Optimized Deadline Assignment for Tasks and Messages in Distributed Real-Time Systems

Juan M. Rivas, J. Javier Gutiérrez, J. Carlos Palencia, and Michael González Harbour

Computers and Real-Time Group, Universidad de Cantabria, 39005-Santander, SPAIN
{rivasjm, gutierjj, palencij, mgh}@unican.es

Abstract[1]

The assignment of scheduling parameters under the Earliest Deadline First (EDF) scheduling policy is trivial in single processor systems because deadlines are used directly. However, it is still difficult to find a feasible deadline assignment for EDF distributed systems when the utilization levels of the CPUs and communication networks are pushed near to their limits. Most distributed applications specify end-to-end deadlines for each transaction and there are no individual deadlines assigned to their tasks or messages. This paper presents a new heuristic algorithm, called HOSDA (Heuristic Optimized Scheduling Deadline Assignment), for optimizing the assignment of deadlines to tasks and messages in distributed hard real-time systems. The algorithm is based on HOPA (Heuristic Optimized Priority Assignment), a previous method for the assignment of priorities in fixed priority distributed systems. The results of the proposed algorithm are compared with two other algorithms that exist for solving the same problem, and show that a utilization increase of up to 18% is possible. The paper also proposes a new schedulability analysis technique for EDF distributed systems with local scheduling deadlines.

1. Introduction

Distributed real-time systems are increasingly important in today's embedded systems, since low-cost networking facilities allow the interconnection of multiple devices and their controllers into a single large system. The system's software is composed of concurrent tasks that are often statically allocated to processing nodes where each task may exchange messages with other tasks in the same processing node or in a different one.

Although fixed priority scheduling is the most popular on-line scheduling policy, usage of the Earliest deadline first (EDF) policy is starting to get more attention in industrial environments, given its benefits in terms of increased resource usage. Analysis techniques are available to determine whether a given system, either single processor or distributed, will meet all of its timing requirements. EDF is now available in real-time languages like Ada 2005 [21] or RTSJ [23]. It is available in real-time operating systems such as SHaRK [18] or ERIKA [4], and has been implemented at the application level in OSEK/VDX embedded operating systems [2]. There are also real-time networks using EDF for scheduling messages, for instance in general purpose networks [3], or in the CAN Bus [14]. Given its maturity, it is expected that the number of industrial applications using EDF will increase in the near future.

One of the problems for scheduling distributed real-time systems is finding an assignment of scheduling parameters that leads to a feasible scheduling. This problem is fully solved for single processor systems. However, most timing requirements for distributed transactions are in the form of end-to-end deadlines, and finding scheduling parameters for the tasks and messages that constitute the transaction has no known optimum solution other than an intractable brute-force mechanism.

Different heuristics that can provide acceptable solutions to the assignment of scheduling parameters in a reasonable time have been studied for fixed priorities: simulated annealing or genetic algorithms can be used, as general-purpose optimization techniques; in [6] an algorithm called HOPA (Heuristic Optimized Priority Assignment), based on iteratively applying response time analysis (RTA), is shown to usually find better solutions than simulated annealing, in less time. There are also some algorithms for EDF that distribute end-to-end deadlines into individual deadlines for tasks and messages. The technique presented in [10] tries to reduce the number of missed deadlines in soft real-time systems. In [8][9] the assignment of deadlines is solved together with the allocation of tasks to processors; only a subset of the tasks are constrained by predetermined assignments to specific processors while the others can be allocated by the algorithm in order to make the system schedulable with the deadlines

1. This work has been funded in part by the Spanish Ministry of Science and Technology under grant number TIN2008-06766-C03-03 (RT-MODEL), and by the IST Programme of the European Commission under project FP6/2005/IST/5-034026 (FRESCOR). This work reflects only the author's views; the EU is not liable for any use that may be made of the information contained herein.

previously assigned. Other works, like the one proposed in [7], deal with strategies to analyze the schedulability of the distributed system based on pipelines, which is a restrictive model.

In [11], Liu proposes four basic strategies for assigning deadlines in EDF distributed systems. However these are not iterative algorithms capable of improving on the solution found. In this paper we explore a new heuristic algorithm that tries to reuse some of the ideas of HOPA [6]. We call the new algorithm HOSDA (Heuristic Optimized Scheduling Deadline Assignment), and its goal is to find a feasible assignment of deadlines in a real-time distributed system by distributing end-to-end deadlines into intermediate deadlines, and by iterating over response time analysis to improve on the solution found.

In distributed EDF scheduling it is possible to find two kinds of schedulers: *global-deadline* schedulers have their deadlines referenced to the arrival of the event that releases the transaction, possibly in a different resource; *local-deadline* schedulers have deadlines referenced to the release time of each task in its own processing resource. Global-deadline schedulers require clock synchronization among all the processing resources involved, and the precision of the deadlines depends on the precision of the clock synchronization mechanism. While some systems do have precise clock synchronization this is not always the case. Local-deadline schedulers just use the local clock of each processor and are more general and easier to implement. The techniques proposed in [8][9][11] for assigning deadlines are based on local-deadline schedulers, and [10] used global deadlines.

It is interesting to notice that current response-time analysis techniques for EDF [20][13] are developed for global-deadline schedulers. Since local-deadline schedulers are more useful to general distributed platforms, in this paper we show how to adapt RTA techniques to support local-deadline schedulers. These RTA techniques are the basis for the HOSDA deadline distribution algorithm that we propose in this paper. In addition, by using the new analysis we are able to compare the results with those obtained by applying the techniques in [11]. As we will see, HOSDA outperforms the basic algorithms and is able to increase the processor utilization by up to 18% in some of the tested examples.

The paper is organized as follows. In Section 2 we provide a quick review of the model that we use for the distributed system, and we briefly describe the state of the art for the analysis techniques. Section 3 describes the proposed analysis algorithm for EDF with local scheduling deadlines. In Section 4 we show the details of the heuristic algorithm for optimizing the assignment of deadlines in distributed real-time systems. Section 5 evaluates the results of our algorithm by comparing them with those obtained by current deadline distribution strategies, showing that in most cases our algorithm finds better solutions in a reasonable time. Finally, in Section 6 we give our conclusions.

2. System Model and Current Analysis Techniques

For our system model we assume a real-time distributed system with multiple processing resources (CPUs) and one or more communication networks. We will use EDF scheduling in both the processors and the networks. Since the analysis of message traffic on the networks can be carried out using the same techniques that are used in the CPUs, in our distributed system we treat messages and communication resources exactly as if they were tasks in processing resources, except for a small amount of non preemption that must be taken into account because messages can only be preempted at packet boundaries. Consequently, for simplifying the analysis we assume a model equivalent to the distributed system with only processing nodes and tasks, and we model the non-preemptive effects as blocking times. The set of available processors is $\{PR_1, PR_2, ...\}$.

In our hard real-time system model we assume that all event sequences that arrive to the system are known in advance, and that the worst-case rates of those events with hard real-time requirements are also known. We also assume that tasks are statically assigned to processors (similarly messages to communication networks). In many hard real- time systems tasks are tied to specific processors because of the presence of special hardware devices that are needed by the tasks.

We will consider a task model with periodic distributed *transactions*. Each transaction Γ_i is released by a periodic sequence of external events with period T_i, and contains a set of m_i *tasks*, each one statically assigned to a specific processor, PR_k. Each periodic release of a transaction causes the execution of one instance of that transaction called a *transaction job*. Each task is released when the previous task in its transaction finishes its execution. Each release of a task causes the execution of one instance of that task, that we will call a *task job* or simply a *job*.

Although we assume a periodic task model, the techniques presented in this paper can be applied to sporadic transactions (i.e., aperiodic transactions with a specific minimum interarrival time that can be used as the period for the purpose of worst-case analysis).

Figure 1 shows an example of one of these transactions, with just three tasks, each executing in a different resource (two CPUs and one network in this case). The external event that releases the transaction is represented by a horizontal arrow labeled e_i, and has a period of T_i. The other

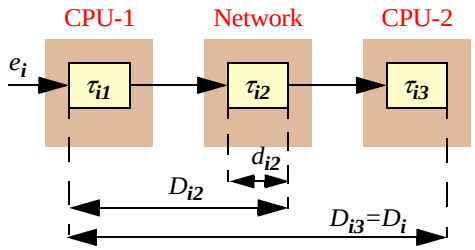

Figure 1. System model

horizontal arrows represent the internal events by which each task releases the following task in its transaction. These internal events represent a kind of precedence constraints because a task job cannot execute before it is released.

Each task is identified with two subscripts: the first one identifies the transaction to which it belongs, and the second one the position that the task occupies within the tasks of its transaction, when they are ordered by precedence. In this way, τ_{ij} will be the j-th task of transaction Γ_i, with a worst-case execution time of C_{ij}.

The timing requirements that we consider are end-to-end deadlines that start at the transaction job's period, and must be met by the final task in the transaction. We will call D_i the relative end-to-end deadline of Γ_i. We allow deadlines to be larger than the periods, and thus at each time there may be several jobs of the same transaction pending. Each task also has an associated local deadline, d_{ij}, which is relative to the release time of the task job, and a global deadline, D_{ij}, which is relative to the start of the transaction job's period. The global deadline of the last task in the transaction coincides with the end-to-end deadline, $D_{im_i} = D_i$.

For each task τ_{ij} job we define its response time as the difference between its completion time and the instant at which the period of the transaction job started, t_n. The worst-case response time will be called R_{ij}.

We allow the external event that triggers a transaction to have a maximum release jitter J_{i1} in relation to the theoretical start of each period. This implies that the release time of the first task of the n-th transaction job is in $[t_n, t_n+J_{i1}]$. Despite this jitter, global deadlines and response times always refer to the theoretical start of their respective job's period (t_n), not to the actual release of the transaction.

In addition to jitter originated by the timing of the external event, tasks after the first are also affected by the variability of the response times of the preceding tasks in the transaction. This implies that every task may have release jitter, which we will call J_{ij}. If we conservatively assume that best case execution times are zero, the best case response time of any task is zero and therefore the release time of the n-th job of task τ_{ij}, $j \neq 1$, is in $[t_n, t_n+J_{ij}]$ with J_{ij} being equal to the worst-case response time of the previous task, $R_{i,j-1}$. We assume that J_{ij} may be larger than the period of its transaction, T_i.

We will assume that if tasks synchronize for using shared resources in a mutually exclusive way they will be using a hard real-time synchronization protocol such as Baker's protocol [1]. Under this assumption, the effects of other tasks on a task under analysis τ_{ij}, as well as the effects of any small non-preemptive section (such as the non-preemptible network packets), are bounded by an amount called the blocking term, B_{ij}.

In summary, the notation used for the parameters of each transaction Γ_i is the following:

- T_i Period ($T_i > 0$)
- D_i End-to-end deadline ($D_i > 0$)

And for each task, τ_{ij},

- C_{ij} Worst-case execution time ($C_{ij} > 0$)
- d_{ij} Relative local deadline ($d_{ij} > 0$)
- D_{ij} Relative global deadline ($D_{ij} > 0$)
- R_{ij} Worst-case response time ($R_{ij} > 0$)
- J_{ij} Maximum release jitter ($J_{ij} \geq 0$)
- B_{ij} Blocking time ($B_{ij} \geq 0$)

This model described here is well-suited to represent a large number of real-time architectures that are found in practice. For example, if the system is using a client-server approach, each portion of a task that invokes a service from a remote server may be decomposed into the following activities: the activity before invoking the service, the message sent to the server, the server's activity, the reply message, and the activity after invoking the server.

There are different analysis techniques that can be used to analyze a system model like the one proposed here. Spuri presents the so-called holistic response-time analysis [20], an extension of the analysis for fixed priority systems by Tindell and Clark [22], that is based on the assumption that all tasks of a transaction are independent of the others, except that the variability of the response time of each task can cause release jitter for the next task in the transaction. This analysis is pessimistic because it can create initial conditions for the analysis that may not occur in practice. Palencia et al provide an improvement of this algorithm [13] that better exploits the interdependencies among tasks of the same transaction by considering offsets for the release of each task. This offset-based analysis is still pessimistic, but offers better results than the holistic analysis at the expense of more complexity. Pellizzoni and Lipari further improve schedulability by using offsets to eliminate the negative effects of jitter [15][16]. However, all these techniques rely on global-deadline schedulers, and for that

reason we develop a new analysis here to address local-deadline schedulers.

3. EDF Response-Time analysis for Local Deadlines

In this section we will extend the holistic analysis [20][22] for EDF distributed systems to support local deadline schedulers. Even though the new analysis is based on similar principles, the underlying model and the resulting equations are different.

In the holistic analysis we assume for each analysis step that every task is independent of the other tasks, even from those belonging to the same transaction. After each analysis step, the dependencies are captured into the jitter terms of each task.

Response time analysis is based on the creation of the longest busy period, found from a critical instant. The following theorem helps us to find the critical instant for a task in the context of the independent tasks assumption. It was proven by Spuri [19] for global deadlines but we adapt it to local deadlines.

Theorem 1. The worst-case response time of a task τ_{ab} is found in a busy period in which each task τ_{ij} in the same processor (different from τ_{ab}) is scheduled such that its first job that is inside the busy period is released at the beginning of that busy period, after having experienced its maximum jitter (i.e., the start of the corresponding task job's period was J_{ij} time units before the start of the busy period), and the remainder of the jobs are released with the minimum jitter that makes the job start inside the busy period.

Proof: The proof can be found in [17] and is omitted here because it is similar to the one provided by Spuri in [19]. ❐

Note that, contrary to the other tasks, releasing the analyzed task at the start of the busy period may not lead to its worst-case response time. So, the critical instant for a task is found in a busy period that is started by the simultaneous release of all tasks except perhaps the one under analysis.

Under the conditions of theorem 1, the worst-case contribution of a task τ_{ij} to a busy period of length l when the deadline of τ_{ab} occurs at instant D is [17]:

$$W_{ij}(l, D) = min\left(p_l, p_D\right) \cdot C_{ij} \quad (1)$$

where p_l is the number of releases of τ_{ij} in the busy period:

$$p_l = \left\lceil \frac{l + J_{ij}}{T_i} \right\rceil \quad (2)$$

and p_D the number of releases with deadline before or at D:

$$p_D = \begin{cases} 0 & if\ D < d_{ij} \\ \left\lfloor \frac{J_{ij} + D - d_{ij}}{T_i} \right\rfloor + 1 & otherwise \end{cases} \quad (3)$$

Using this expression we can calculate the worst-case response time of task τ_{ab}. Unfortunately, we don't know how to phase the release time of τ_{ab} in relation to the busy period, but it is easy to see that the worst case situation must be found when the release time is placed at the beginning of the busy period, or at an instant such that the deadline of the analyzed job of τ_{ab} coincides with the deadline of a task τ_{ij}'s job. The set of instants, Ψ_{ij}, at which the deadline of τ_{ab}'s job coincides with the deadlines of one of the task jobs in the busy period is:

$$\Psi_{ij} = \bigcup \{(p-1)T_i - J_{ij} + d_{ij}\} \quad \forall p = 1 \ldots \left\lceil \frac{L + J_{ij}}{T_i} \right\rceil \quad (4)$$

where L corresponds to longest busy period, calculated as:

$$L = \sum_{\forall \tau_{ij}} \left\lceil \frac{L + J_{ij}}{T_i} \right\rceil \cdot C_{ij} \quad (5)$$

This set Ψ_{ij} must be augmented with the deadlines corresponding to task τ_{ab}

$$\Psi_{ab} = \bigcup \{(p-1)T_a + d_{ab}\} \quad \forall p = 1 \ldots \left\lceil \frac{L}{T_a} \right\rceil \quad (6)$$

And so the full set of situations for which τ_{ab} has to be analyzed corresponds to those releases whose deadline is in the set

$$\Psi = \Psi_{ij} \cup \Psi_{ab} \quad (7)$$

Each potential release time for τ_{ab} is obtained by subtracting d_{ab} from each value in Ψ. Checking the response times under all these release times we can find the one that causes the worst-case response time of the task. Given that there may be several releases of τ_{ab} in the busy period, we must analyze them all. For each value $\psi \in \Psi$, the completion time of release p of τ_a, $w_{ab}^{\psi}(p)$, can be calculated by adding the worst-case contribution of all tasks in the same processor, and the blocking term:

$$w_{ab}^{\psi}(p) = B_{ab} + pC_{ab} + \sum_{\forall ij \neq ab} W_{ij}(w_{ab}^{\psi}(p), \psi) \quad (8)$$

The worst-case response time is calculated by subtracting the start of the job's period from the resulting completion time:

$$R_{ab}^{\psi}(p) = w_{ab}^{\psi}(p) - (\psi - d_{ab} - J_{ab}) \qquad (9)$$

For each value of p, we only need to check the values of ψ in one period, because if the release time corresponding to ψ was greater than the corresponding period, then we would be analyzing another job with a different value of p. This allows us to restrict the set of values to be checked:

$$\Psi^* = \{\psi_x \in \Psi \,|\, (p-1)T_a + d_{ab} \le \psi_x < pT_a + d_{ab}\} \qquad (10)$$

Finally, to calculate the worst-case response time of task τ_{ab} we must determine the maximum response times within all the potential release times examined:

$$R_{ab} = max(R_{ab}^{\psi}(p)) \quad \forall p = 1 \ldots \left\lceil \frac{L}{T_a} \right\rceil, \forall \psi \in \Psi^* \qquad (11)$$

We can now feed these response times into the holistic analysis loop like in [20], obtaining new jitter values from the response times and repeating the analysis until a stable solution is obtained. Since the dependencies of response times on jitters are monotonically increasing, the algorithm is known to converge to the final solution, except when the utilization is close to 100% and in special cases that experience shows that are uncommon.

4. Heuristic Algorithm for Optimized Deadline Assignment

Paper [6] proposes the HOPA algorithm that, unlike general-purpose optimization algorithms such as simulated annealing, uses knowledge of the factors that influence the timing behavior to find an optimized solution to the priority assignment problem in fixed-priority distributed systems.

HOPA is based on the distribution of the global deadlines of each transaction among the different tasks that compose it. Once each task is assigned an artificial local deadline, deadline monotonic priorities are assigned in each processing resource and an analysis of the whole system is carried out. As a result of the analysis, new local deadlines are calculated. The iteration proceeds until a schedulable solution is found or some stopping condition is reached.

In this section we propose a new heuristic algorithm called HOSDA which is the adaptation of HOPA to systems scheduled by EDF. The objective is to check if the basic method to calculate the artificial deadlines that lead to schedulable solutions for fixed priorities can also be used to assign scheduling deadlines for EDF distributed systems. From a high-level point of view, the algorithm is:

```
algorithm HOSDA is
begin
   assign initial scheduling deadlines;
   loop
      calculate worst-case response times;
      exit when some stopping criterion;
      calculate new scheduling deadlines;
   end loop;
end HOSDA;
```

The scheduling deadlines should be assigned in some way that preserves the global deadlines:

$$D_{ij} = \sum_{k \in pr_i(j)} d_{ik} \qquad (12)$$

where:

D_{ij} is the global deadline (intermediate or end-to-end) of task τ_{ij}.

d_{ij} is the local scheduling deadline of task τ_{ij}.

$pr_i(j)$ is the set of tasks preceding task τ_{ij} in the transaction i to which it belongs, including itself.

The proportional deadline assignment algorithm proposed in [11] can be used as the initial deadline assignment.

After all the scheduling deadlines have been assigned, the system is analyzed using the technique described in Section 3 which is adapted to systems with local deadlines. If the solution is not schedulable, new scheduling deadlines are calculated by redistributing the global deadlines among the tasks of each transaction.

The redistribution of local deadlines uses the concept of "*excess*" of each task which, intuitively and in the same way than in HOPA, measures the distance that separates each task from schedulability. To avoid convergence problems or very long calculations that the holistic analysis may have when utilizations are very high, we have added a termination condition to the holistic analysis that makes it stop when the response time of a task exceeds the imposed deadline by a configurable factor. In this way we bound and shorten the analysis time and we assure that the assignment algorithm can continue working.

The original algorithm for fixed priorities had two different definitions for the excess of a particular task τ_{ij} that made it behave differently. The first definition, "excess of response time", was based on the difference between the local response time and the local deadline, and led to faster solutions; the second definition, "excess of compute time", was based on the calculation of the slack time [6] and led to more schedulable solutions.

For the HOSDA algorithm we have made extensive experiments trying to find a suitable definition for the excess that would give reasonable results in most cases,

and as a result we have redefined the "excess of response time" in two different ways:

$$exc(\tau_{ij}) = \begin{cases} (R_{ij} - J_{ij} - d_{ij})\dfrac{\Delta R_{ij}}{\Delta D_{ij}} & (1) \\ or & \\ (R_{ij} - d_{ij})\dfrac{\Delta R_{ij}}{\Delta D_{ij}} & (2) \end{cases} \quad (13)$$

where:

ΔR_{ij} is the difference between the worst case global response times of the task before τ_{ij} that has a global deadline and the task after τ_{ij} that also has a global deadline.

ΔD_{ij} is the difference between the global deadlines of the tasks before and after τ_{ij} that have a global deadline.

If there is no previous task with a global deadline, then 0 is used to find the difference in ΔR_{ij} and ΔD_{ij}. In the common case when there is only a single end-to-end deadline for the transaction all the ΔR_{ij} terms of the same transaction have the same value related to the response time of the last task in the transaction, $\Delta R_{ij} = R_{im_i}$, and the same applies to $\Delta D_{ij} = D_i$.

Both definitions of excess can lead separately to feasible deadline assignments. As in our tests the majority of the cases are solved with definition (2) this is the recommended option to try first.

Currently we do not use the excess of computation time as it is defined in [6] because the calculation of slack time requires a repetitive utilization of the analysis algorithm, and given the much longer computation times of the analysis for EDF scheduling there is a huge cost in time of analysis except for very small examples.

We use the same definitions as in [6] for the excess of each processing resource, $exc(PR_k)$, the maximum excess of all the processing resources, $Mex(PR)$, and the maximum of the excesses of all the tasks belonging to a particular transaction responding to external event e_i, which we call $Mex(e_i)$.

Given these definitions of excess times, we calculate the new scheduling deadline for each task $d_{ij}(new)$ as a function of the old scheduling deadline for that task, the excess for that task, the excess for the resource to which that task belongs, and the values of two empirical constants k_R and k_a.

$$d_{ij}(new) = d_{ij}(old)\left(1 + \frac{exc(PR_k)}{k_R \cdot Mex(PR)}\right)\left(1 + \frac{exc(\tau_{ij})}{k_a \cdot Mex(e_i)}\right) \quad (14)$$

The k_R and k_a constants, like in HOPA, control the relative influence of the processing resource and task components, respectively, in the calculation of the new deadline. HOSDA uses an array of pairs of constants allowing the algorithm to evolve with the current pair for a bounded number of iterations; then the pair is changed to the next one. In order to empirically determine which values give the best results, extensive experiments were made. These experiments consisted of applying the HOSDA algorithm over a wide set of examples. This examples are comprised of systems with different number of processors (2-15), transaction lengths (number of tasks between 1 and the number of processors) and periods (differences of up to 3 orders of magnitude), and end-to-end deadlines (between the period, *T*, and 2**T**number of processors).

For each example generated, different sets of values for k_R and k_a were applied. We found out that higher utilizations could be reached if the values for this parameters were between 1.0 and 3.0. Normally, in the HOSDA algorithm we start with values of k_R=k_a=1.5, and we then change both values to 2.0, 2.5, 3.0, etc., until one stopping condition is reached.

Like in HOPA, once the new local deadlines d_{ij} have been obtained for all the tasks in the transaction, we adjust them proportionally to make them fit into the global deadlines.

The HOSDA algorithm stops whenever one of the following stopping conditions is met:

- A schedulable solution has been found.
- Two consecutive deadline assignments are identical (in which case the algorithm would continue providing the same solution).
- A maximum number of iterations has been reached. This number is configurable and, for a given size of the system, it sets a limit to the time that the user is willing to wait for obtaining a solution.

After the algorithm finds a deadline assignment that makes the system schedulable, it is capable of finding a better optimized solution by executing more iterations, up to an adjustable limit. The best way to compare the schedulability of a system is by making a sensitivity analysis using the slack of the system as it is defined in the MAST tools [12]. However, sensitivity analysis is slow, and in order to faster determine how good a solution is, compared to others, we define a normalized schedulability index, the transaction index, which is a function of the worst distance

between the worst-case response times and the global deadlines for each transaction:

$$TransactionIndex = \frac{(D_i - R_i)}{D_i} \qquad (15)$$

Finally, the normalized schedulability index is the average of the transaction indexes for all the transactions of the system. The larger the index becomes, the better the schedulability of the system is in terms of distance between the response times and the deadlines.

5. Evaluation of the Heuristic Algorithm

In this section we compare the performance of the heuristic algorithm proposed with existing techniques for deadline distribution described by J. Liu in [11]. We choose the best two of them: Proportional Deadline (PD) and Normalized Proportional Deadline (NPD).

In order to carry out the comparison between the HOSDA algorithm and the two selected techniques (PD, NPD), we have implemented and integrated the deadline assignment algorithms and the analysis technique into the MAST suite of tools [12]. This allows us to use both a precise model [5] to describe the systems under test and a uniform way to check the results. Since MAST is free software this also makes the implementation of the techniques available to the community.

A generator of examples has also been developed to test the execution of the algoritms over a wide spectrum of cases. This generator starts with a base system, characterized by the number of processors and transactions, and for each generated example, it assigns a random number of tasks and periods to each transaction. The WCET of each task is sequentially increased from a very low utilization until the utilization of the system reaches 100%. Systems generated by this tool are stored to be processed later.

In order to illustrate the performance of the HOSDA algorithm, we have chosen three base systems with different levels of complexity: a Small, Intermediate, and Big Size Example (SSE, ISE, and BSE respectively). Table 1 shows the number of processors and the number of transactions for each base system. The number of tasks in each transaction varies randomly from 1 to the number of processors, and can be different in each example. We have generated 100 examples for each base system, and we apply five types of end-to-end deadlines to the transactions: deadline equal to period (D=T); deadline proportional to the period and the number of processors (N) traversed by the transaction multiplied by 0.5 ($D = 0.5 \cdot N \cdot T$), one ($D = N \cdot T$), or two ($D = 2 \cdot N \cdot T$); and finally a random deadline between T and $2 \cdot N \cdot T$.

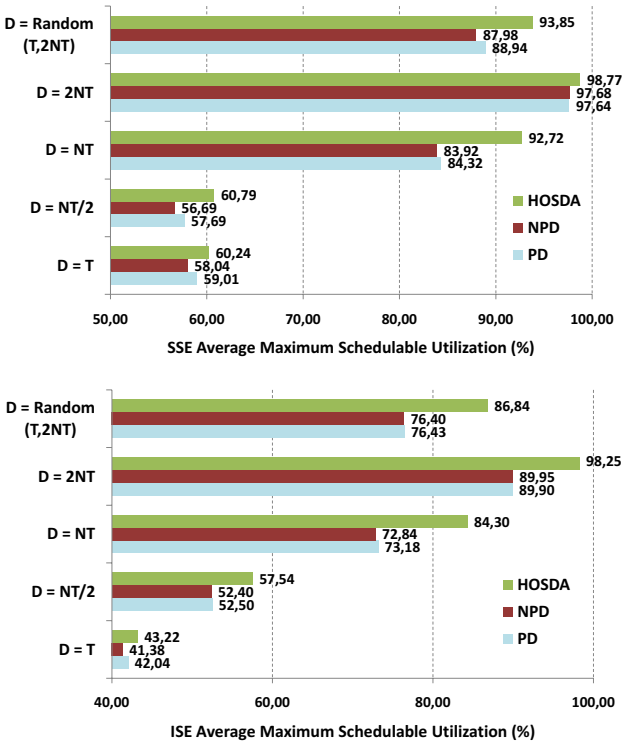

Table 1. Base systems for the examples generator

	SSE	ISE	BSE
Number of Processors	3	5	8
Number of Transactions	6	8	12

The deadline assignment algorithms PD, NPD and HOSDA have been applied to the resulting set of 1500 examples. Starting from the very low utilization assigned to each example, our purpose is to measure which is the highest utilization reached by each technique while keeping the system schedulable, and the CPU time spent in finding every solution. We have run the test with three fixed values of k_R=k_a=[1.5,2.0,3.0].

Figure 2 shows the average maximum schedulable utilization reached by the algoritms for each case-study. In this figure we can see that, on average, HOSDA can schedule systems with higher utilizations than PD and NPD. However, there is a small amount of examples in which HOSDA, at its first attempt with the constants selected, has not been better than NPD. Of course, HOSDA is always better or equal than PD, as this is the starting assignment. For each base system, Table 2 shows the percentage of examples in which HOSDA has obtained higher, equal, or lower utilizations than NPD. Despite of this small set of

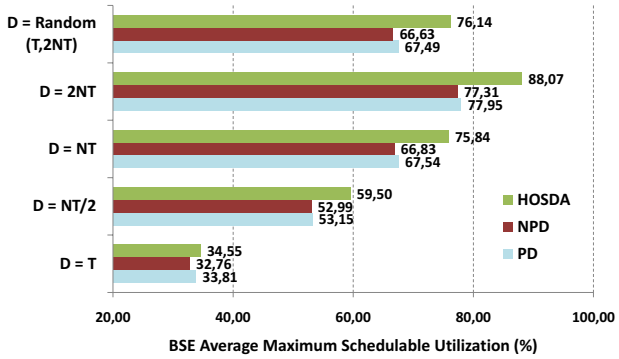

Figure 2. Schedulability results of the three algorithms

examples in favor of NPD, we can also appreciate that on average NPD found worst solutions than PD.

Table 2. HOSDA/NPD (% of examples)

	Higher	Equal	Lower
SSE	59,4	35	5,6
ISE	79,4	15,2	5,4
BSE	87,0	7,6	5,4

In Figure 3 we can see the average CPU time spent by each algorithm to find a feasible solution for the BSE base system. For the PD and NPD algorithms this CPU time is composed of the time needed to make the assignment of the deadlines plus the time to run the schedulability analysis over the solution proposed. The time spent by HOSDA depends on the number of iterations needed to find the solution. Of course HOSDA has higher execution times as it repeats the analysis potentially many times, but as can be seen, it finds feasible solutions in reasonable times. Figure 3 only shows the average execution times for the cases when schedulable solutions have been found for each algorithm (see that in the tested cases, nor PD or NPD found solutions above 94% utilization). The time spent by the algorithm when a solution is not found is adjustable with the maximum number of iterations and the limit on the response times that is used to stop the analysis.

HOSDA has been applied to many more examples with very similar results. The best results are obtained in systems with deadlines larger than periods, usually in those in which deadlines are proportional to the period multiplied by the number of tasks in the transaction.

In some cases, the k_R and k_a constants used in the calculation of the excess (14) can be rebalanced to minimize the influence of the excess in the resource (e.g., by assigning values of k_R=10k_a, or k_R=100k_a) or the excess in the transactions (e.g., by assigning values of k_a=10k_R, or k_a=100k_R), which increase the possibility of reaching a feasible solu-

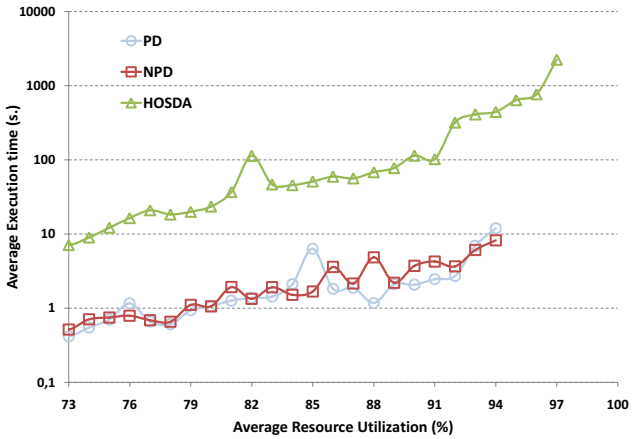

Figure 3. CPU time spent by the three algorithms

tion. As future work we plan to automate the selection and application of different sets of k_a and k_R, in order to seek better deadline assignments.

6. Conclusions

In this paper we propose a heuristic algorithm for assigning scheduling deadlines in distributed hard real-time systems. It is an adaptation to EDF of a previous algorithm called HOPA, designed for fixed priorities. We have shown that this method can find feasible optimized solutions in a reasonable amount of time, even in situations where the utilization of the resources is high and thus the number of solutions is small.

Since HOSDA is based on iteratively applying response time analysis and previous techniques for EDF scheduling were based on global deadlines, in this paper we have extended the analysis techniques to support local deadlines, which are more useful in distributed environments because they do not require global clock synchronization.

We have compared HOSDA with two other reference algorithms previously proposed to address the same problem but which are not able to optimize. The results of the comparison are that our method finds feasible solutions in many cases where these other methods fail. This is especially noticeable when utilizations are high or when the deadlines of the transactions are larger than the periods of the events that release them.

The quality of the results obtained by HOSDA are quite similar to those obtained by HOPA. Although both use the concept of excess to assign virtual or scheduling deadlines, they define these concepts in a different way. We are currently working on comparing whether HOPA for fixed priorities or HOSDA for EDF can find better solutions to schedule the same system. We are also working on the assignment of scheduling parameters in mixed systems

where some processing resources are scheduled with fixed priorities while the others are scheduled with EDF. Finally, we also plan to test HOSDA with more advanced RTA techniques, such as those based on offsets, and to give HOSDA the capability to manage preassigned or local deadlines, i.e. respectively, deadlines that cannot be changed or that can be less than or equal to a specific value.

REFERENCES

[1] T. P. Baker, "Stack-based scheduling for realtime processes," Real-Time Systems, v.3 n.1, pp.67-99, March 1991

[2] Claas Diederichs, Ulrich Margull, Frank Slomka, and Gerhard Wirrer, "An application-based EDF scheduler for OSEK/VDX," Proceedings of the conference on Design, Automation and Test in Europe, Munich (Germany), pp. 1045-1050, 2008.

[3] M. Di Natale, and A. Meschi, "Scheduling Messages with Earliest Deadline Techniques," Real-Time Systems, 20, pp. 255-285, Kluwer Academic Publishers, 2001.

[4] ERIKA (Embedded Real tIme Kernel Architecture) home page, `http://erika.sssup.it/`

[5] M. González Harbour, J.J. Gutiérrez, J.C. Palencia, and J.M. Drake, "MAST: Modeling and Analysis Suite for Real Time Applications," Proceedings of 13th Euromicro Conference on Real-Time Systems, Delft (The Netherlands), pp. 125-134, 2001.

[6] J.J. Gutiérrez, and M. González Harbour, "Optimized Priority Assignment for Tasks and Messages in Distributed Real-Time Systems," Proceedings of 3rd Workshop on Parallel and Distributed Real-Time Systems, Santa Barbara (California), pp. 124-132, 1995.

[7] Praveen Jayachandran, Tarek Abdelzaher, "A Delay Composition Theorem for Real-Time Pipelines," Proceedings of 19th Euromicro Conference on Real-Time Systems (ECRTS'07), pp. 29-38, IEEE, 2007.

[8] Jan Jonsson, and Kang G. Shin, "Robust Adaptive Metric for Deadline Assignment in Distributed Hard Real-Time Systems," Real-Time Systems, 23, pp. 239-271, Kluwer Academic Publishers, 2002.

[9] Jan Jonsson, and Kang G. Shin, "Deadline Assignment in Distributed Hard Real-Time Systems with Relaxed Locality Constraints," Proc. of the IEEE Int'l Conf. on Distributed Computing Systems (ICDCS'97), Baltimore (Maryland) pp. 432-440, 1997.

[10] B. Kao, and H. García-Molina, "Deadline Assignment in a Distributed Soft Real-Time System," IEEE Transactions on Parallel and Distributed Systems, Vol. 8, No. 12, pp. 1268-1274, 1997.

[11] J. Liu, "Real-Time Systems," Prentice Hall, 2000.

[12] MAST home page, `http://mast.unican.es/`

[13] J.C. Palencia, and M. González Harbour, "Offset-Based Response Time Analysis of Distributed Systems Scheduled under EDF," Proceedings of the 15th Euromicro Conference on Real-Time Systems, ECRTS, Porto (Portugal), 2003.

[14] P. Pedreiras, and L. Almeida, "EDF Message Scheduling on Controller Area Network," Computing & Control Engineering Journal, 13(4), pp. 163-170, 2002.

[15] R. Pellizzoni, and G. Lipari, "Improved Schedulability Analysis of Real-Time Transactions with Earliest Deadline Scheduling," Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium, RTAS, pp. 66 - 75, 2005.

[16] R. Pellizzoni, and G. Lipari, "Holistic analysis of asynchronous real-time transactions with earliest deadline scheduling," Journal of Computer and System Sciences, Volume 73, Issue 2, pp. 186-206, 2007.

[17] Juan M. Rivas, et al. "Optimized Deadline Assignment for Tasks and Messages in Distributed Real-Time Systems". Technical report, University of Cantabria. URL: `http://www.ctr.unican.es/publications/jmr-jjg-jcp-mgh-2009a.pdf`

[18] S.Ha.R.K. (Soft Hard Real-Time Kernel) home page, `http://shark.sssup.it/`

[19] M. Spuri, "Analysis of Deadline Scheduled Real-Time Systems," RR-2772, INRIA, France, 1996.

[20] M. Spuri, "Holistic Analysis for Deadline Scheduled Real-Time Distributed Systems," Tech. Rep. RR-2873, INRIA, France, April 1996

[21] S. Tucker Taft, Robert A. Duff, Randall L. Brukardt, Erhard Ploedereder, and Pascal Leroy (Eds.), "Ada 2005 Reference Manual. Language and Standard Libraries. International Standard ISO/IEC 8652:1995(E) with Technical Corrigendum 1 and Amendment 1," LNCS 4348, Springer, 2006.

[22] K. Tindell, and J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems," Microprocessing & Microprogramming, Vol. 50, Nos.2-3, pp. 117-134, 1994.

[23] RTSJ (Real-Time Specification for Java) home page, `http://www.rtsj.org/`

Appendix A. EDF Response-Time analysis for Local Deadlines

In this appendix we will fully describe the extension of the holistic analysis [20][22] for EDF distributed systems to support local deadline schedulers. Even though the new analysis is based on similar principles, the underlying model and the resulting equations are different. In the holistic analysis we assume for each analysis step that every task is independent of the other tasks, even from those belonging to the same transaction. After each analysis step, the dependencies are captured into the jitter terms of each task. Therefore we will rely on this assumption of independent tasks for the purpose of each analysis step. With this assumption, each task assumes a possibly different start of the transaction period, which is impossible in

practice and leads to pessimism in the results of the analysis. In this section we will refer to the start of the period of the *n*-th job of a task τ_{ab} as t_{nab}. With this notation, the release time of that job is in $[t_{nab},t_{nab}+J_{ab}]$.

Response time analysis is based on the creation of the longest busy period. A busy period is defined for EDF scheduling as an interval of time during which the CPU is busy processing pending jobs of any task. In fixed priority scheduling, the worst-case response time of a task τ_{ab} is found from a critical instant, when the release of τ_{ab} coincides with the release of all tasks with higher priority after having experienced the maximum jitter. In that situation, the critical instant coincides with the start of a busy period. In EDF scheduling that property is not true, but the busy period concept is still useful. The following theorem helps us to find the critical instant for a task in the context of the independent tasks assumption. It was proven by Spuri [19] for global deadlines but we rework the proof here to adapt it to local deadlines.

Theorem 1. The worst-case response time of a task τ_{ab} is found in a busy period in which each task τ_{ij} in the same processor (different from τ_{ab}) is scheduled such that its first job that is inside the busy period is released at the beginning of that busy period, after having experienced its maximum jitter (i.e., the start of the corresponding task job's period was J_{ij} time units before the start of the busy period), and the remainder of the jobs are released with the minimum jitter that makes the job start inside the busy period.

Proof: Without loss of generality let us assume the start of the busy period to the origin of time, t=0. Let t_0 be the instant at which a task τ_{ij} is released the first time in the busy period, and let D be the absolute deadline of an instance of the analyzed task, τ_{ab}. Suppose that t_0 does not coincide with the beginning of the busy period: in this circumstance, if we move the release pattern of τ_{ij} to occur earlier, down to the point when the first release coincides with the beginning of the busy period, it is possible that new releases occur inside the busy period, making it longer. The deadlines of each release of τ_{ij} will be earlier, so a release with a deadline after instant D may have been moved to have a deadline before that time, and in that case the response time of that job of task τ_{ab} would be increased. In addition, if this first job of τ_{ij} would not have experienced its maximum jitter, by increasing it and simultaneously decreasing times of the release pattern so that the job continues to coincide with the start of the busy period, we would cause the following jobs to be released earlier, which would also cause their deadlines to be earlier and possibly increment the response time of τ_{ab}.

Under these conditions in which the first job of τ_{ij} in the busy period experiences its maximum jitter, we can anticipate the deadlines of the following jobs of τ_{ij}, and possibly increase the response time of $\tau_{ab,}$ if these jobs are released as early as possible inside the busy period, by changing their jitters to the minimum value that makes them fall inside the busy period. ❒

Note that, contrary to the other tasks, releasing the analyzed task at the start of the busy period may not lead to its worst-case response time. If we move the release pattern of τ_{ab} to occur later, we are causing the deadline at D to be later too, and this could imply that some deadlines of other tasks that previously occurred after D could now occur before the modified D, and thus make the response time of task τ_{ab} become larger. So, the critical instant for a task is found in a busy period that is started by the simultaneous release of all tasks except perhaps the one under analysis.

In order to calculate the worst-case response time of task τ_{ab}, we will now calculate, under the conditions of theorem 1, the worst-case contribution of a task τ_{ij} to a busy period of length l when the deadline of τ_{ab} occurs at instant D. We will name this contribution $W_{ij}(l,D)$. Figure 4 shows a scenario for calculating this contribution.

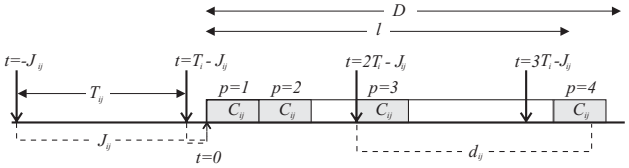

Figure 4. Scenario for the worst-case contribution

When we calculate the worst-case contribution of a task τ_{ij} to the response time of τ_{ab} we must consider from the releases that occur in the interval $[0,l)$ only those with deadline before or at D. Each of the releases will be identified with a sequence number p, starting at p=1. In Figure 4, release p=4 occurs before t, but its deadline is after D, so under the EDF rules it must not be considered for the worst-case contribution.

To calculate the number of releases of τ_{ij} in the busy period we can see that the identifier of the last release in that busy period, p_l, is the only value of p that simultaneously satisfies:

$$(p-1)T_i - J_{ij} < l \tag{16}$$

and:

$$pT_i - J_{ij} \geq l \tag{17}$$

from which we get:

$$p < \frac{l + J_{ij}}{T_i} + 1 \qquad \text{and} \qquad p \geq \frac{l + J_{ij}}{T_i} \tag{18}$$

Given that p_l is an integer number, the solution to the above two expressions is:

$$p_l = \left\lceil \frac{l + J_{ij}}{T_i} \right\rceil \quad (19)$$

To calculate the last release that verifies the deadline condition, p_D, we need to distinguish between those releases produced at the start of the busy period (corresponding to jobs with periods that started before the busy period and with deadlines at $t=d_{ij}$) from those releases that are after the start of the busy period. In the latter case, p_D is the only value of p that simultaneously verifies:

$$-J_{ij} + (p-1)T_i + d_{ij} \le D \quad (20)$$

and:

$$-J_{ij} + pT_i + d_{ij} > D \quad (21)$$

from which we get:

$$p \le \frac{J_{ij} + D - d_{ij}}{T_i} + 1 \quad \text{and} \quad p > \frac{J_{ij} + D - d_{ij}}{T_i} \quad (22)$$

so, we get the expression:

$$p_D = \left\lfloor \frac{J_{ij} + D - d_{ij}}{T_i} \right\rfloor + 1 \quad (23)$$

In case the last release inside the busy period would be one released at $t=0$, the above condition is only true if $D \ge d_{ij}$ (because if $D<d_{ij}$, the value $p_D = 0$). Therefore, the number of releases of task τ_{ij} with deadlines smaller than D is:

$$p_D = \begin{cases} 0 & if\ D < d_{ij} \\ \left\lfloor \frac{J_{ij} + D - d_{ij}}{T_i} \right\rfloor + 1 & otherwise \end{cases} \quad (24)$$

Given that the releases that contribute to the worst case are those with $p \le p_l$ and $p \le p_D$, using eqs. (19) and (24) the worst-case contribution of task τ_i to the busy period is:

$$W_{ij}(l, D) = min\left(p_l, p_D\right) \cdot C_{ij} \quad (25)$$

Using this expression we can calculate the worst-case response time of task τ_{ab}. The worst-case situation for the analyzed task occurs when all its jobs experience their maximum jitter, because in this way release times are as late as possible (compared to the start of the period to which the response times are relative) and so are the response times. Unfortunately, we don't know how to phase the release time of τ_{ab} in relation to the busy period, but it is easy to see that the worst case situation must be found when the release time is placed at the beginning of the busy period, or at an instant such that the deadline of the analyzed job of τ_{ab} coincides with the deadline of a task τ_{ij}'s job. Otherwise the release of task τ_{ab} could be moved to the previous such earlier time without changing the execution schedule, but making the response time larger. The set of instants, Ψ_{ij}, at which the deadline of τ_{ab}'s job coincides with the deadlines of one of the task jobs in the busy period is:

$$\Psi_{ij} = \bigcup \{(p-1)T_i - J_{ij} + d_{ij}\} \quad \forall p = 1 \ldots \left\lceil \frac{L + J_{ij}}{T_i} \right\rceil \quad (26)$$

where L corresponds to longest busy period, calculated as:

$$L = \sum_{\forall \tau_{ij}} \left\lceil \frac{L + J_{ij}}{T_i} \right\rceil \cdot C_{ij} \quad (27)$$

The equation above is one of many recurrence equations found in response time analysis in which the value to be calculated is in both sides of the equation; of the many solutions, only the one with the minimum positive value is valid. These equations can be easily solved iteratively by starting with a small value of L and using the value obtained from the equation in the next iteration, until a stable solution is found. The equation is guaranteed to have a solution if the utilization of the task set is under 100%. Although the computation time is pseudopolynomial, it is usually short except for utilizations very close to 100%.

This set Ψ_{ij} must be augmented with the deadlines corresponding to task τ_{ab}

$$\Psi_{ab} = \bigcup \{(p-1)T_a + d_{ab}\} \quad \forall p = 1 \ldots \left\lceil \frac{L}{T_a} \right\rceil \quad (28)$$

And so the full set of situations for which τ_{ab} has to be analyzed corresponds to those releases whose deadline is in the set

$$\Psi = \Psi_{ij} \cup \Psi_{ab} \quad (29)$$

Each potential release time for τ_{ab} is obtained by subtracting d_{ab} from each value in Ψ. Checking the response times under all these release times we can find the one that causes the worst-case response time of the task. Given that there may be several releases of τ_{ab} in the busy period, we must analyze them all. For each value $\psi \in \Psi$, the completion time of release p of τ_a, $w_{ab}^{\psi}(p)$, can be calculated by adding the worst-case contribution of all tasks in the same processor, and the blocking term:

$$w_{ab}^{\psi}(p) = B_{ab} + pC_{ab} + \sum_{\forall ij \ne ab} W_{ij}(w_{ab}^{\psi}(p), \psi) \quad (30)$$

The worst-case response time is calculated by subtracting the start of the job's period from the resulting completion time:

$$R_{ab}^{\psi}(p) = w_{ab}^{\psi}(p) - (\psi - d_{ab} - J_{ab}) \qquad (31)$$

For each value of p, we only need to check the values of ψ in one period, because if the release time corresponding to ψ was greater than the corresponding period, then we would be analyzing another job with a different value of p. This allows us to restrict the set of values to be checked:

$$\Psi^* = \{\psi_x \in \Psi \,|\, (p-1)T_a + d_{ab} \le \psi_x < pT_a + d_{ab}\} \qquad (32)$$

Finally, to calculate the worst-case response time of task τ_{ab} we must determine the maximum response times within all the potential release times examined:

$$R_{ab} = max(R_{ab}^{\psi}(p)) \quad \forall p = 1 \ldots \left\lceil \frac{L}{T_a} \right\rceil, \forall \psi \in \Psi^* \qquad (33)$$

We can now feed these response times into the holistic analysis loop like in [20], obtaining new jitter values from the response times and repeating the analysis until a stable solution is obtained. Since the dependencies of response times on jitters are monotonically increasing, the algorithm is known to converge to the final solution, except when the utilization is close to 100% and in special cases that experience shows that are uncommon.

Appendix B. HOSDA Algorithm

This appendix provides a more complete description of the HOSDA algorithm. From a high-level point of view, the algorithm is:

```
algorithm HOSDA is
begin
   assign initial scheduling deadlines;
   loop
      calculate worst-case response times;
      exit when some stopping criterion;
      calculate new scheduling deadlines;
   end loop;
end HOSDA;
```

The scheduling deadlines should be assigned in some way that preserves the global deadlines:

$$D_{ij} = \sum_{k \in pr_i(j)} d_{ik} \qquad (34)$$

where:

- D_{ij} is the global deadline (intermediate or end-to-end) of task τ_{ij}.
- d_{ij} is the local scheduling deadline of task τ_{ij}.
- $pr_i(j)$ is the set of tasks preceding task τ_{ij} in the transaction i to which it belongs, including itself.

One possible initial assignment is to distribute the deadline proportionally to the worst-case execution time of each task (C_{ij}), which is similar to proportional deadline assignment algorithm proposed in [11].

After all the scheduling deadlines have been assigned, the system is analyzed using the technique described in Section 3 which is adapted to systems with local deadlines. The analysis provides the worst-case response times for each task, for that scheduling deadline assignment. If the solution is not schedulable, new scheduling deadlines are calculated by redistributing the global deadlines among the tasks of each transaction.

The redistribution of local deadlines uses the concept of "*excess*" of each task which, intuitively and in the same way than in HOPA, measures the distance that separates each task from schedulability. The new scheduling deadlines are obtained as a function of two factors: on the one hand, the excess of each task relative to other tasks in the same transaction; on the other hand, the excess of each task relative to the other tasks in the same processing resource. To avoid convergence problems or very long calculations that the holistic analysis may have when utilizations are very high, we have added a termination condition to the holistic analysis that makes it stop when the response time of a task exceeds the imposed deadline by a configurable factor. In this way we bound and shorten the analysis time and we assure that the assignment algorithm can continue working.

The original algorithm for fixed priorities had two different definitions for the excess of a particular task τ_{ij} that made it behave differently. The first definition, "excess of response time", was based on the difference between the local response time and the local deadline, and led to faster solutions; the second definition, "excess of compute time", was based on the calculation of the slack time and led to better solutions because, for deadlines smaller than periods, the slack time is a much better representation of the excess. The slack time of a task τ_{ij} is defined as the amount of execution time that must be subtracted from task τ_{ij} for it to meet its local deadline d_{ij}, if τ_{ij} is not schedulable, or the amount of execution time that may be added to task τ_{ij} for it to continue being schedulable, if it is schedulable.

For the HOSDA algorithm we have made extensive experiments trying to find a suitable definition for the excess that would give reasonable results in most cases,

and as a result we have redefined the "excess of response time" in two different ways:

$$exc(\tau_{ij}) = \begin{cases} (R_{ij} - J_{ij} - d_{ij})\dfrac{\Delta R_{ij}}{\Delta D_{ij}} & (1) \\ or & \\ (R_{ij} - d_{ij})\dfrac{\Delta R_{ij}}{\Delta D_{ij}} & (2) \end{cases} \quad (35)$$

where:

ΔR_{ij} is the difference between the worst case global response times of the task before τ_{ij} that has a global deadline and the task after τ_{ij} that also has a global deadline.

ΔD_{ij} is the difference between the global deadlines of the tasks before and after τ_{ij} that have a global deadline.

If there is no previous task with a global deadline, then 0 is used to find the difference in ΔR_{ij} and ΔD_{ij}. In the common case when there is only a single end-to-end deadline for the transaction all the ΔR_{ij} terms of the same transaction have the same value related to the response time of the last task in the transaction, $\Delta R_{ij} = R_{im_i}$, and the same applies to $\Delta D_{ij} = D_i$.

Both definitions of excess can lead separately to feasible deadline assignment. As in our tests the majority of the cases are solved with definition (2) this is the recommended option to try first.

Currently we do not use the excess of computation time as it is defined in [6] because the calculation of slack time requires a repetitive utilization of the analysis algorithm, and given the much longer computation times of the analysis for EDF scheduling there is a huge cost in time of analysis except for very small examples.

For each processing resource PR_k we define the excess in that resource as:

$$exc(PR_k) = \sum_{\tau_{ij} \in PR_k} exc(\tau_{ij}) \quad (36)$$

In addition, we define the maximum excess of all the processing resources *PR*, and the maximum of the excesses of all the tasks belonging to a particular transaction responding to external event e_i as the maximum of the absolute values of their corresponding values:

$$\begin{aligned} Mex(PR) &= \max_{\forall PR_i}(|exc(PR_i)|) \\ Mex(e_i) &= \max_{\forall j}(|exc(\tau_{ij})|) \end{aligned} \quad (37)$$

We use the absolute value because the maximum is used to normalize the individual excess values in Eq. (38) below.

Given these definitions of excess times, we calculate the new scheduling deadline for each task as a function of the old scheduling deadline for that task, the excess for that task, and the excess for the resource to which that task belongs, as:

$$d_{ij}(new) = d_{ij}(old)\left(1 + \frac{exc(PR_k)}{k_R \cdot Mex(PR)}\right)\left(1 + \frac{exc(\tau_{ij})}{k_a \cdot Mex(e_i)}\right) \quad (38)$$

where k_R and k_a, like in HOPA, are constants that control the relative influence of the processing resource and task components, respectively, in the calculation of the new deadline. The smaller these constants are, the higher this influence is. HOSDA is defined to use an array of pairs of constants allowing the algorithm to evolve with the current pair for a bounded number of iterations; then the pair is changed to the next one. By extensive experimentation, we have found that recommended empirical values for these constants are between 1.0 and 3.0. Normally, in the HOSDA algorithm we start with values of k_R=k_a=1.5, and we then change both values to 2.0, 2.5, 3.0, etc., until one stopping condition is reached.

Once the new local deadlines have been obtained for all the tasks in the transaction, we adjust them proportionally to make them fit into the global deadlines. For an end-to-end deadline we do:

$$d_{ij} = \frac{d_{ij}(new)}{\sum_{k \in pr_i(j)} d_{ik}} D_{ij} \quad (39)$$

The HOSDA algorithm stops whenever one of the following stopping conditions is met:

- A schedulable solution has been found.
- Two consecutive deadline assignments are identical (in which case the algorithm would continue providing the same solution).
- A maximum number of iterations has been reached. This number is configurable and, for a given size of the system, it sets a limit to the time that the user is willing to wait for obtaining a solution.

After the algorithm finds a deadline assignment that makes the system schedulable, it is capable of finding a better optimized solution by executing more iterations, up to an adjustable limit. The best way to compare the schedulability of a system is by making a sensitivity analysis using the slack of the system as it is defined in the MAST

tools [12]. However, sensitivity analysis is slow, and in order to faster determine how good a solution is, compared to others, we define a normalized schedulability index, the transaction index, which is as a function of the worst distance between the worst-case response times and the global deadlines for each transaction:

$$TransactionIndex = \frac{(D_i - R_i)}{D_i} \quad (40)$$

Finally, the normalized schedulability index is the average of the transaction indexes for all the transactions of the system. The larger the index becomes, the better the schedulability of the system is in terms of distance between the response times and the deadlines.