Task Partitioning and Priority Assignment for Hard Real-time Distributed Systems

Ricardo Garibay-Martínez, Geoffrey Nelissen, Luis Lino Ferreira, Luís Miguel Pinho
CISTER/INESC-TEC, ISEP
Polytechnic Institute of Porto, Portugal
{rgmz, grrpn, llf, lmp}@isep.ipp.pt

***Abstract*—The partitioning of fixed-priority hard real-time tasks and messages in a distributed system is a well know NP-hard problem. Therefore, there are no methods that provide an optimal solution in polynomial time. In this paper, we propose the Distributed using Optimal Priority Assignment (DOPA) heuristic, which simultaneously solves the problem of assigning task to processors and assigning priorities to tasks. DOPA makes use of Audsley's Optimal Priority Assignment (OPA) algorithm to assign priorities to tasks and messages. However, in order to use the OPA algorithm for task sets with dependencies, we first transform the task set into a set of independent tasks by imposing intermediate deadlines. The experimental results show how the utilisation of the OPA algorithm increases in average the number of schedulable tasks and messages in a distributed system when compared to the utilisation of the Deadline Monotonic (DM) priority assignment usually used in other works.**

Keywords—real-time; distributed systems; task allocation; priority assignment; intermediate deadlines; holistic analysis.

I. INTRODUCTION

Real-time distributed systems are present in our everyday life. These systems range from safety critical to entertainment and domestic applications, presenting a very diverse set of requirements. Although diverse, in all these areas, modern distributed applications are becoming larger and more complex. Therefore, integrating the system's functional requirements to comply with their associated real-time constraints has shown to be a challenging problem within the real-time domain.

Hard real-time distributed systems are composed of two main elements: (i) a set of real-time *applications* and (ii) a distributed computing *platform* that executes such applications. Applications are composed of a set of tasks that communicate through messages to perform a certain functionality (e.g. realizing input/output operations, processing data, etc.). On the other hand, distributed platforms are composed of a set of processing elements (e.g. processors, ECUs, etc.) and networks, that provide the needed computational resources for tasks to be executed and messages to be transmitted.

When considering real-time applications, the processing of tasks and messages must comply with their associated time constraints. Commonly, for applications, this time constraint is expressed by an *end-to-end deadline*, which is the longest elapsed time a sequence of tasks and messages (an application) is permitted to take from the time at which it was activated until it completes its execution.

Furthermore, for a given set of applications and a given computing platform, the main objective is to find a *feasible allocation* for tasks and messages in a way that all application's end-to-end deadlines are met. Unfortunately, this problem is known to be *NP-hard* [1]. The problem of task allocation can be viewed as a two-sided problem: (i) finding the partitioning of tasks and messages onto the processing elements of the distributed system, and (ii) finding the priority assignment for tasks and messages for that partition, so that real-time tasks and messages are executed within their deadlines. Therefore, a careful trade-off between the solutions of those two subproblems needs to be taken in order to obtain a correct global solution.

Contribution. This paper presents the Distributed using Optimal Priority Assignment (DOPA) heuristic to find a feasible partitioning and priority assignment for tasks in distributed computing platforms by using the Optimal Priority Assignment (OPA) algorithm, known as Audsley's algorithm [2]. The algorithm is an optimal priority assignment algorithm for independent fixed priority tasks on uniprocessor systems. The OPA algorithm requires tasks to be independent, therefore, in order to use the OPA algorithm for task sets with dependencies (applications), we first need to transform tasks with dependencies to a set of independent tasks by imposing *intermediate deadlines*. Also, the use of intermediate deadlines makes our approach easily extensible to more powerful task models such as multithreaded parallel real-time models, when compared to previous approaches. Furthermore, our simulations show how the use of the OPA algorithm increases, in average, the number of schedulable task and messages in a distributed system, when compared to the usual method consisting in using the Deadline Monotonic (DM) priority assignment [3].

Structure of the paper. The remainder of the paper is structured as follows. Section II presents the related work, whilst Section III introduces the system model. Section IV presents the DOPA heuristic. Section V shows some experimental results, and finally, in Section VI we draw our conclusions and propose future work.

II. Related Work

The problem of task allocation is divided in two sub problems: finding the partitioning of tasks and messages onto the distributed system, and finding the priority assignment for that partition. In this section we present some relevant works that address such problems, nevertheless restraining our attention to the case of preemptive fixed-priority task scheduling based approaches.

In [4], Tindell *et al.*, addressed this issue as an optimisation problem with the general purpose *simulated annealing* algorithm. The simulated annealing algorithm is used for iterating in a random manner, over a given allocation for tasks and messages to processors and networks, and performs an evaluation based on an "energy" function, which evaluates the quality of the encountered solution (allocation). Tindell used the DM scheduling algorithm [3] to assign priorities to periodic tasks with constrained deadlines assuming that each task in an application has its own deadline and period. The latter assumption may however not always be true in real systems.

In [5], Gutierrez *et al.*, proposed an optimisation technique for the priority assignment of tasks and messages in a distributed system. They assumed a set of tasks and messages that are statically allocated to processors and networks (therefore, no partitioning phase is considered); thus, focusing on the problem of assigning the priorities to the allocated tasks and messages. Their method is based on imposing intermediate deadlines to the tasks and messages that compose a "sequence of actions" and then using DM to assign the task priorities.

The problem of partitioning tasks and messages in distributed systems is also addressed in Richard *et al.* [6]. They propose a solution based on branch-and-bound; branching (enumerating) the possible paths that can lead to and allocation, and bounding (cutting the path) whenever a feasible schedule cannot be reached by following such a path. Again, DM is used to assign the priorities assuming that each task is defined by its own deadline and period. The bounding step is performed by checking the schedulability of each branch based on the schedulability technique for RMA derived by Tindell *et al.* in [7].

In [8] and [9], the authors model the task partitioning problem as an optimisation problem. However, this work still assumes that each task has its own period and deadline, and it uses DM to assign priorities.

More recently, Azketa et al. [10] addressed the problem of task and message allocation in a distributed system by taking hand of the general purpose genetic algorithms. They use a genetic algorithm with a permutational solution encoding. They initiate their genetic algorithm by assigning priorities using the HOPA heuristic [5] which is based on DM priority assignment [3] and iterate over different solutions by applying crossover, mutation and clustering operations. To test schedulability they use the Tindell's holistic analysis [7] and Palencia's schedulability tests [11, 12].

Although there are some similarities between our method and previous works, none of the previous approaches has used Audsley's OPA to assign priorities to tasks in a distributed system. As proved in [2], the OPA algorithm is optimal for the case of preemptive fixed priority tasks, thereby implying that if the system is schedulable with DM then OPA will also find a priority assignment to schedule the task set. Further, DM has been proved to not be optimal for systems where all tasks do not release jobs simultaneously [3], which is typically the case in distributed systems due to the task precedence constraints. By adding intermediate deadlines to tasks, we show that the Audsley's algorithm can be used and that it increases the number of schedulable task sets (i.e. applications) in comparison with DM.

III. System Model

A. Real-Time Applications

A distributed real-time system is composed of software applications that we model as a set $\Gamma = \{\Gamma_1, \dots, \Gamma_n\}$ of n concurrent sequential applications Γ_i with $i \in \{1, \dots, n\}$. An application Γ_i is composed of a set $\tau_i = \{\tau_{i,1}, \dots, \tau_{i,n_i}\}$ of n_i tasks and a set of $\mu_i = \{\mu_{i,1}, \dots, \mu_{i,n_i-1}\}$ of $n_i - 1$ messages, which are executed and transmitted on processors and networks, respectively. We assume the *linear model of event-driven distributed system* [13]. In this model, each application Γ_i is activated by an external event e_i with a minimum inter-arrival time of T_i. The arrival of an external event e_i is followed by the activation of the first task $\tau_{i,1}$ of Γ_i. Whenever a task $\tau_{i,j}$ completes its execution, it sends a message $\mu_{i,j}$ to the next task $\tau_{i,j+1}$ and hence triggers its execution.

Each task $\tau_{i,j}$ is characterized by its Worst-Case Execution Time (WCET) $C_{i,j}$, and each message $\mu_{i,j}$ by a transmission time $C_{i,j}^{msg}$. Communications between tasks can be carried out within the same or different processors in the distributed system. If two tasks $\tau_{i,j}$ and $\tau_{i,j+1}$ communicate via a message $\mu_{i,j}$ and execute in the same processor, we consider that the message transmission time is negligible, thereby assuming that $C_{i,j}^{msg} = 0$. Also, an application Γ_i is characterised by an end-to-end deadline D_i, which is the longest elapsed time that the sequence of tasks and messages is permitted to take from the time in which it is activated (e_i) until it completes its execution (time at which the last task τ_{i,n_i} in the sequence of tasks and messages completes its execution). We assume that $D_i \leq T_i$. The density δ_i of an application Γ_i is given by $\delta_i = \frac{\sum_{j=1}^{n_i} C_{i,j} + C_{i,j}^{msg}}{D_i}$ and the total density of the system is defined as $\delta = \sum_{\Gamma_i \in \Gamma} \delta_i$.

We consider that tasks are scheduled with a preemptive fixed-priority algorithm. On the other hand, messages are scheduled with a non-preemptive fixed priority algorithm. We also consider that some tasks of an application can be

restrained to execute in some specific processors due to some *design constraints*. Such constraints can be related to design reasons, safety reasons, or to specific functionality offered by the processors in the distributed system (e.g. sensors, actuators, required libraries, etc.), and required for the execution of a task $\tau_{i,j}$ within an application Γ_i. Therefore, there exists a set $A \subseteq \Gamma$ of tasks that are resource constrained (they need to be executed in a specific processor), thus, those tasks are statically assigned to those processors. Also, there exist a set $\Upsilon \subseteq \Gamma$ of tasks that do not have any resource constrains and can be allocated onto any processor.

B. Distributed Computing Platform

A distributed computing platform is composed of a set of processors that provide the computing power to execute tasks, which are connected through a fixed-priority real-time network (e.g. a CAN network [14]). We assume a set $\pi = \{\pi_1, \ldots, \pi_m\}$ of m identical uniprocessor nodes for the execution of the tasks, and a single shared real-time network ϖ for message transmission.

Figure 1 shows an example of a computing platform composed of three processors and one real-time network. There are three applications Γ_1, Γ_2 and Γ_3, each composed of only one task ($\tau_{1,1}$ and $\tau_{2,1}$ and $\tau_{3,1}$). Tasks $\tau_{1,1}, \tau_{2,1}, \tau_{3,1}$ and $\tau_{5,1}$ are resource constrained (thus belonging to the set A) being pre-assigned to the specific processors 1, 2, 3 and 1 respectively. Also, there exists a list Υ of unallocated tasks $\tau_{4,1}, \tau_{4,2}$, $\tau_{5,2}$, $\tau_{6,1}$ and $\tau_{6,2}$ that can be allocated to any processor.

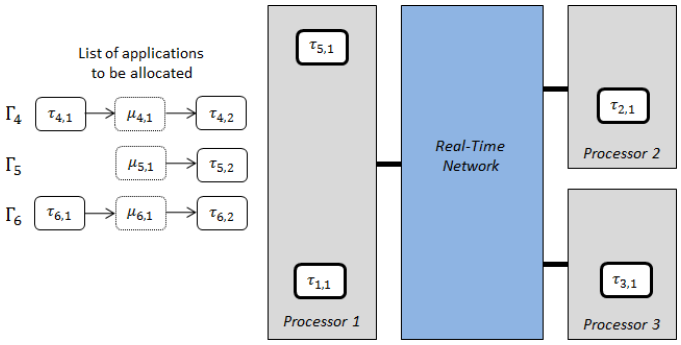

Figure 1. List of applications to be allocated in a computing platform.

The objective is to find (i) a feasible partitioning of the tasks constituting the applications onto the processors and (ii) a priority assignment to the tasks in a way that all end-to-end deadlines are met. Figure 2, shows an example of allocation α^* of tasks and messages for the unallocated applications shown in Figure 1. By looking at Figure 2 it is possible to notice that tasks in application Γ_4 ($\tau_{4,1}, \tau_{4,2}$) are allocated to the same processor, and therefore the message $\mu_{4,1}$ can be neglected, thereby reducing its communication cost to $C_{i,k}^{msg} = 0$. In the following section we present an heuristic solving this problem.

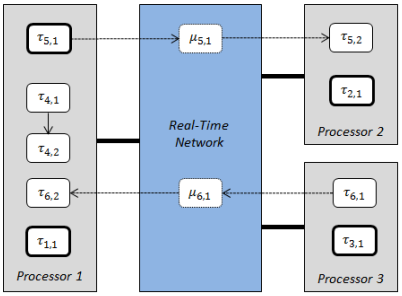

Figure 2. Example of a task allocation for the system presented in Figure 1.

IV. The DOPA Heuristic

The DOPA heuristic simultaneously addresses the two interrelated sub-problems of: (i) finding the partitioning of tasks and messages onto the elements of the distributed system, and (ii) finding the priority assignment for that partitioning.

A. Optimal Priority Assignment (OPA) algorithm

Regarding the problem of priority assignment, there exist several techniques to assign priorities to a set of preemptive independent tasks. DM [3] is the one usually considered in every work on distributed systems. DM is optimal for assigning priorities if there exists an instant in the schedule where all the tasks release a job simultaneously. However, in distributed systems tasks and/or messages have dependencies on other tasks or messages of the same application. Hence, because a task $\tau_{i,j+1}$ never starts its execution before the completion of task $\tau_{i,j}$, $\tau_{i,j}$ and $\tau_{i,j+1}$ will never release a job simultaneously, thereby violating the optimality condition of DM. One should therefore conclude that DM is not optimal for distributed systems. On the other hand, Davis and Burns [15] proved that the Audsley's OPA algorithm is optimal regarding the assignment of task priorities if there exists a schedulability test S respecting the three following conditions: (C1) the schedulability of a task $\tau_{i,j}$ according to S, may be dependent on the set of higher priority tasks $HP_{i,j}$, but not on the relative priority order of those tasks, (C2) the schedulability of a task $\tau_{i,j}$ according to a test S, may be dependent on the set of lower priority tasks, but not on the relative priority order of those tasks, and, (C3) for two tasks with adjacent priority, if their priorities are swapped, the task that has been assigned the higher priority cannot become unschedulable according to the test S, if it was schedulable at the lower priority.

```
1.  for each priority level k, lowest first{
2.     for each unassigned task τi,j{
3.        if (τi,j is schedulable at priority k according to
             test VERIFY_SCHEDULABILITY(τi,j → πk) with all
             unassigned tasks assumed to have higher
             priorities){
4.           assign τi,j to priority k
5.           break (continue outer loop)
6.        }
7.     }
8.     return unschedulable
9.  }
10. return schedulable
```

Figure 3. OPA algorithm pseudocode.

The OPA algorithm is based on three simple steps (see Figure 3): (i) check the schedulability according to S of all non-priority-assigned tasks by assuming they have the lowest priority, (ii) arbitrarily chose one that respects its deadline, and (iii) remove the chosen task from the list of non-priority-assigned tasks and start again. To verify the schedulability (VERIFY_SCHEDULABILITY($\tau_{i,j} \in \pi_k$)) of the task set, we use the schedulability analysis presented in [7]. Note however that other tests could also be used (e. g. [11, 12]). We know from [16] that the worst-case response time $r_{i,j}$ of an independent task $\tau_{i,j}$, scheduled with a preemptive fixed priority scheduling algorithm is given by the following equation:

$$r_{i,j} = C_{i,j} + \sum_{\tau_{a,b} \in HP_{i,j}} \left\lceil \frac{r_{i,j}}{T_a} \right\rceil C_{a,b} \quad (1)$$

where $HP_{i,j}$ is the set of tasks with a higher priority than $\tau_{i,j}$ that can interfere with $\tau_{i,j}$. Due to the presence of the term $r_{i,j}$ on both side of (1), this equation is usually solved in an iterative manner, $r_{i,j}^{k+1} = C_{i,j} + \sum_{\tau_{a,b} \in HP_{i,j}} \left\lceil \frac{r_{i,j}^{k}}{T_a} \right\rceil C_{a,b}$ with $r_{i,j}^{1} = C_{i,j}$.The iteration stops when $r_{i,j}^{k} = r_{i,j}^{k+1}$.

In a distributed time system, the worst-case response time $WCRT_{i,j}$ of a task $\tau_{i,j}$ can then be computed as [7]:

$$WCRT_{i,j} = r_{i,j} + \sum_{k=1}^{j-1} \left(r_{i,k} + r_{i,k}^{msg} \right) \quad (2)$$

where $r_{i,k}^{msg}$ is the response time of a message $\mu_{i,k}$ obtained with a network dependent analysis such as [14]. An application Γ_i (and hence its constituting tasks and messages) is deemed schedulable if $WCRT_{i,n_i} \leq D_i$.

Unfortunately, this schedulability test makes the schedulability of a task $\tau_{i,j}$ dependent on the response times and hence the priorities of all the other tasks and messages in Γ_i, thereby making OPA unusable. We therefore transform the tasks and messages with dependencies to an equivalent set of tasks and messages without dependencies by imposing an intermediate deadline $d_{i,j}$ ($d_{i,j}^{msg}$, resp.) to each task $\tau_{i,j}$ (each message $\mu_{i,j}$, resp.) as shown in Figure 4. The intermediate deadline $d_{i,j}$ of $\tau_{i,j}$ then becomes an offset on the release of the message $\mu_{i,j}$, and the deadline $d_{i,j}^{msg}$ of $\mu_{i,j}$, becomes an offset on the release of $\tau_{i,j+1}$. Therefore, we now have that:

$$\begin{cases} WCRT_{i,j} = d_{i,j-1}^{msg} + r_{i,j} \\ WCRT_{i,j}^{msg} = d_{i,j} + r_{i,j}^{msg} \end{cases} \quad (3)$$

implying that the worst-case response time of each task and message becomes independent of the relative priority order of higher and lower priority tasks. Now, a task $\tau_{i,j}$ (a message $\mu_{i,j}$, resp.) is deemed schedulable if $WCRT_{i,j} \leq d_{i,j}$ ($WCRT_{i,j}^{msg} \leq d_{i,j}^{msg}$, resp.) and the three Audsley's OPA algorithm validity conditions (C1, C2 and C3) are respected.

The tasks and messages intermediate deadlines are computed as a function of the application end-to-end deadline and the tasks and messages WCETs ($C_{i,j}$ and $C_{i,j}^{msg}$, respectively). For tasks and messages, the intermediate deadlines are given by:

$$d_{i,j} = d_{i,j-1}^{msg} + \frac{C_{i,j}}{\sum_{k=1}^{n_i} C_{i,k} + C_{i,k}^{msg}} D_i \quad (4)$$

$$d_{i,j}^{msg} = d_{i,j} + \frac{C_{i,j}^{msg}}{\sum_{k=1}^{n_i} C_{i,k} + C_{i,k}^{msg}} D_i \quad (5)$$

Note that from those definitions, we have that $d_{i,n_i} = D_i$. Hence, if all tasks (and messages) respect their intermediate deadlines $d_{i,j}$, i.e., $WCRT_{i,j} \leq d_{i,j}$, the end to end deadline D_i of the application is also respected.

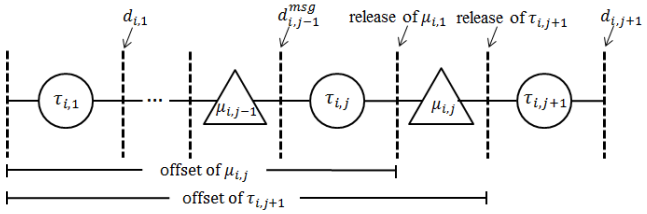

Figure 4. Intermediate deadlines.

B. Partitioning Algorithm

The problem of partitioning the task set onto the processors of the platform and assigning priorities to the tasks and messages is solved by the algorithm PARTITION(Γ) presented in Figure 5. The algorithm is entirely based on the following idea; if two successive tasks $\tau_{i,j}$ and $\tau_{i,j+1}$ of the same application Γ_i are assigned to a same processor π_k, then the message $\mu_{i,j}$ can be ommited, thereby reducing the load on the network and increasing the acceptable response time for the other tasks and messages in Γ_i. Therefore, PARTITION(Γ) tries to maximize the number of successive tasks of the same application being assigned on the same processor.

The pseudo code of the partitioning algorithm can be understood as follows. Applications $\Gamma_i \in \Gamma$ are assigned in a non-increasing density order. Tasks in Γ_i are considered in a lexical order. Each task $\tau_{i,j}$ is first assigned on the same processor than the previous task $\tau_{i,j-1}$ (if any) of the application Γ_i, thereby assuming that the message $\mu_{i,j-1}$ is unneeded and hence $C_{i,j-1}^{msg} = 0$. If the priority assignment (using OPA) does not succeed (i.e., the task is unschedulable on that processor) and the next task (if any) that must be executed in the application Γ_i has already been assigned on a processor π_k, then PARTITION(Γ) tries to assign $\tau_{i,j}$ on π_k assuming that the message $\mu_{i,j}$ is unneeded and hence $C_{i,j}^{msg} = 0$. If the priority assignment fails again, then the algorithm tries to assign $\tau_{i,j}$ on any other processor in a worst-

fit order (i.e., the processor with the smallest total density first). Finaly, the schedulability on the network is checked. Note that the intermediate deadlines of the tasks in Γ_i are recomputed at each step since their values depend on the number of messages the application must send on the network, i.e., the number of messages with $C_{i,j}^{msg} > 0$. However, this modification of the intermediate deadlines does not jeopardize the schedulability of the tasks that are already assigned to processors since, by studying Equations (4) and (5), we can see that the intermediate deadlines increase whenever a message is ommited (i.e., one of the terms $C_{i,k}^{msg}$ becomes equal to 0). Therefore, if the previous deadlines were respected, the new one will also be.

```
PARTITION(Γ)
1.   for all Γ_i ordered by non-increasing δ_i {
2.      for all τ_{i,j} ∈ Γ_i {
3.         assign τ_{i,j} to π_k| τ_{i,j-1} ∈ π_k, assuming C_{i,j}^{msg} = 0
4.         call OPA_ASSIGNMENT(τ_{i,j}, τ_{a,b} ∈ π_k)
5.         if OPA succeed to assign τ_{i,j} {
6.            break
7.         }
8.        else if OPA fails to assign τ_{i,j} {
9.            assign τ_{i,j} to π_k| τ_{i,j+1} ∈ π_k, assuming C_{i,j}^{msg} = 0
10.           call OPA_ASSIGNMENT(τ_{i,j}, τ_{a,b} ∈ π_k)
11.           if OPA succeed to assign τ_{i,j} {
12.              break
13.            }
14.           else if OPA fails to assign τ_{i,j} {
15.              for all π_k in Worst-Fit order {
16.
17.                 assign τ_{i,j} to π_k
18.                 call OPA_ASSIGNMENT(τ_{i,j}, τ_{a,b} ∈ π_k)
19.                 if OPA succeeds {
20.                    assign message μ_{i,j} to the network
21.                    VERIFY_SCHEDULABILITY(μ_{i,j} ∈ ϖ)
22.                    if message μ_{i,j} schedulable
23.                       break
24.                    else
25.                      return unschedulable
26.                 }
27.                 else
28.                    return unschedulable
29.              }
30.           }
31.        }
32.     }
32.  }
33.  return schedulable
```

Figure 5. Partitioning algorithm pseudocode.

V. EXPERIMENTAL EVALUATION

In this section we present some experimental results of our simulations of the DOPA heuristic. Let us recall that the DOPA heuristic simultaneously (i) finds the partitioning of tasks and messages onto the elements of the distributed system, and (ii) finds the priority assignment for that partitioning. For all experiments we use the `PARTITION(Γ)` algorithm for the partition of tasks and messages onto the elements of the distributed system, and we use two different priority assignment algorithms, namely DM and OPA.

One of the main objectives of this work is to demonstrate that by using the OPA algorithm, for the case of tasks with dependencies, it is possible to increase in average the number of schedulable tasks and messages in a distributed system when compared to the utilisation of the DM priority assignment, frequently used in other works.

For generating the applications Γ_i and their respective tasks $\tau_{i,j}$ and messages $\mu_{i,j}$ we follow the guidelines presented in [17] for generating random task sets for multiprocessor systems, using the Stafford's Randfixedsum algorithm [18]. The Randfixedsum algorithm generates a set of n values which are evenly distributed and whose components sum to a constant value. Thus, we use the Randfixedsum algorithm for generating unbiased sets of applications with a fixed total density $\delta_{tot} = \sum \delta_i$. For a given total density δ_{tot}, the algorithm returns n applications with density δ_i; with values from a minimum density bound $\delta_i^{min} = 0.1$ for each application, and a maximum density bound $\delta_i^{max} = 0.9$. For generating the tasks' and messages' densities we use again the Randfixedsum algorithm taking as an input the previous generated densities $\delta_i = \sum(\delta_{ij} + \delta_{i,j}^{msg})$, obtaining a set of values $\delta_{i,j} = d_{i,j}/T_i$ for tasks and $\delta_{i,j}^{msg} = d_{i,j}^{msg}/T_i$ for messages with values from a minimum density bound for tasks and messages $u_{i,j}^{min} = 0.01$ and a maximum density of $u_{i,j}^{max} = 0.9$. The WCETs of tasks $C_{i,j}$, messages $C_{i,j}^{msg}$ and end-to-end deadlines D_i are generated as recommended in [17]; we considered that applications have implicit end-to-end deadlines ($D_i = T_i$) following a uniform distribution. For each experiment 100 sets are generated.

Figure 6 (a) shows the number of accepted tasks sets over 100 experiments for different total densities δ_{tot}. We simulate 50 applications that execute tasks and transmit messages in a computing platform of 10 processors and 1 network. It is possible to see that OPA in average performs better in terms of the number of accepted task sets. For example, the OPA algorithm accepts 52% of task sets with total system density of 9. In contrast, the DM algorithm reaches 16% with the same system density.

In Figure 6 (b) we show the number of accepted task sets for 100 experiments simulating 50 applications that execute tasks and transmit messages in a computing platform composed of 1 network and a varying number of processors. The density is fixed to $\delta_{tot} = 8$. It is possible to see that OPA in average performs better, for example, when the number of processors is equal to 9, the OPA algorithm accepts 70% of task sets, whilst the DM algorithm only accepts 30% of task sets.

Figure 6 (c) shows the number of accepted tasks sets over 100 experiments, where we vary the number of applications with a fixed total density $U_{tot} = 8$ to be scheduled in a computing platform of 10 processors and 1 network. It is possible to see that the OPA algorithm, in average performs better; in the range between 10 and 50 applications, OPA always accepts more task sets than DM. For example, for the case of 40 applications, the OPA algorithm accepts 69% of task sets, in contrast the number of accepted tasks sets obtained by the DM algorithm is 34%. Note that the number of accepted task sets increases with the number of generated applications. This behaviour can be explained by the fact that the average density of tasks and messages decreases, thereby meaning that more tasks can be scheduled on each processor in average.

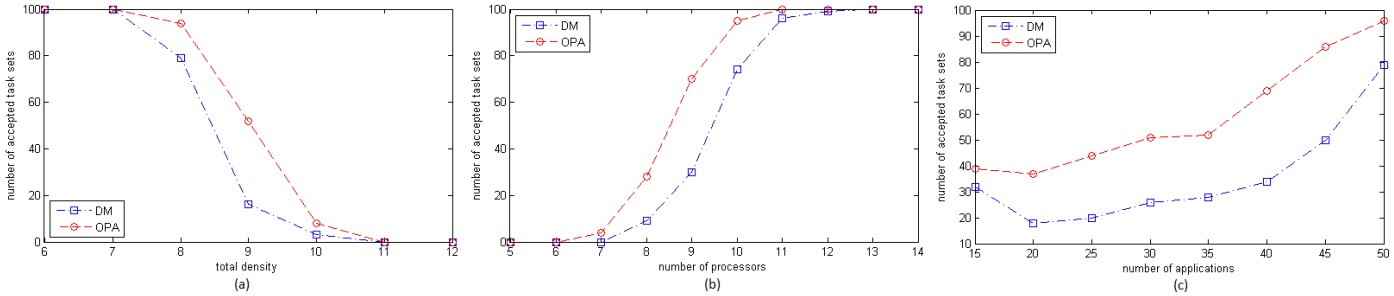

Figure 6. 100 experiments varying (a) the total density, (b) the number of processors, and (c) the number of applications in the system.

The effects presented in Figures 6 (a), (b), (c), can be explained because, when DM is used for assigning priorities, it fails more often than OPA due to its non-optimality. Therefore, such non-schedulable tasks need to be partition onto other processor in the distributed system, thus increasing the number of messages in the network, which leads to an increasing number of unschedulable systems.

VI. Conclusions and Future Work

This paper presented the DOPA heuristic for the simultaneous partitioning and priority assignment of tasks and messages (applications) onto the constituting elements of the distributed system by using the OPA algorithm known as Audsley's algorithm [2].

We proposed a method that imposes intermediate deadlines to tasks and messages thus permitting the use OPA for tasks with dependencies. Furthermore, our approach is easily extensible to more powerful task models such as multithreaded parallel real-time models, when compared with other works addressing sequential dependent tasks and messages in a distributed system. We demonstrated through simulations that OPA increases, in average, the number of schedulable tasks and messages in a distributed system, when compared to the DM algorithm, when using the same partition algorithm.

We are currently working on the extension of the DOPA heuristic for considering multithreaded parallel real-time models and the inclusion of more complex topologies of communication networks.

Acknowledgments

This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within projects FCOMP-01-0124-FEDER-015006 (VIPCORE) and FCOMP-01-0124-FEDER-037281 (CISTER); by FCT and EU ARTEMIS JU, within project ARTEMIS/0003/2012, JU grant nr. 333053 (CONCERTO); by FCT and ESF (European Social Fund) through POPH (Portuguese Human Potential Operational Program), under PhD grant SFRH/BD/71562/2010.

References

[1] A. Burns, "Scheduling hard real-time systems: a review," *Software Engineering Journal,* vol. 6, no. 3, pp. 116-128, 1991.

[2] N. C. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," University of York, Department of Computer Science, 1991.

[3] J. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance evaluation,* vol. 2, no. 4, pp. 237-250, 1982.

[4] K. Tindell, A. Burns and A. J. Wellings, "Allocating hard real-time tasks: an NP-hard problem made easy," *Real-Time Systems,* vol. 4, no. 2, pp. 145-165, 1992.

[5] J. J. Gutiérrez-García and M. González-Harbour, "Optimized priority assignment for tasks and messages in distributed hard real-time systems," in *Proceedings of the Third IEEE Workshop on Parallel and Distributed Real-Time Systems*, 1995.

[6] M. Richard, P. Richard and F. Cottet, "Allocating and scheduling tasks in multiple fieldbus real-time systems," in *Proceedings of IEEE Conference on Emerging Technologies and Factory Automation (ETFA'03)*, 2003.

[7] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and Microprogramming,* vol. 40, no. 2-3, pp. 117 - 134 , 1994.

[8] W. Zheng, Q. Zhu, M. Di Natale and A. S. Vincentelli, "Definition of task allocation and priority assignment in hard real-time distributed systems," in *Proc. 28th IEEE International Real-Time Systems Symposium (RTSS'2007)*, 2007.

[9] A. Metzner and C. Herde., "Rtsat--an optimal and efficient approach to the task allocation problem in distributed architectures," in *Proc. of the 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, 2006.

[10] E. Azketa, J. P. Uribe , J. J. Gutiérrez, M. Marcos and L. Almeida, "Permutational genetic algorithm for the optimized mapping and scheduling of tasks and messages in distributed real-time systems," in *In Proceedings of the 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, TRUSTCOM*, 2011.

[11] J. C. Palencia and M. González-Harbour, "Schedulability analysis for tasks with static and dynamic offsets," in *Proceedings of The 19th IEEE Real-Time Systems Symposium (RTSS'98)*, 1998.

[12] J. C. Palencia and M. Gonzalez-Harbour, "Exploiting precedence relations in the schedulability analysis of distributed real-time systems," in *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS'99)*, 1999.

[13] J. C. Palencia, J. J. Gutiérrez and M. González-Harbour, "On the schedulability analysis for distributed hard real-time systems," in *Proc. of IEEE Ninth Euromicro Workshop on Real-Time Systems*, 1997.

[14] R. I. Davis, S. Kollmann, V. Pollex and F. Slomka, "Schedulability analysis for Controller Area Network (CAN) with FIFO queues priority queues and gateways," *Real-Time Systems,* vol. 49, no. 1, pp. 73-116, 2013.

[15] R. I. Davis and A. Burns , "Priority Assignment for Global Fixed Priority Pre-Emptive Scheduling in Multiprocessor Real-Time Systems," in *Proc. of the 30th IEEE Real-Time Systems Symposium (RTSS 2009)*, 2009.

[16] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal,* vol. 29, no. 5, pp. 390-395, 1986.

[17] P. Emberson, R. Stafford and R. I. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *In 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS'10)*, 2010.

[18] R. Stafford , "Random vectors with fixed sum," [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/9700, 2006.