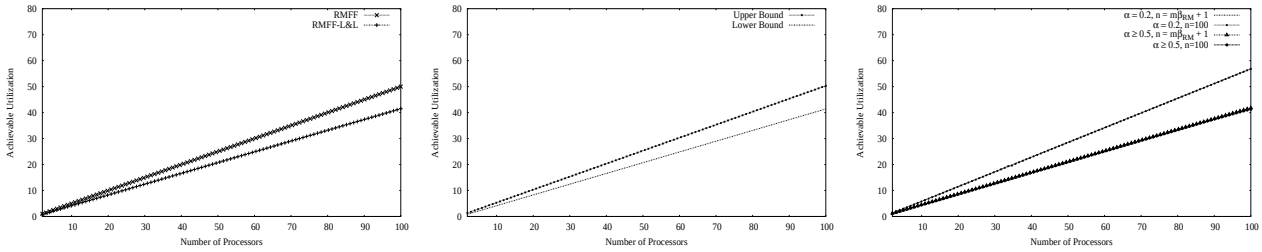

Fig. 16. (a). Dhall and Liu [14], Oh and Son [34], (b). Oh and Baker[32], (c) Lopez *et al.*[30]

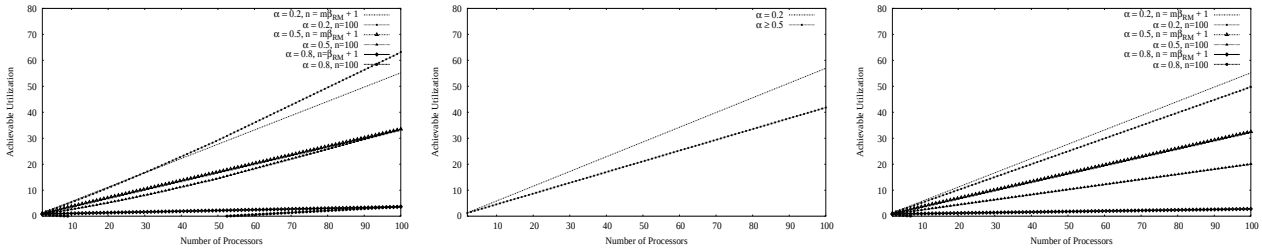

Fig. 17. (a). RMWF, (b). RM-FFD, (c) RM-WFI

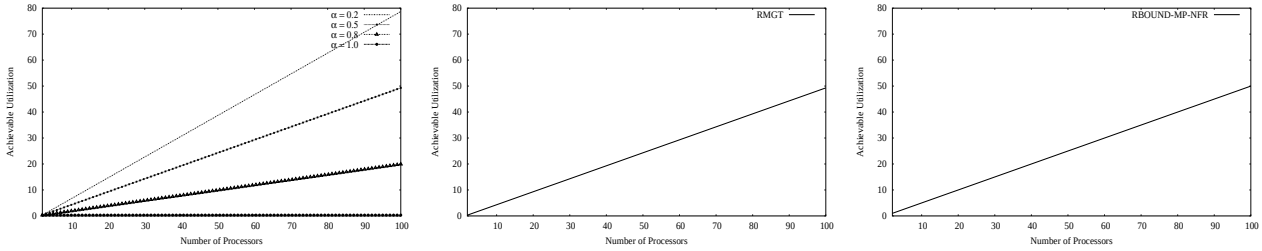

Fig. 18. (a). RMST Bound, (b). RMGT Bound, (c). RBOUND-MP-NFR Bound

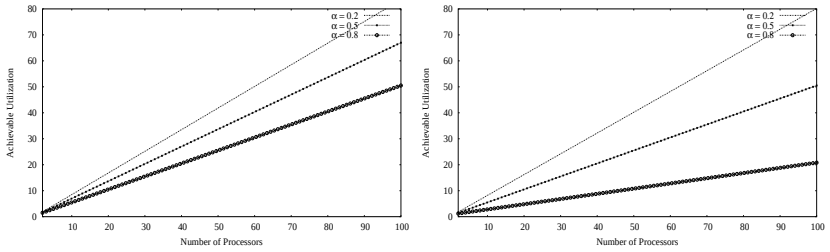

Fig. 19. (a). EDF-FF Bound, (b). EDF-WF Bound

$$U_{RMWF}(n,m,\alpha) = \begin{array}{ll} N_aU_a + N_bU_b - (m-1)\alpha & if\ \alpha < U_a \\ N_bU_b - (N_b-1)\alpha & if\ U_a \le \alpha \le U_b \\ U_b & if\ U_b < \alpha \end{array} \quad (17)$$

where

$$N_a = n - \lfloor \frac{n-1}{m} \rfloor m - 1 \quad (18)$$

$$N_b = m - N_a \quad (19)$$

$$U_a = \lceil \frac{n+m-1}{m} \rceil (2^{1/\lceil \frac{n+m-1}{m} \rceil} - 1) \quad (20)$$

$$U_b = \lfloor \frac{n+m-1}{m} \rfloor (2^{1/\lfloor \frac{n+m-1}{m} \rfloor} - 1) \quad (21)$$

where β_{RM} has been defined in Theorem 8.

Lopez *et al.* also developed utilization bounds for algorithms with task's ordering. Their results are shown in the following Theorems.

Theorem 10: [28]: If $n > m$ the utilization bound for RM-FFD, RM-BFD and RM-WFD is given by

$$\begin{array}{l} U_{RM-FFD}(m>1,\beta_{RM}) = U_{RM-BFD} = \\ U_{RM-WFD} = (\beta_{RM}+1)(2^{1/(\beta_{RM}m+1)} - 1) \end{array} \quad (22)$$

Theorem 11: [29]: If $(n > m\beta_{RM})$ the tight utilization bound RM-WFI is given by

$$U_{RM-WFI} = \left\{ \begin{array}{ll} mU_b - (m-1)\alpha & if\ \alpha \le U_b \\ U_b & if\ U_b < \alpha \end{array} \right. \quad (23)$$

where

$$U_b = \lfloor \frac{n+m-1}{m} \rfloor (2^{1/\lfloor \frac{n+m-1}{m} \rfloor} - 1) \quad (24)$$

where β_{RM} has been defined in Theorem 8.

The utilization bounds for RMWF,RM-FFD and RM-WFI are shown in Figure 17.

Burchard *et al.* developed utilization bounds in their RMST and RMGT algorithms [10]. They showed that in a system containing $m \geq 2$ processors, the schedulable utilization of the RMST algorithm is

$$U_{RMST} = (m-2)(1-\alpha) + 1 - ln\ 2 \quad (25)$$

This algorithm can always find a feasible allocation if the total utilization U of the task set is no greater than U_{RMST}. Contrary to RMST, the utilization bound of the RMGT algorithm does not depend on the utilization of the individual tasks (α). RMGT can achieve a feasible allocation on m processors whenever the total utilization of all periodic tasks is no greater than

$$U_{RMGT} = 0.5 \left(m - \frac{5}{2} ln\ 2 + \frac{1}{3} \right) \approx 0.5(m - 1.42) \quad (26)$$

The utilization bounds for RMST and RMGT are shown in Figure 18(a) and 18(b).

In [3], Andersson and Jonsson developed a utilization bound for algorithm RBOUND-MP. In their bound, called RBOUND-MP-NFR, Next-Fit task allocation is used, instead of the First-Fit used by RBOUND-MP.

Theorem 12: [3]: If RBOUND-MP-NFR is used its utilization bound is proved to be,

$$U_{RBOUND-MP-NFR} = m/2 \quad (27)$$

The utilization bound RBOUND-MP-NFR is shown in Figure 18(c).

From Figures 16 to 18 it is clear that the RMST Bound, with low values of α (i.e., $\alpha \leq 0.5$), provide the best performance among all bounds for Rate Monotonic.

2) EDF Utilization Bounds.: López *et al* computed the utilization bound for EDF, using the FF, BF and WF allocation algorithms, as shown in the following Theorems.

Theorem 13: [29]: If all the tasks have a utilization under a value α, and ($n > \beta_{EDF}\ \ m$) the utilization bound $U_{EDF-FF} = U_{EDF-BF}$, is proved to be

$$\begin{array}{l} U_{EDF-FF}(m > 1, \beta_{EDF}) = \\ U_{EDF-BF} = \frac{\beta_{EDF} m + 1}{\beta_{EDF} + 1} \end{array} \quad (28)$$

where $\beta_{EDF} = \lfloor \frac{1}{\alpha} \rfloor$ is the maximum number of tasks with utilization α that can be assigned to any processor and n the number of tasks in the set.

Theorem 14: [31]: If $n > (\beta_{EDF}\ \ m)$ the tight utilization bound for EDF-WF, EDF-WFI and EDF-WFD is given by

$$\begin{array}{l} U_{EDF-WF}(n, m, \alpha) = U_{EDF-WFI} = \\ U_{EDF-WFD} = m - (m-1)\ \alpha \end{array} \quad (29)$$

The utilization bounds for heuristics EDF-FFD, EDF-FFI, EDF-BFD, EDF-BFI and EDF-WFD are the same as for those of the heuristic EDF-FF [31]. The utilization bounds for EDF-FF and EDF-WF are shown in Figure 19. Note that the EDF utilization bounds of EDF-FF and EDF-WF outperform the RM utilization bounds.

V. Global Scheduling on Multiprocessors

In global scheduling, a task is put in a queue of ready tasks that is shared by all processors and, at every moment, the m highest priority task is selected for execution on the m processors. Contrary to the partitioning scheme, in global scheduling tasks execute on one processor for only one instance, and then return to the global queue to be allocated to possibly another processor. Therefore in the global scheme, migration is allowed.

Since global scheduling does not reduce the multiprocessor scheduling problem, to *many* uniprocessor scheduling problems, as partitioned scheduling does, and because tasks in the global scheme are allowed to migrate, this gives rise to many unexpected effects and disadvantages that complicate the design of scheduling and allocation algorithms for the global scheme.

- In the global scheme, the migration of tasks to processors introduce high overhead to the system.
- Since migration is allowed, the information flow between processors tends to be high. This communication may require the use of shared memory of communication channels.
- Predictability associated to this scheme is much lower than that associated to the partitioned scheme.
- In this scheme the *Dhall effect* may occur, and tasks sets with small utilization may be unschedulable [14].
- Some scheduling anomalies may occur in this scheme, such as those described in [2], [22], [1]

An advantage of the global scheme is its generality. Since tasks can migrate from one processor to another, the processor system "could be" better utilized.

A. Scheduling Anomalies under Global and Partitioned Scheduling

1) Dhall Effect: The *Dhall effect* introduced by Dhall and Liu [14], describes a problem that occur in the scheduling of real-time tasks in multiprocessor system under RM and EDF.

This effect is illustrated in the following example [14]:

Example 1. *Consider $m+1$ periodic tasks that should be scheduled on m processors using RM. Let task τ_i (where $1 \leq i \leq m$) have $T_i = 1$, $C_i = 2\epsilon$, and task τ_{m+1} have $T_{m+1} = 1 + \epsilon$, $C_{m+1} = 1$. All tasks arrive at the same time (e.g., at time t=0). Tasks τ_i will execute immediately when they arrive and complete their execution 2ϵ time units later. Task τ_{m+1} then executes from time 2ϵ until time 1, that is, $1 - 2\epsilon$ time units. Since, task τ_{m+1} needs to execute 1 time unit, it will miss its deadline. By letting $m \to \infty$ and $\epsilon \to \infty$, we have a task set with a system utilization near zero, but with a deadline missed.*

From this example, it can be concluded that RM is not optimal in global multiprocessor scheduling because it can perform poorly even when the utilization of the task set is too small. Similar behavior was observed for EDF [14].

2) Critical Instant Effect: In schedulability analysis, it is necessary to compute how many time units of execution, higher priority tasks can perform, during a time interval. In uniprocessor scheduling, it holds that during a time interval of length L, a task τ_i can execute at most $\lceil L/T_i \rceil \cdot C_i$ time units. However, in global multiprocessor scheduling, this number can be higher, as illustrated in the following example[2].

Example 2. *Consider the following five periodic tasks scheduled in two processors: ($T_1 = 5, C_1 = 3$), ($T_2 = 5, C_2 = 1$), ($T_3 = 6, C_3 = 2$), ($T_4 = 11, C_4 = 4$), ($T_5 = 10, C_5 = 2$). Here, we assume that τ_3 arrives at time 1 and task τ_5 arrives at time 5 (and all other tasks arrive at time 0). For this case, the amount of execution from the four high priority tasks in the interval $[5, 15)$ are: 6 for τ_1, 2 for τ_2, 4 for τ_3, and 6 for τ_4. Task τ_5 is delayed by 9 time units due to the execution of higher priority tasks, which causes task τ_5 to miss its deadline at time 15 (since $T_5 = 10$ and $C_5 = 2$). In this case, more than one instance of τ_4 will contribute to the amount of execution that delays task τ_5 in that interval, even when the period of τ_4 is longer than that of τ_5.*

In global multiprocessor scheduling, it is not only the amount of execution of higher priority tasks the reason for the delay of lower priority tasks, but also the delay depends on whether these higher priority tasks execute at the same time. This leads to additional scheduling effects, reported in [22], [1], for which the assumptions for RM on uniprocessors do not hold on global multiprocessor scheduling.

Observation 1: (**Critical Instant for Uniprocessor**)[27] For static preemptive uniprocessor systems, the critical instant is defined to be the instant at which a request for a task will have the largest response time. The critical instant occurs when all tasks arrive at the system at the same time.

From this observation, in [27] concluded that if each task in the system meets its first deadline when a critical instant occurs, then the task set is schedulable.

In the following observation, we will show that the critical instant for global multiprocessor scheduling does not occur when all tasks arrive simultaneously at the system.

Observation 2: (**Critical Instant for Multiprocessors**) [22], [1] For static priority preemptive global multiprocessor scheduling, there exist task sets where a critical instant of one of the tasks does not occur when it arrives at the same time as its higher priority tasks.

Example 3.[1] *Consider the following three periodic tasks: ($T_1 = 2, C_1 = 1$), ($T_2 = 3, C_2 = 2$), ($T_3 = 4, C_3 = 2$). These tasks can be scheduled on two processors. The first instance of task τ_3 has a response time of $R_{3,1} = 3$, when it arrives at the same time as τ_1 and τ_2. However, the second instance of τ_3 has a response time of $R_{3,2} = 4$ (which is higher than the response time of task τ_3 in its first instance) although τ_1 and τ_2 do not arrive at the same time as τ_3.*

The implication from this observation is that it is not enough to guarantee that all tasks do meet their deadline when they arrive simultaneously, since later deadlines are not always guaranteed. Therefore, in the worst case it may be necessary to check that no task misses its deadline for the first $LCM(T_i) + max(T_i)$ time units, where $LCM(T_i)$ is the least common multiplier of the periods of all tasks. Checking the first $LCM(T_i) + max(T_i)$ time units will add overhead to the admission control in the global scheduler, and thus a stronger condition will be needed [22].

3) Period Anomalies in Global Scheduling: In the scheduling of uniprocessor real-time systems it is usual to assume that if tasks periods are increased it will cause a decrease in the system utilization. The same effect in utilization decrease would be the result of reducing the execution time of some tasks. In both cases, it is clear that no tasks would miss their deadline for this utilization decrease. Contrary to the assumptions made on uniprocessor systems, in multiprocessor scheduling the above assumptions may not hold.

One anomaly that may occur in multiprocessor scheduling is that an increase in the period causes tasks to arrive at different times. This increase in the period does not affect the schedulability of the system directly, because the total utilization of the system decreases. However, the tasks may be allocated differently. This change in allocation of the tasks to processors causes more instants when all processors are busy, and these delays may affect the schedulability of lower priority tasks.

Observation 3: (**Period Anomalies in Global Scheduling**[1]) For static-priority preemptive global multiprocessor scheduling, there exists task sets that meet all deadlines with one priority assignment, but if the period of a task increases and the priorities remain the same then a task misses its deadline.

Example 4[1]: *Consider the following three periodic tasks: ($T_1 = 3, C_1 = 3$), ($T_2 = 4, C_2 = 2$), ($T_3 = 12, C_3 = 7$). These tasks can be scheduled on two processors. Here, if we increase the period of task τ_1, from 3 to 4 (but not change the relative priority order), the resulting task set misses its deadline.*

In [1] Andersson discussed several other period and execution-time anomalies, for static and dynamic global scheduling as well as for the partitioned scheme.

B. Global Multiprocessor Scheduling under RM

1) Global Rate Monotonic Schedulability Condition: Lauzac *et al.*[22] introduced a polynomial-time necessary schedulability condition for global scheduling under RM. Before using this condition, tasks are ordered increasingly according to their periods.

Theorem 15: **(Global Rate Monotonic Scheduling)** [22] : If the task set $\{\tau_1, \tau_2, \ldots, \tau_{i-1}\}$ is schedulable on m processors under Global Rate Monotonic Scheduling, then task τ_i is schedulable with other $i-1$ tasks if:

$$C_i \leq (mT_i - \sum_{j=1}^{i-1}(\lfloor \frac{T_i}{T_j} \rfloor + 2)C_j)/m \tag{30}$$

2) AdaptiveTkC Algorithm: Andersson and Jonsson [1] developed the AdaptiveTkC algorithm in which the Dhall effect is avoided.

Theorem 16: **(AdaptiveTkC)** [1]: Consider a system with $m \geq 2$ processors and $n = m+1$ tasks. The m highest priority tasks have the same period and execution time. The tasks are sorted with respect to $T_i - k \cdot C_i$ and the task with the least

$T_i - kC_i$, obtains the highest priority. The following value of k avoid the Dhall effect and maximizes the least system utilization of fully utilized task sets.

$$k = \frac{1}{2}\ \frac{m - 1 + \sqrt{5m^2 - 6m + 1}}{m} \tag{31}$$

and the corresponding least system utilization of fully utilized task sets is:

$$U_s = 2\ \frac{m}{3m - 1 + \sqrt{5m^2 - 6m + 1}} \tag{32}$$

$U_s = U/m$ denotes the average system utilization on each processor. The lower bound of the system utilization is never lower than $lim_{m\to\infty} U_s > 0.38$, which shows that the Dhall effect is circumvented for the constrained task set. Note that when k = 0, AdaptiveTkC has the same priority assignment as in RM. It is worth to point out that for tasks sets that are not constrained by the same period and the same execution time, the least system utilization of fully utilized task sets is still not know.

3) Scheduling Algorithm RM-US[m/(3m-2)]: Andersson, Baruah and Jonsson [4] introduced the static priority global scheduling algorithm RM-US[m/(3m-2)] which schedules a task set using a utilization-based sufficient condition. Andersson *et. al.* demonstrated that algorithm RM-US[m/(3m-2)] schedules a task set as long as the total utilization of the task set $U \leq m^2/(3m - 2)$, where m denotes the number of processors.

Anderson *et al.* [4] demonstrated that a task set with $u_i \leq \frac{m}{3m-2}$, for $i = 1, ..., n$ (*light system*) is schedulable under RM. Also, they extended this results with task sets following a general condition $u_i \leq 1$, under the priority assignment scheme named $RM - US[m/(3m - 2)]$.

Definition 1: **Light Systems (RM)** [4]: A periodic task system τ is said to be a light system on m processors under RM, if it satisfies the following two properties.

Property 1. For each $\tau_i \in \tau, u_i \leq \frac{m}{3m-2}$.
Property 2. $U(\tau) \leq \frac{m^2}{3m-2}$.

Theorem 17: [4]: Any periodic task system τ that is light on m processors will be scheduled to meet all deadlines on m processors by algorithm RM.

Andersson [4] developed Algorithm RM-US[m/(3m-2)] which assigns static priorities to tasks in τ according to the following rule:

- **if** $u_i > \frac{m}{3m-2}$ **then** τ_i has the highest priority (ties broken arbitrarily).
- **if** $u_i \leq \frac{m}{3m-2}$ **then** τ_i has rate-monotonic priority.

Theorem 18: **(RM-US[**$m/(3m-2)$**])** [4]: Any periodic task system τ with utilization $U(\tau) \leq m^2/(3m - 2)$ will be scheduled to meet all deadlines on m identical processors by algorithm RM-US[m/(3m-2)].

A variant of the RM-US[$m/(3m - 2)$ algorithm for harmonic task sets [26] was introduced by Andersson *et al* [4], refereed as the RM-US[m/(2m-1)] algorithm. This algorithm guarantees that an harmonic task set will be schedulable if the system utilization is $U(\tau) \leq m^2/(2m - 1)$.

Theorem 19: **(RM-US[m/(2m-1)])** [4]: Any task set τ in which all the periods of the tasks are harmonic and with a utilization $U(\tau) \leq m^2/(2m - 1)$ will be scheduled to meet all their deadlines on m identical processors by algorithm RM-US[m/(2m-1)].

4) Baker Conditions under RM: Baker [5] defined several schedulability tests for Deadline Monotonic Scheduling (DMS) and EDF [5]. On DMS, task deadlines may be smaller than their periods, that is $D_i \leq T_i$.

Theorem 20: (DM Schedulability Test) [5]: A set of periodic tasks is schedulable on m processors using preemptive deadline-monotonic scheduling if, for every task τ_k,

$$\sum_{i-1}^{k-1} \beta_i \leq m(1 - \frac{C_k}{D_k}) \tag{33}$$

where β_i is defined as follows,

$$\beta_i = \begin{cases} \frac{C_i}{T_i}(1 + \frac{T_i - \delta_i}{D_k}) & if \alpha \geq \frac{C_i}{T_i} \\ \frac{C_i}{T_i}(1 + \frac{T_i - \delta_i}{D_k}) + \frac{C_i - \alpha T_i}{D_k} & if \alpha < \frac{C_i}{T_i} \end{cases} \tag{34}$$

where $\delta_i = C_i$ for $i < k$, and $\delta_k = D_k$, and $\alpha = max\{\frac{C_i}{D_i} \mid i = 1, ..., n\}$.

From the above Theorem, Baker defined a simplified DM schedulability test in the following Corollary.

Corollary 3: (Simplified DM test) [5]: A set of periodic tasks $\tau_1, ..., \tau_n$ is schedulable on m processors using preemptive DM scheduling if

$$\sum_{i=1}^{n} \frac{C_i}{T_i}\left(1 + \frac{Ti - \delta_i}{D_k}\right) \leq m(1 - \alpha) + \alpha \tag{35}$$

where $\delta_i = C_i$, for $i < k$, and $\delta_k = D_k$.

Baker [5] also defined an schedulability test for RM, as defined in the following Corollary.

Corollary 4: [5]: A set of periodic tasks, all with deadlines equal to their periods, is guaranteed to be schedulable on m processors, $m \geq 2$, using preemptive rate monotonic scheduling if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq \frac{m}{2}(1 - \alpha) + \alpha \tag{36}$$

C. Global Multiprocessor Scheduling under EDF

In this section we will introduce the algorithms developed for global multiprocessing scheduling under EDF.

1) Scheduling Algorithm $EDF-US[m/(2m-1)]$*:* In [38], Srinivasan and Baruah defined a multiprocessor scheduling algorithm based on the following properties. Let $\tau = \{\tau_1, ...\tau_n\}$ be a light system on m processors if it satisfies the following properties:

Definition 2: **Light Systems (EDF)** [4]: A periodic task system τ is said to be a light system on m processors under EDF, if it satisfies the following two properties.

Property 1. $\frac{C_i}{T_i} \leq \frac{m}{2m-1}$, for $1 \leq i \leq n$.
Property 2. $\sum_{i=1}^{n} \frac{C_i}{T_i} \leq \frac{m^2}{2m-1}$

Theorem 21: [38]: Any periodic task system that is light on m processors is scheduled to meet all deadlines on m processors by EDF

Theorem 21 shows that EDF schedules feasibly any periodic task system τ with utilization $m^2/(2m-1)$ on m identical processors, provided that each $\tau_i \in \tau$ has a utilization $u_i \leq m/(2m-1)$. In the same work, Srinivansan and Baruah, relaxed the restriction of property 1, and defined Algorithm $EDF-US[m/(2m-1)]$ as follows.

Algorithm $EDF-US[m/(2m-1)]$[38] assigns priorities to jobs of tasks in τ according to the following rule:

- **if** $u_i > m/(2m-1)$ **then** τ_i's jobs are allocated highest priority (ties broken arbitrarily)
- **if** $u_i \leq m/(2m-1)$ **then** τ_i's jobs are allocated priorities according to EDF

Note that when $m = 1$, it reduces to EDF, since U_i can never be greater than 1.

Theorem 22: [38]: Algorithm $EDF-US[m/(2m-1)]$ correctly schedules on m processors any periodic task system τ with utilization $U(\tau) \leq m^2/(2m-1)$.

2) Goossens et al *Conditions under EDF:* Goossens *et al* [17] developed an schedulability condition under EDF, showed in the following Corollary.

Corollary 5: [17]: A set of periodic tasks, $\tau_1, \tau_2, ...\tau_n$ all with deadline equal to period, is guaranteed to be schedulable on m processors, using preemptive EDF scheduling if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq m(1-\alpha)+\alpha \tag{37}$$

where $\alpha = max\{\frac{C_i}{T_i} \mid i = 1, ..., n\}$.

3) Baker Conditions under EDF: Baker [5] introduced the following schedulability conditions for tasks sets under EDF.

Theorem 23: (EDF Schedulability Test) [5]: A set of periodic tasks is schedulable on m processors using preemptive EDF scheduling if, for every task τ_k,

$$\sum_{i-1}^{n} min\{1, \beta_i\} \leq m(1-\frac{C_k}{D_k})+\frac{C_k}{D_k} \tag{38}$$

where β_i is defined as follows,

$$\beta_i = \begin{cases} \frac{C_i}{T_i}(1+\frac{T_i-D_i}{D_k}) & if \alpha \geq \frac{C_i}{T_i} \\ \frac{C_i}{T_i}(1+\frac{T_i-D_i}{D_k})+\frac{C_i-\lambda T_i}{D_k} & if \alpha < \frac{C_i}{T_i} \end{cases} \tag{39}$$

Corollary 6: (Simplified EDF Test)[5] A set of periodic tasks $\tau_i, ...\tau_n$ is schedulable on m processors using preemptive EDF scheduling if

$$\sum_{i=1}^{n} min\{1, \frac{C_i}{T_i}(1+\frac{T_i-D_i}{D_{min}})\} \leq m(1-\alpha)+\alpha \tag{40}$$

where $D_{min} = min\{D_k \mid i = 1, ..., n\}$.

Baker [5] assumed that $C_i \leq D_i \leq T_i$.

4) Scheduling Algorithm fpEDF: The fpEDF algorithm [8], proved that any periodic task system with utilization at most $(m+1)/2$ can be scheduled by algorithm fpEDF to meet all the deadlines on m processors.

Theorem 24: (EDF Schedulability Test) [8]: Any periodic task system τ satisfying the following two properties:

Property P1 $\alpha \leq 1/2$
Property P2 $\sum_{i=1}^{n} u_i \leq \frac{m+1}{2}$

is correctly scheduled to meet all deadlines on m processors under EDF.

Input: Task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$
and a set of processors $\{P_1, \ldots, P_m\}$.
1. Order tasks decreasingly by their utilizations
2. **for** $(i = 1)$ **to** $(m-1)$ **do**
3. **if** $(u_i > 1/2)$ **then**
4. $\tau_i's$ jobs are allocated highest priority
5. **else break;**
6. the remaining task's jobs allocate priorities according to EDF
7. **end**

Fig. 20. Priority assignment rule of Algorithm fpEDF

Algorithm fpEDF [8] is illustrated in Figure 20, with its corresponding sufficient schedulability condition, shown in Theorem 25.

Theorem 25: [8]: Algorithm **fpEDF** guaranteed that the task set is schedulable upon m processors if $\sum_{i=1}^{n} u_i \leq (m+1)/2$.

Additionally, Baruah [8] extended the schedulability condition of Theorem 25 to include the α value, in the following Theorem.

Theorem 26: [8]: Algorithm **fpEDF** guaranteed that the task set is schedulable upon m processors if $\sum_{i=1}^{n} u_i \leq max\left(m-(m-1)\alpha, \frac{m}{2}+\alpha\right)$.

VI. Simulation Experiments

The following simulation experiments have been designed to test the performance of the partitioning and global multiprocessor scheduling algorithms using synthetic task sets. The goal in this simulation experiments is to measure the quality of the results over a large set of experimental executions.

Each plot in the graphs represents the average of a set of 500 task sets. For each task set, the period P_i of each task follow a uniform distribution with values $1 \leq T_i \leq 500$. The execution time C_i of each task follow a uniform distribution with values $1 \leq C_i \leq \alpha T_i$, where α denotes the maximum utilization factor allowed for each task. Parameter α varies in the range $[0.2, 0.5, 0.8, 1.0]$. The purpose of these experiments is to observe the performance of the algorithms under different utilization factors.

We will consider two sets of experiments. The first experiment considers an execution with 16 processors. In this experiment, for each algorithm, the performance metric used is the *Guarantee Ratio*, which denotes the percentage of feasibly scheduled task sets. In this experiment, utilization is varied from 6 to 16. Since the partitioned and the global schemes

use different allocation approaches we will consider that a task set is schedulable on m processors under the partitioning scheme if no task in the set misses its deadline. For the global scheme, a feasible schedule denotes a schedule on which no task will miss its deadline during the interval of time $[0, lcm(T_1, T_2, ..., T_n)]$.

The aim in the second experiment is to compute the number of processors on which a task set can be feasibly scheduled. In this experiment, parameter α is also varied in the range $[0.2, 0.5, 0.8, 1.0]$, and the number of tasks is varied in the range $[100, ..., 1000]$ in incremental steps of 100 tasks. Although the maximum number of processors in this experiments could be considered *infinite*, we limited the experiment to 1000 processors. All experiments were conducted in a Dell 2.0 GHz PowerEdge 6600 Server Quad Xeon Processor with 2GBytes of RAM and 1 MByte of Cache memory.

A. Experiments with 16 processors

In Figures 21 to 32, the algorithms with better performance are showed in the right-most place. The order of performance of the algorithms is illustrated in the upper-left corner of the graphs.

Partitioned Algorithms

Figures 21 to 22 illustrate the experiments performed for partitioned off-line algorithms under RM on 16 Processors, under different values of α. From these graphs we can observe that under $\alpha = 0.2$ and $\alpha = 0.5$ the algorithms with better performance are RMGT and RMST. However, the performance of these algorithms degrades with higher values of α. Algorithm RM-FFDU yields the third place for $\alpha \leq 0.5$ and the second place for $\alpha > 0.5$, in performance among all algorithms. Algorithm RBOUND-MP improves its performance with higher values of α. Note that RBOUND-MP yield the best performance for values of $\alpha = 0.8$ and $\alpha = 1.0$.

Algorithms RMBF and RMFF achieve similar performance in almost all experiments. RMBF achieves performance places $5th$,$6th$,$4th$, and $4th$, for values of $\alpha = 0.2$, $\alpha = 0.5$, $\alpha = 0.8$, $\alpha = 1.0$ respectively. RMFF achieves performance place $7th$ for values $\alpha = 0.2$ and $\alpha = 0.5$, and a performance place $5th$ for values $\alpha = 0.8$ and $\alpha = 1.0$. Algorithm RMNF achieves the worst performance in all cases.

Figures 23 and 24 illustrate the experiments performed for partitioned on-line algorithms under RM on 16 Processors, under different values of α. From these graphs we can observe that RMGT/M is the best algorithm only for values of $\alpha = 0.2$, while for other cases RMGT/M is only better than algorithm RMNF-L&L. Algorithms RMBF-L&L and RMFF-L&L are the best algorithms for $\alpha > 0.2$. Note that RMBF-L&L always improves RMFF-L&L.

Figures 25 and 26 illustrate the experiments performed for partitioned off-line algorithms under EDF on 16 Processors, under different values of α. From these Figures it can be noted that the best performance in general is achieved when $\alpha = 0.2$. In all cases the best performance is achieved by EDF-BFD while the worst performance is always achieved by EDF-NFI. Note that for all values of α, algorithms EDF-BFD, EDF-FFD and EDF-WFD yield similar (and the best) performance.

Figures 27 and 28 illustrate the experiments performed for partitioned on-line algorithms under EDF on 16 Processors, under different values of α. In these experiments, for all cases the algorithm with best performance is EDF-BF followed by EDF-FF, EDF-WF and EDF-NF in that order. Note that algorithms EDF-BF and EDF-FF yield similar performance for values of $\alpha = 0.2$ and $\alpha = 0.5$.

Global Algorithms

Figures 29 to 30 illustrate the experiments performed for global algorithms under RM on 16 Processors, under different values of α. Algorithm AdaptiveTkC yields the best performance among all algorithms for all values of α. Note that algorithm RM-US[m/(3m-2)] yields higher (for $\alpha > 0.8$), lower (for $\alpha = 0.5$ and $\alpha = 0.8$) or equal (for $\alpha = 0.2$) performance than RM. Algorithm RM-US[m/(3m-2)] yields similar performance than RM for $\alpha = 0.2$.

Figures 31, to 32 illustrate the experiments performed for global algorithms under EDF on 16 Processors, under different values of α. From these Figures it can be noted that EDF yields similar performance than $EDF-US[m/(2m-1)]$ for values of $\alpha = 0.2$ and $\alpha = 0.5$, but for values of $\alpha = 0.8$ and $\alpha = 1.0$ the algorithm with best performance is $EDF-US[m/(2m-1)]$.

B. Experiments with 1000 processors

In Figures 33 to 44, we illustrate the second experiment. The algorithms with better performance are showed in the bottom-most part of the graphs. As discussed before, the best performance occurs when the algorithms allocate the task sets within the minimum number of processors. The total load, illustrated as the bottom-most graph, is the sum of the utilizations of all tasks in the system.

Partitioned Algorithms

Figures 33 and 34 illustrate the experiments performed for partitioned off-line algorithms under RM on 1000 Processors. It can be noted that for low values of α (i.e., $\alpha = 0.2$ and $\alpha = 0.5$) the best algorithms are RMGT and RMST. However their performance degrades for high values of α. The performance of RBOUND-MP improves rapidly with higher values of α. RBOUND-MP is the second worst algorithm for $\alpha = 0.2$. For $\alpha = 0.5$ RBOUND-MP is only worst than RMST and RMGT, but it is the best algorithm for values of $\alpha = 0.8$ and $\alpha = 1.0$.

RM-FFDU yields good performance and only worst than RMST (for $\alpha = 0.2$ and $\alpha = 0.5$), RMGT (for $\alpha = 0.2$, $\alpha = 0.5$ and $\alpha = 0.8$) and RBOUND-MP (for $\alpha = 0.5$, $\alpha = 0.8$ and $\alpha = 1.0$). It can be noted that its performance improves with higher values of α. Algorithm RMFF and RMBF yield between the second and fourth worst places among all algorithms. The algorithm with worst performance in all cases is RMNF.

In general, it can be noted in this experiment that for values of $\alpha = 0.2$ and $\alpha = 0.5$, algorithms RMST and RMGT yield similar performance. Also, for the same values of α similar performance is achieved by algorithms RMFF, RMBF, RBOUND-MP, RM-FFDU and FFDUF.

Figures 35 and 36 illustrate the experiments performed for partitioned on-line algorithms under RM on infinite number of processors. From theses Figures it can be noted that RMGT/M is the algorithm with best performance for values of $\alpha = 0.2$ and $\alpha = 0.5$ but it performance degrades for higher values of α. For $\alpha = 0.8$ and $\alpha = 1.0$ its performance is the second worst among all partitioned on-line algorithms. RMFF-L&L and RMBF-L&L yield the best performance among all algorithms except for low values of α (i.e., $\alpha = 0.2$ and $\alpha = 0.5$), for which algorithm RMGT/M is better. Algorithm RMNF-L&L yield the worst performance among all partitioned on-line algorithms.

Note that algorithms RMFF-L&L and RMBF-L&L yield similar performance for all values of α.

Figures 37 and 38 illustrate the experiments performed for partitioned off-line algorithms under EDF on 1000 Processors. The best performance for all values of α is yielded by EDF-BFD followed by EDF-FFD, EDF-WFD, EDF-NFD, EDF-BFI, EDF-FFI, EDF-WFI and EDF-NFI in that order. However, note that EDF-BFD, EDF-FFD, and EDF-WFD yield similar performance for all values of α, while EDF-NFD, EDF-BFI, EDF-FFI, EDF-WFI and EDF-NFI also yield similar performance.

Figures 39 and 40 illustrate the experiments performed for partitioned on-line algorithms under EDF on 1000 Processors. In this case, the best performance for all values of α is yielded by EDF-BF followed by EDF-FF, EDF-WF and EDF-NFD in that order.

Global Algorithms

Figures 41 and 42 illustrate the experiments performed for global algorithms under RM on 1000 Processors, under different values of α. From these Figures it can be noted that for all values of α the best algorithm is AdaptiveTkC. Algorithm RM yields better performance than RM-US[m/(3m-2)] except for $\alpha > 0.8$.

Figures 43 and 44 illustrate the experiments performed for global algorithms under EDF on 1000 Processors, under different values of α. From these Figures it can be noted that the algorithm with best performance is fpEDF followed by $EDF - US[m/(2m-1)]$ and EDF in that order. In general note that all algorithms yield similar performance for $\alpha = 0.2$ and $\alpha = 0.5$.

C. General Results

For the partitioned algorithms on 16 processors note that in general most of the RM algorithms tend to improve their performance with higher values of α. Quite opposite behavior is obtained by the EDF partitioned algorithms, where better performance is obtained for lower values of α.

The global algorithms under RM tend to improve their performance with higher values of α, while the global EDF algorithms tend to improve their performance with smaller values of α.

When comparing the performance of partitioned and global algorithms we observe the following:

- **RM (16 processors): Off-line** vs **On-line partitioned algorithms.** In general it can be observed that for all values of α the performance of the off-line algorithms tend to outperform the performance of the on-line algorithms.
- **RM (16 processors): Partitioned** vs **Global.** For $\alpha \leq 0.2$ the best partitioned algorithms RMST and RMGT outperform the best global algorithm AdaptiveTkC. For $\alpha = 0.5$ the best global algorithm AdaptiveTkC outperform the best partitioned algorithms RMST and RMGT. For $\alpha > 0.5$ the best global algorithm AdaptiveTkC outperform the best partitioned algorithms RBOUND-MP.
- **EDF (16 processors): Off-line** vs **On-line algorithms.** In general it can be observed that for all values of α the performance of the off-line algorithms tend to slightly outperform the performance of the on-line algorithms.
- **EDF (16 processors): Partitioned** vs **Global.** Global algorithms EDF-US[m/2m-1)] and EDF yield similar performance for $\alpha \leq 0.5$.
  For $\alpha \leq 0.2$ the best global algorithms EDF-US[m/2m-1)] and EDF yield better performance (in most of the utilizations) than the best partitioned algorithms EDF-BFD and EDF-FFD. For $\alpha > 0.2$ the best partitioned algorithms EDF-BFD and EDF-FFD yield better performance than the best global algorithm EDF-US[m/2m-1)].
- **Partitioned Scheduling (16 processors): RM** vs **EDF.** Note that in general, EDF partitioned algorithms tend to achieve better performance than the RM partitioned algorithms. The algorithms with best performance among all partitioned algorithms are EDF-BFD and EDF-FFD.
- **Global Scheduling (16 processors): RM** vs **EDF.** Note that in general, EDF algorithms outperform the RM algorithms. The algorithm with best performance among all global algorithms is EDF-US[m/2m-1)].
- **RM (1000 processors): Off-line** vs **On-line partitioned algorithms.** It can be observed that for $\alpha <= 0.2$ the performance of the best off-line algorithms RMGT and RMST is similar than the performance of the best on-line algorithm RMGT/M. For $\alpha = 0.5$ the performance of the best off-line algorithms RMGT and RMST is better than the best on-line algorithm RMGT/M.
  For $\alpha \geq 0.8$ the performance of the best of-line algorithm RBOUND-MP is better than the performance of the best on-line algorithms RMBF-L&L and RMFF-L&L.
- **RM (1000 processors): Partitioned** vs **Global.** It can be observed that the partitioned algorithms RMST and RMGT outperform all global algorithms for $\alpha \leq 0.2$. For $\alpha = 0.5$ the performance of algorithms RMST and RMGT is similar to that of the global AdaptiveTkC algorithm. For value of $\alpha > 0.5$, partitioned algorithm RBOUND-MP slightly outperforms global AdaptiveTkC.
- **EDF (1000 processors): Off-line** vs **On-line algorithms.** In general it can be observed that for all values of α the performance of the best off-line algorithm EDF-BFD tend to be similar to the performance of the best on-line algorithm EDF-BF.
- **EDF (1000 processors): Partitioned** vs **Global.** Note that for value of $\alpha = 0.2$ all algorithms yield similar

performance. For $\alpha \geq 0.5$ the partitioned algorithms EDF-BF, EDF-FF, EDF-WF, and EDF-NF outperform all global algorithms.

- **Partitioned Scheduling (1000 processors): RM** vs **EDF.** Note that all values of α, EDF partitioned algorithms tend to achieve better performance than the RM partitioned algorithms.
- **Global Scheduling (1000 processors): RM** vs **EDF.** Note that for value of $\alpha = 0.2$ the global EDF algorithms outperform the global RM algorithms. For value of $\alpha = 0.5$ all global EDF algorithms yield similar performance, which is also similar to the ADAPTIVE-TkC global RM algorithm. For value of $\alpha \geq 0.8$ the global EDF algorithms fpEDF and EDF-US[m/(2m-1)] yield similar performance than the ADAPTIVE-TkC global RM algorithm.

VII. Conclusions

In this paper we introduced the best-known real-time partitioned and global scheduling algorithms for identical multiprocessors. We introduced the schedulability conditions used for the partitioned algorithms. We introduced the best-known global algorithms and we studied the off-line and on-line schemes for all algorithms.

The main goal of this paper was to survey the best known partitioned and global multiprocessor scheduling algorithms and to compare their performance. We compared the performance of the global and the partitioned algorithms using RM and EDF scheduling policies under off-line and on-line algorithms. Also, we compared the performance of the algorithms under fixed and infinite number of processors.

Our performance evaluation study was designed with the aim to introduce the conditions under which some algorithms are better than the others, and as a future reference for the design of new algorithms.

Extensive simulation experiments were conducted to evaluate the performance and computational of the algorithms. In our simulation experiments, the schedulability tests were evaluated for different α values and number of tasks.

As a future work, we plan to test and compare the performance of the partitioned and global algorithms on non-identical multiprocessors platforms. Also, we intend to test the available non-preemptive and non-independent partitioned and global algorithms available on the real-time literature.

References

[1] B. Andersson, "Static Priority Scheduling in Multiprocessors", *PhD Thesis, Department of Comp.Eng., Chalmers University*, 2003.

[2] B. Andersson and J. Jonsson, "Fixed-Priority Preemptive Multiprocessor Scheduling: To Partition or not to Partition", *IEEE Int'l Conference on Real Time Computing Systems and Applications*, Dec. 2000.

[3] B. Andersson and J. Jonsson, "The Utilization Bounds of Partitioned and PFAIR Static-Priority Scheduling on Multiprocessors are 50 %.", *IEEE EuroMicro Conference on Real-Time Systems, July 2003.*

[4] B. Andersson, S. Baruah and J. Jonsson, "Static-priority Scheduling on Multiprocessor", *IEEE Real-Time Systems Symposium*, Dec. 2001.

[5] T. P. Baker, "Multiprocessor EDF and Deadline Monotonic Schedulability Analysis", IEEE Real-Time Systems Symposium, Dec, 2003.

[6] S. Baruah, "Robustness Results Concerning EDF Scheduling upon Uniform Multiprocessor", *Euromicro Conf. on Real-Time Systems* 2002.

[7] S. Baruah, "Scheduling Periodic Tasks on Uniform Multiprocessor", *Euromicro Conference on Real-Time Systems* June 2000.

[8] S. Baruah, "Optimal Utilization Bounds for the Fixed-Priority Scheduling of Periodic Tasks Systems on Identical Multiprocessors", *IEEE Transactions on Computers*, pp. 781-784. 2004.

[9] A. Burchard, Y. Oh, J. Liebeherr, and S. H. Son, "A Linear Time Online Task Assignment Scheme for Multiprocessor Systems", *IEEE Workshop on Real-Time Operating Systems and Software*, May 1994.

[10] A. Burchard, J. Liebeherr, Y. Oh. and S.H. Son, "New Strategies for Assigning Real-Time Tasks to Multiprocessor Systems", *IEEE Transactions on Computers*, vol. 44, number 12, Dec. 1995.

[11] E. G. Coffman, G. Galambos, S. Martello and D. Vigo, "Bin Packing Approximation Algorithms: Combinational Analysis", Kluwer Academic Publishers, Ed. D. Z. Du and P.M. Pardalos, 1998.

[12] S. Davari and S.K. Dhall, "On a Periodic Real Time Task Allocation Problem", *Annual International Conference on System Sciences*, 1986.

[13] M. L. Dertouzos, "Control Robotics: The Procedural Control of Physical Processes", *Proceedings of IFFP Congress*, 807-813, 1974.

[14] S. K. Dhall and C. L. Liu, "On a Real-Time Scheduling Problem", *Operations Research*, vol. 26, number 1, pp. 127-140, 1978.

[15] M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", W. H. Freeman, New York, 1979.

[16] S. Ghosh, D. Mosse and R. Melhem, "Fault-Tolerant Rate Monotonic Scheduling", *Journal of Real-Time Systems*, 1998.

[17] J. Goossens, S. Funk and S. Baruah, "Priority-Driven Scheduling of Periodic Task Systems on Multiprocessor", *Real Time Systems*, vol 25, 187-205, 2003.

[18] R. L. Graham, "Bounds on Multiprocessing Timing Anomalies", *SIAM Journal of Applied Mathematics*, 416-429, 1969.

[19] R. Ha and J. Liu, "Validating Timing Constraints in Multiprocessor and Distributed Real-Time Systems", *Int'l Conf. on Distributed Computing Systems*, pp. 162-171, June 21-24, 1994.

[20] M. Joseph and P. Pandya, "Finding Response Times in a Real Time System", *Computer Journal. British Computer Society*, 29(5), 1986.

[21] S. Lauzac, R. Melhem and D. Mosse, "An Improved Rate-Monotonic Admission Control and its Application", *IEEE Transactions on Computers*. vol. 52. No. 3, March 2003.

[22] S. Lauzac, R. Melhem and D. Mosse, "Comparison of Global and Partitioning Schemes for Scheduling RM Tasks on a Multiprocessor", *Euromicro Workshop on Real-Time Systems* June 1998.

[23] S. Lauzac, "Multiprocessor Scheduling of Preemptive Periodic Real-Time Tasks with Error Recovery", *PhD. Thesis. Computer Science Department, University of Pittsburgh*, 2000.

[24] J. P. Lehoczky, L. Sha and Y. Ding, "The Rate-Monotonic Scheduling Algorithm: Exact Characterization and Average Behavior", *IEEE Real-Time Systems Symposium*, pp. 166-171, 1989.

[25] J. Y.-T. Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks, *Performance Evaluation*, number 2, pp. 237-250, 1982

[26] J.W.S. Liu, "Real-Time Systems", *Prentice-Hall*, 2000.

[27] C. L. Liu and W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", *Journal of the ACM*, vol. 20, number 1, pp. 46-61, January 1973.

[28] J. M. López, J.L. Díaz, and D.F. García, "Minimum and Maximum Utilization Bounds for Multiprocessor Rate Monotonic Scheduling", *IEEE Trans. on Parallel and Distributed Systems*, 15(7), July 2004.

[29] J.M. López, M. García, J.L. Díaz and D.F. García, "Worst-Case Utilization Bound for EDF Scheduling on Real-Time Multiprocessor Systems", *Euromicro Workshop on Real-Time Systems*, pp. 25-33, 2000.

[30] J.M. López. M. García. J.L. Díaz and D.F. García, "Utilization Bounds for Multiprocessor Rate-Monotonic Scheduling", *Real Time Systems*, vol. 24, Issue 1, January 2003.

[31] J.M. López. M. García. J.L. Díaz and D.F. García, "Utilization Bounds for EDF scheduling on Real-Time Multiprocessor Systems", *Real Time Systems*, vol. 28, Issue 1, Oct. 2004.

[32] D.-I. Oh, and T.P. Baker, "Utilization Bounds for N-Processor Rate Monotonic with Static Processor Assignment", *Real Time Systems*, 15(2):183-193, 1998.

[33] Y. Oh and S. H. Son, "Fixed Priority Scheduling of Periodic Tasks on Multiprocessor Systems", *Tech. Report CS-95-16, Univ. Of Virginia. Dept. of Computer Science*, March 1995.

[34] Y. Oh and S. H. Son, "Preemptive Scheduling of Periodic Tasks on Multiprocessor: Dynamic Algorithms and Their Performance", *Tech. Report CS-93-26 Univ. Of Virginia. CS Dept.* May 1993.

[35] Y. Oh and S.H. Son, "Tight Performance Bounds of Heuristics for a Real-Time Scheduling Problem", *Tech. Report CS-93-24. Univ. Of Virginia. Dept. of Computer Science*, May 1993.

[36] L. Sha, R. Rajkumar and J. P. Lehoczky, “Priority Inheritance Protocols: An Approach to Real-Time Synchronization”, *IEEE Transactions on Computers*, 39(9):1175-1185, 1990.
[37] B. Sprunt, L. Sha and J. Lehoczky, “ Aperiodic Task Scheduling for Hard Real Time Systems”, *Journal of Real-Time Systems*, 1989.
[38] A. Srinivasan and Sanjoy Baruah, “Deadline-based Scheduling of Periodic Task Systems on Multiprocessor”, *Information Processing Letters*, 84 (2), 93-98, Nov 2002.

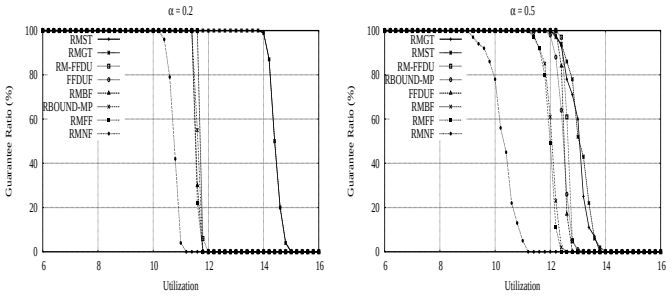

Fig. 21. Partitioned Off-line Algorithms under RM on 16 Processors

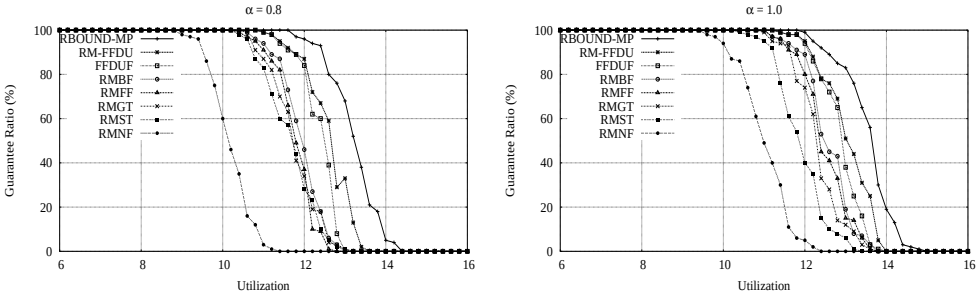

Fig. 22. Partitioned Off-line Algorithms under RM on 16 Processors

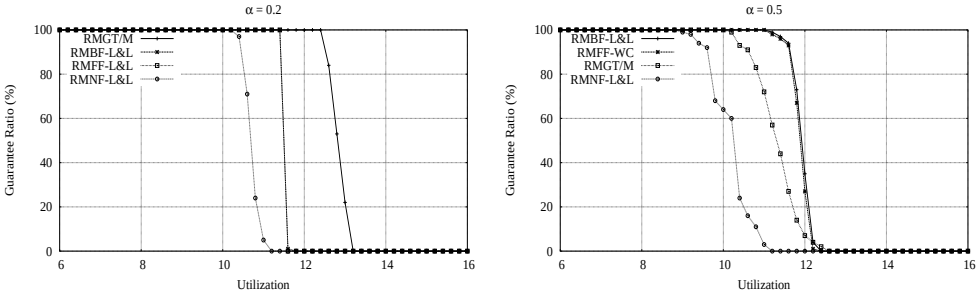

Fig. 23. Partitioned On-line Algorithms under RM on 16 Processors

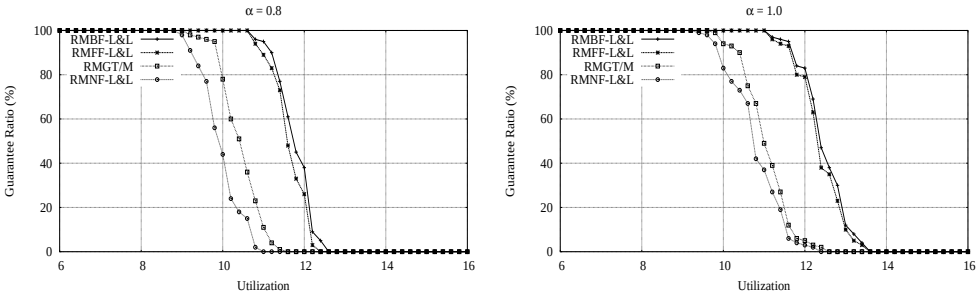

Fig. 24. Partitioned On-line Algorithms under RM on 16 Processors

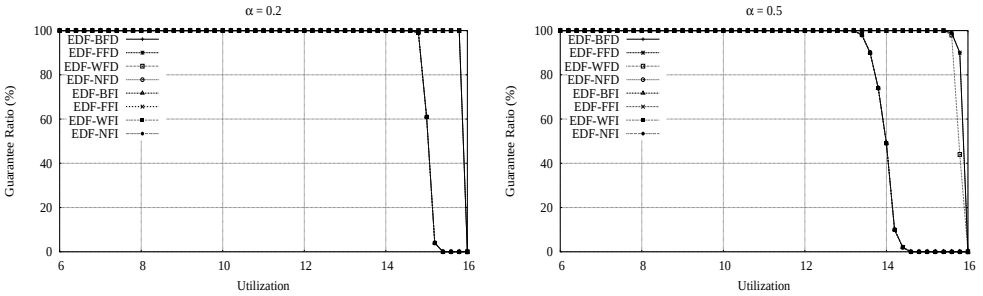

Fig. 25. Partitioned Off-line Algorithms under EDF on 16 Processors

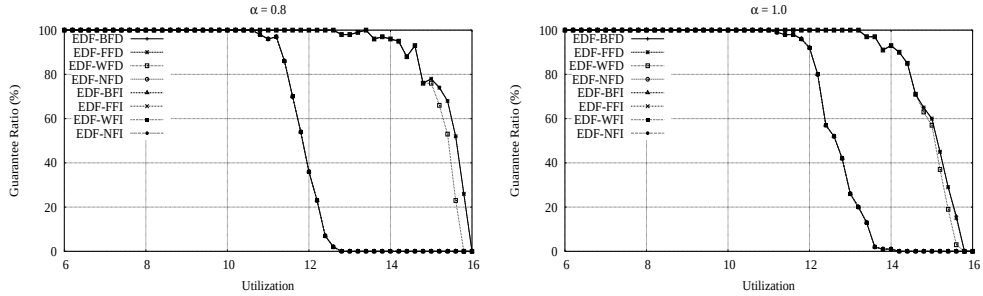

Fig. 26. Partitioned Off-line Algorithms under EDF on 16 Processors

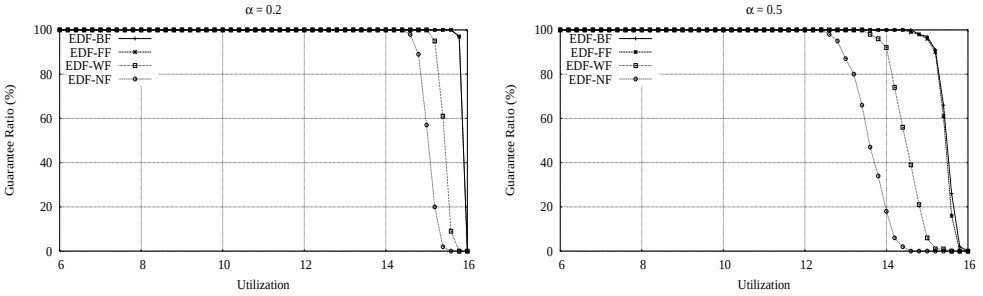

Fig. 27. Partitioned On-line Algorithms under EDF on 16 Processors

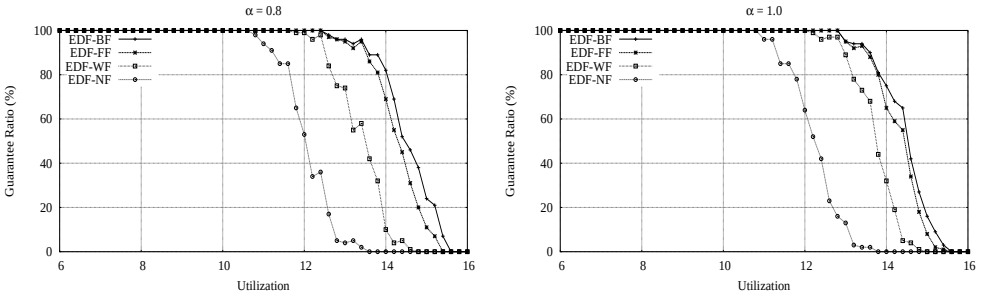

Fig. 28. Partitioned On-line Algorithms under EDF on 16 Processors

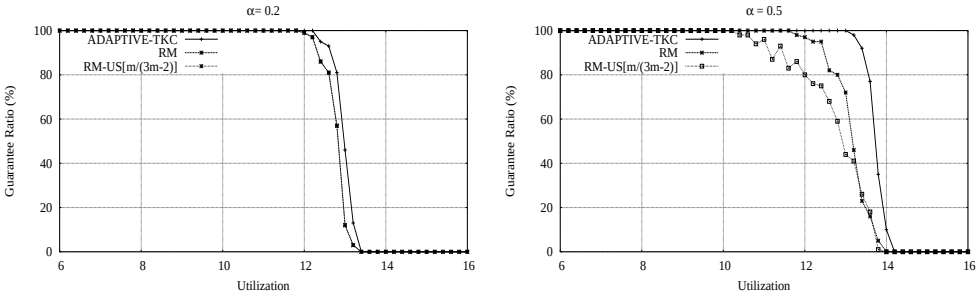

Fig. 29. Global Algorithms under RM on 16 Processors

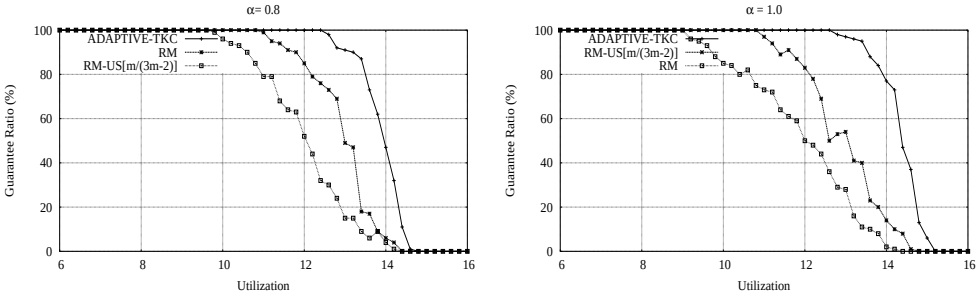

Fig. 30. Global Algorithms under RM on 16 Processors

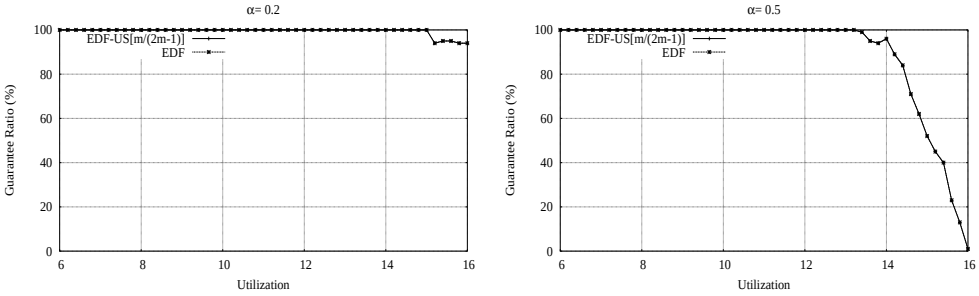

Fig. 31. Global Algorithms under EDF on 16 Processors

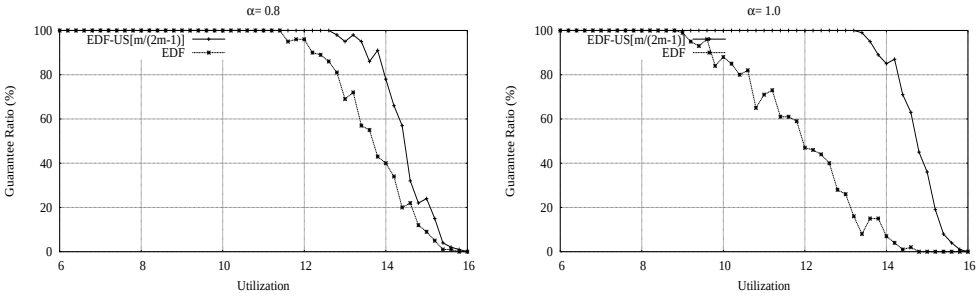

Fig. 32. Global Algorithms under EDF on 16 Processors

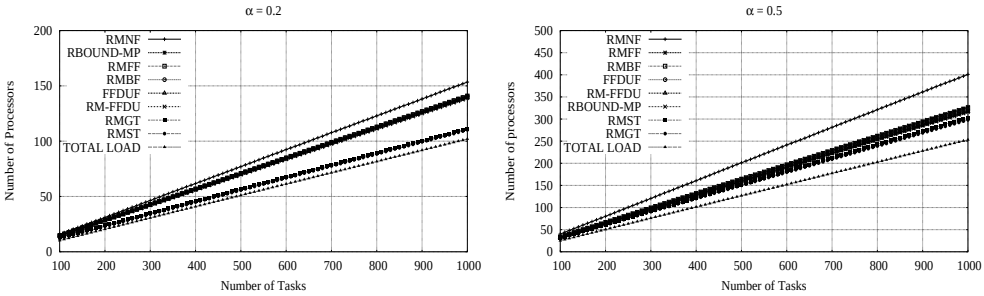

Fig. 33. Partitioned Off-line Algorithms under RM on 1000 Processors

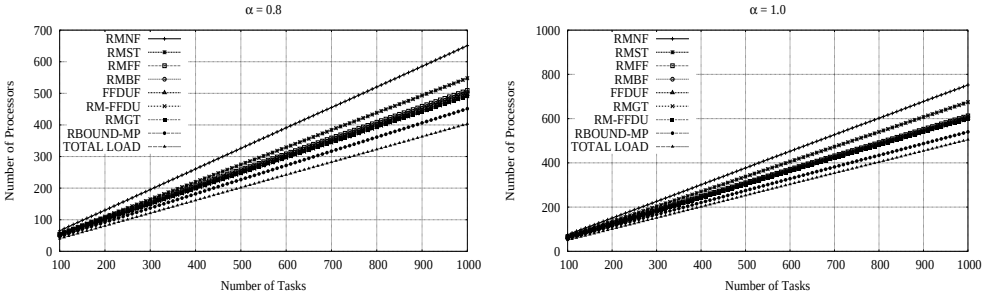

Fig. 34. Partitioned Off-line Algorithms under RM on 1000 Processors

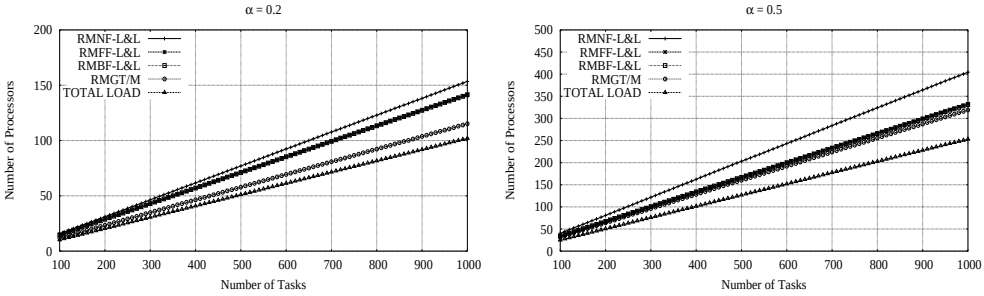

Fig. 35. Partitioned On-line Algorithms under RM on 1000 Processors

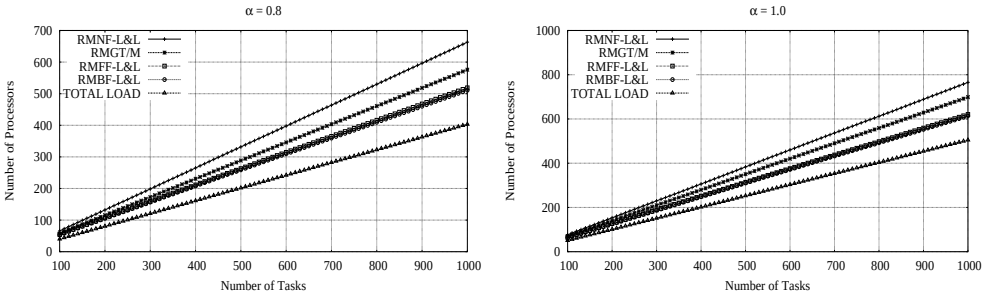

Fig. 36. Partitioned On-line Algorithms under RM on 1000 Processors

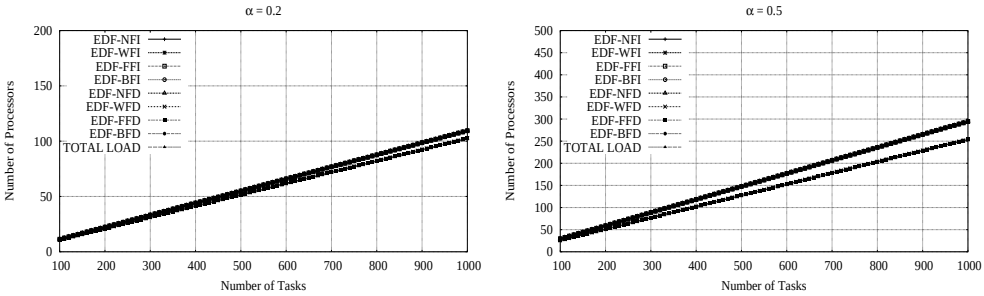

Fig. 37. Partitioned Off-line Algorithms under EDF on 1000 Processors

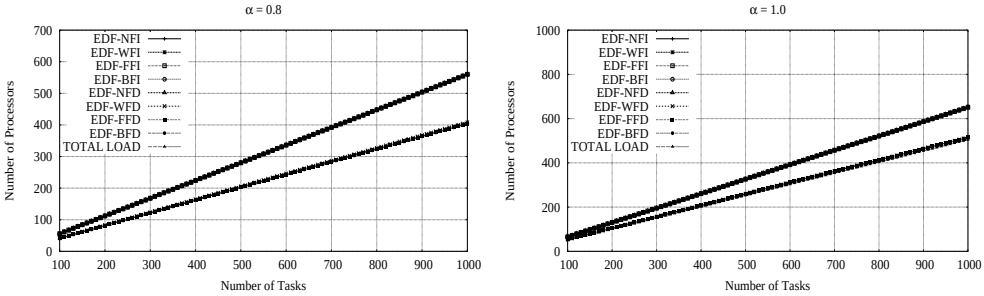

Fig. 38. Partitioned Off-line Algorithms under EDF on 1000 Processors

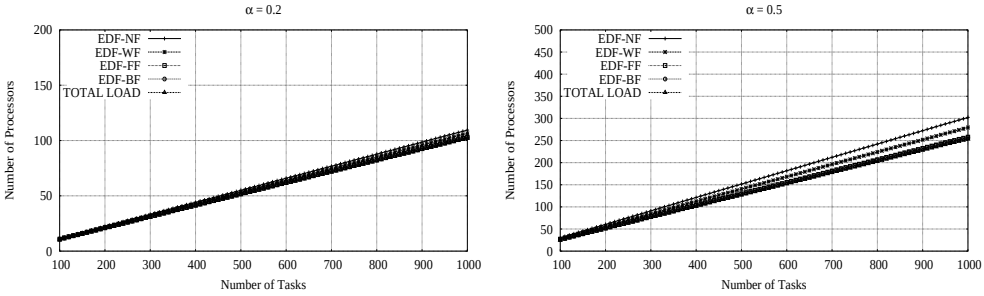

Fig. 39. Partitioned On-line Algorithms under EDF on 1000 Processors

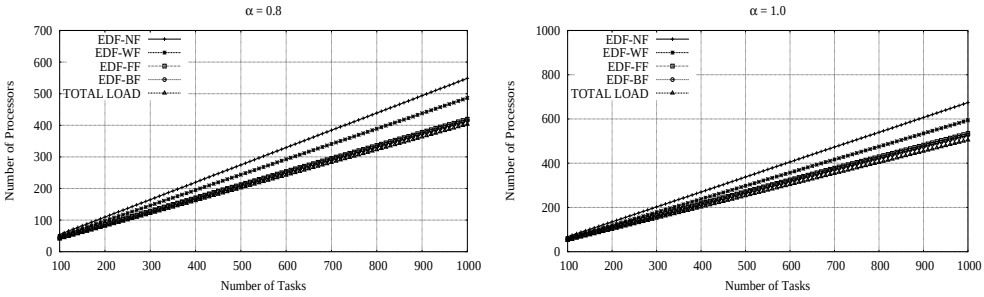

Fig. 40. Partitioned On-line Algorithms under EDF on 1000 Processors

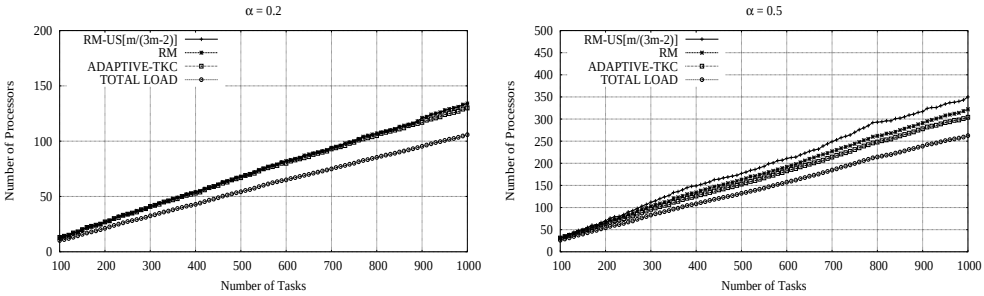

Fig. 41. Global Algorithms under RM on 1000 Processors

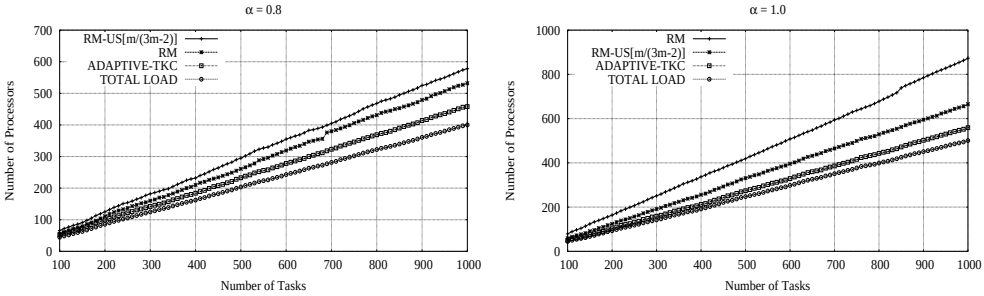

Fig. 42. Global Algorithms under RM on 1000 Processors

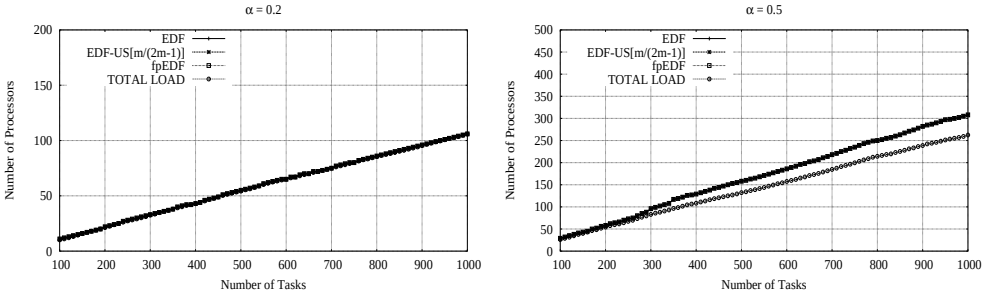

Fig. 43. Global Algorithms under EDF on 1000 Processors

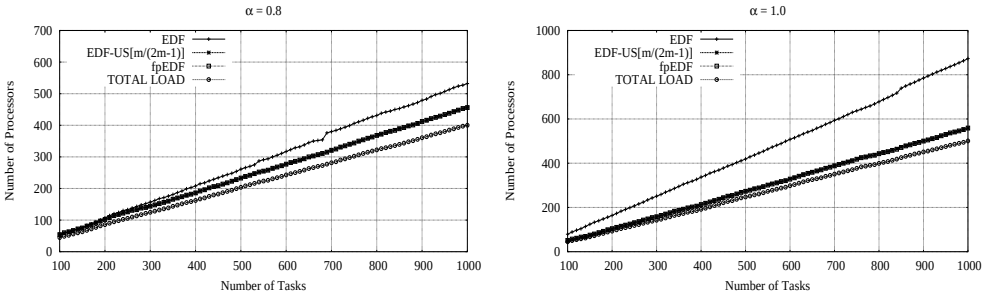

Fig. 44. Global Algorithms under EDF on 1000 Processors