On the Scheduling of Fork-Join Parallel/Distributed Real-Time Tasks

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

***Abstract*— Modern real-time embedded applications present high computation requirements which need to be realized within strict time constraints. The current trend towards parallel processing in the embedded domain allows providing higher processing power. However, in some embedded applications, the use of powerful enough multi-core processors, may not be possible due to energy, space or cost constraints. A solution for this problem is to extend the parallel execution of the applications, allowing them to distribute their workload among networked nodes, on peak situations, to remote neighbour nodes in the system. In this context, we present the *Partitioned-Distributed-Deadline Monotonic Scheduling* algorithm for *fork-join parallel/distributed fixed-priority tasks*. We study the problem of scheduling fork-join tasks that execute in a distributed system, where the inherent transmission delay of tasks must be considered and cannot be deemed negligible, as in the case of multicore systems. Our scheduling algorithm is shown to have a resource augmentation bound of 4, which implies that any task set that is feasible on m unit-speed processors and a single shared real-time network, can be scheduled by our algorithm on m processors and a single real-time network that are 4 times faster. We confirm through simulations our analytical results.**

Keywords*— *real-time; distributed systems; fork-join; parallel execution; resource augmentation bound.

I. INTRODUCTION

Modern real-time applications are becoming larger and more complex, thus demanding more and more computing resources. By using parallel computations, the time required for processing computational-intensive applications can be reduced, thereby gaining in flexibility. This is a known solution in areas that require high performance computing power, and real-time systems are not an exception. That is why the real-time community has been making a large effort to extend real-time tools and scheduling algorithms to multicores [1], and lately to further extend them considering the use of parallel task models [2, 3, 4, 5].

Although, most of the developed work for parallel real-time tasks (multithreaded parallel real-time tasks) has been thought with multi- and many-core systems in mind, it is also possible to provide parallel computing power by aggregating a set of single-core embedded devices connected through an interconnection network, cooperating for achieving a single goal. Furthermore, in some embedded applications, the use of powerful enough multi-core processors, is prohibited due to energy, space or cost constraints. An example of such type of applications is, for instance, the image processing for obstacle detection in cooperating robots, where the computation requirements of the detection algorithms are highly dependent on the robot's current velocity, surrounding environment and obstacles [6]. Thus, it is possible to comply with the requirements of such computational-intensive applications by allowing a single-core embedded device to distribute its workload to remote neighbour nodes connected through a local real-time network.

Commonly, parallel programming applications are based on the fork-join execution model. A fork-join real-time application is an application that starts by executing sequentially and then forks to be executed in parallel, when the parallel execution has completed, the results are aggregated by performing a join operation; this procedure can be repeated several times. Some of the most popular programming models implementing the fork-join structure are the OpenMP programming model [7], and the Message Passing Interface (MPI) model [8]. However, none of these programming models is able to provide any time guarantees, although some efforts on bridging that gap have been presented in [9, 10].

A fork-join real-time distributed application is composed of a set of fork-join parallel/distributed real-time tasks executed in a distributed system. When considering such tasks, the processing of tasks and messages must comply with their associated time constraints, commonly expressed by an *end-to-end deadline*. Therefore, when scheduling fork-join real-time tasks in a distributed system, it is necessary to consider the interaction between the threads and messages that compose such a task and their impact when being scheduled in a set of different computing devices (e.g. processors and networks).

Contribution. In that context, we present the Partitioned/Distributed-Deadline Monotonic Scheduling (P/D-DMS) algorithm for distributed fixed-priority fork-join real-time tasks. The P/D-DMS algorithm is shown to have a resource augmentation bound of 4, which implies that any task set that is feasible on m unit-speed processors and a single shared bus real-time network, can be scheduled by this algorithm on m processors and a single shared bus real-time network that are 4 times faster. We confirm our analytical results through simulations.

Structure of the paper. The remainder of the paper is structured as follows. Section II presents the related work;

Section III introduces the system model. Section IV presents the Distributed Stretch Transformation model for parallel/distributed real-time tasks. The resource augmentation bound for the Partitioned-Distributed-DMS algorithm is explained in Section V. The simulations that confirm our analytical results are provided in Section VI, and finally our conclusions are discussed in section VII.

II. Related Work

We briefly review the related work for fixed-priority fork-join tasks in distributed systems and multicore processors.

Related to distributed systems, Gutiérrez-García *et al.* [11] presented a schedulability analysis technique for distributed hard real-time systems in which responses of different events may synchronize with each other. This is a general method for computing the worst-case response time of different synchronization events. This method allows the study of complex synchronization structures, in which the fork-join structure is included. The technique is based on the existing Rate Monotonic Analysis (RMA) techniques.

Zheng *et al.* [12], studied the case of automotive applications. The approach is based on finding the priorities for tasks and messages, in a way that no end-to-end deadline is missed. They proposed to solve the problem of priority assignment of tasks and messages by modelling it as an optimization problem. In Zhu *et al.* [13] is presented a similar problem as in [12], but for a more detailed system model. The authors presented a sensibility analysis that is able to measure how much the execution time of tasks can be increased without missing its end-to-end deadlines. Their method is based on a combination of mixed integer linear programming for task allocation, which is optimized according to tasks' utilization and deadlines. Also as a second stage of their method, they apply a set of heuristic steps for priority assignment of tasks and messages.

In respect to multicore architectures, Lakshmanan *et al.* [2] introduced the *Task Stretch Transformation* (TST) model for parallel synchronous tasks which follow a fork-join structure. They proposed a parallel model that considers preemptive fixed-priority periodic tasks with implicit deadlines scheduled according to the Deadline Monotonic (DM) rule, which is able to achieve a resource augmentation bound of 3.42. Similarly, Fauberteau *et al.* [3] proposed the *Segment Stretch Transformation* (SST) model. The authors convert the parallel threads into sequential ones by creating a master thread, but with the difference that no thread is ever allowed to migrate between cores. They showed through simulations that the TST and SST algorithms obtain similar results, and that none of them dominates the other. Later, Qamhieh *et al.* [4] proved that SST has the same resource augmentation bound than TST - 3.42. More insights of the TST [2] and DST [3, 4] models are presented in Section IV-A.

Saifullah *et al.* [5] introduced a more general model in which the problem of scheduling synchronous periodic multithreaded parallel tasks with implicit deadlines is addressed. Two main extensions to the previous works were made. First, the limitation of having the same number of threads in all parallel segments within a task was lifted by allowing an arbitrary number of threads to be executed on each parallel segment. And second, they consider the analysis of DM and EDF scheduling. They provided a resource augmentation bound of 4 and 5 when global EDF and partitioned DM are used to schedule tasks, respectively.

Axer *et al.* [14], the presented a method for computing the response time of fixed-priority parallel tasks on multiprocessors, which considers the synchronization effects for fork-join tasks with arbitrary deadlines.

However, none of the previous works addressed the specific problem of scheduling fork-join parallel/distributed tasks. On one hand, the research in the area of distributed systems is focused on task partition and priority assignment to tasks and messages with specific constraints required by the system. On the other hand, existing techniques for scheduling fork-join tasks are designed for multicore systems, in which the transmission delays of parallel threads of a fork-join task can be considered negligible. In this paper, we intend to combine these two domains, and present a scheduling algorithm for parallel/distributed real-time tasks in which transmission delays are accounted. Furthermore, our solution has the advantage to have a low complexity and could therefore be used online to partition new tasks arriving in the system.

III. System Model

We consider a set of fixed-priority fork-join parallel/distributed real-time tasks with implicit deadlines that execute in a distributed computing platform. A distributed computing platform is composed of a set $\pi = \{\pi_1, \dots, \pi_m\}$ of m identical uni-core nodes where tasks are executed. The number of processors m is defined by the architecture.

The processors are interconnected with a single shared bus real-time network, denoted as ϖ. Messages are exchanged between tasks executing on different nodes using the network. Messages sent on the network are considered to be non-preemptive (i.e., the transmission of a message cannot be aborted or interrupted once initiated) and scheduled according to a fixed priority algorithm (i.e., the message with the highest priority is transmitted first). Figure 1 shows an example of such a distributed computing platform.

A. Fork-Join Parallel/Distributed Real-Time Tasks

We consider that a distributed real-time application is composed of a set $\tau = \{\tau_1, \dots, \tau_n\}$ of n fixed-priority fork-join parallel/distributed real-time tasks (*P/D tasks*). P/D tasks have the particularity of making use of distributed resources, interconnected by a shared bus real-time network. Such a configuration imposes an inevitable transmission delay whenever parts of a task are distributed among a set of processors in the distributed system. This delay is due to the necessity for data distribution, but might also include the transmission of the code to be executed and/or control messages.

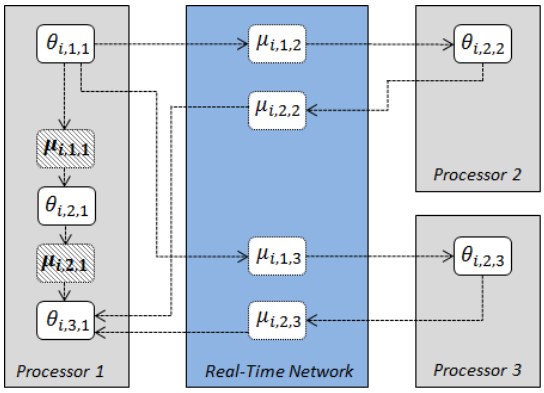

Figure 1. Distributed computing platform.

A P/D task τ_i is activated periodically every T_i time units and is characterised by an implicit end-to-end deadline D_i, which is the longest elapsed time a task is permitted to take from the time instant at which it is activated until it completes its execution. Figure 2 shows an example of a P/D task τ_i.

In the OpenMP programming model, a single thread starts the execution of a parallel program. This thread is the *master thread* of the program. When the master thread encounters a `#parallel` pragma (indicating the starting point of the parallel execution) it creates a team of threads to execute the instructions enclosed within the `#parallel` pragma in a parallel manner. A P/D task follows the fork-join structure of the OpenMP programming model [7].

A P/D task starts by a master thread executing sequentially; and then forks to be executed in parallel on *remote processors*. When the parallel execution has completed on each of the remote processors, the results are aggregated by performing a join operation and the execution of the sequential thread is resumed within the master thread. This procedure can be repeated several times.

We call Distributed-Fork (D-Fork) and Distributed-Join (D-Join), the operations that correspond to the classical fork and join operations usually performed on multicore processors. The main difference is that the execution of the threads resulting from the D-Fork operation are executed on remote processors within the distributed system, implying that the communication between threads is realized through messages sent through a real-time network.

Formally, a P/D task τ_i ($i \in \{1, \dots, n\}$) is composed of a sequence of sequential and parallel/distributed (P/D) segments $\sigma_{i,j}$ with $j \in \{1, \dots, n_i\}$. Where, n_i represents the number of segments composing τ_i, n_i is assumed to be an *odd* integer, as a P/D task should always start and finish with a sequential segment. Therefore, odd segments $\sigma_{i,2j+1}$ identify sequential segments and *even* segments $\sigma_{i,2j}$ identify P/D segments. Each segment $\sigma_{i,j}$ is composed of a set θ of threads $\theta_{i,j,k}$ with $k \in \{1, \dots, n_{i,j}\}$, where $n_{i,j} = 1$ for sequential segments and $n_{i,j} = m_i \leq m$ threads for P/D segments. m_i is the number of P/D threads in each P/D segment, and it is considered to be the same for all P/D segments within a P/D task τ_i.

All sequential segments within a P/D task τ_i must execute within the same processor. This means that the processor that performs a D-Fork operation (*invoker processor*) is in charge of aggregating the result by performing a D-Join operation. Threads within a P/D segment are possibly executed on remote processors. Consequently, for each thread $\theta_{i,2j,k}$ belonging to a P/D segment (P/D thread), two P/D messages $\mu_{i,2j-1,k}$ and $\mu_{i,2j,k}$ are considered for realizing the communication between the invoker and remote processors. This is, P/D threads and messages that belong to a P/D segment and execute on a remote processor, have a precedence relation: $\mu_{i,2j-1,k} \longrightarrow \theta_{i,2j,k} \longrightarrow \mu_{i,2j,k}$. For each sequential and P/D segment, there exists a synchronisation point at the end of each segment, indicating that no thread that belongs to the segment after the synchronisation point can start executing before all threads of the current segment have completed execution. P/D threads are preemptive, but messages' packets are non-preemptive, although large messages can be divided in several non-preemptive packets.

Also, each sequential thread $\theta_{i,2j+1,1}$ has a Worst-Case Execution Time (WCET) of $C_{i,2j+1,1}$. A P/D thread $\theta_{i,2j,k}$ has a WCET of $P_{i,2j,k}$, and each message $\mu_{i,2j,k}$ has a Worst-Case Message Length (WCML) $\mathcal{M}_{i,j,k}$. It is assumed that for a task τ_i, every P/D thread $\theta_{i,2j,k}$ and their respective messages $\mu_{i,2j,k}$ within a P/D segment $\sigma_{i,2j}$, have identical WCETs $P_{i,2j,k}$ and identical WCMLs $\mathcal{M}_{i,j,k}$, respectively. However, the WCET and the WCML of P/D threads and their messages can vary between different P/D segments. Also, P/D threads and messages, share the same period T_i.

To summarise, it is possible to describe a P/D task as:

$$\tau_i = ((C_{i,1,}, \mathcal{M}_{i,1}, P_{i,2}, \mathcal{M}_{i,2}, C_{i,3}, \dots, \mathcal{M}_{i,n_i-1}, C_{i,n_i}), m_i, T_i),$$

where:

- n_i is the total number of segments of a task τ_i,
- $C_{i,j}$ is the WCET of each sequential segment $\sigma_{i,2j+1}$ (containing a single thread $\theta_{i,2j+1,1}$),
- $\mathcal{M}_{i,j}$ is the WCML of a single messages (all k P/D messages on the same P/D segment have exactly the same WCML),
- $P_{i,2j}$ is the WCET of a single P/D thread within a segment $\sigma_{i,2j}$ (all k P/D threads on the same P/D segment have exactly the same WCET),
- m_i is the number of P/D threads (two messages are created for each P/D thread within a P/D segment $\sigma_{i,2j}$) in each P/D segment. It is considered that $m_i \leq m$, where m is the number of available processors in the distributed system,
- T_i is the period of a task, which is equal to its deadline ($D_i = T_i$).

B. Preliminaries

For notational convenience we introduce some definitions that simplify the explanation and analysis of the proposed algorithms.

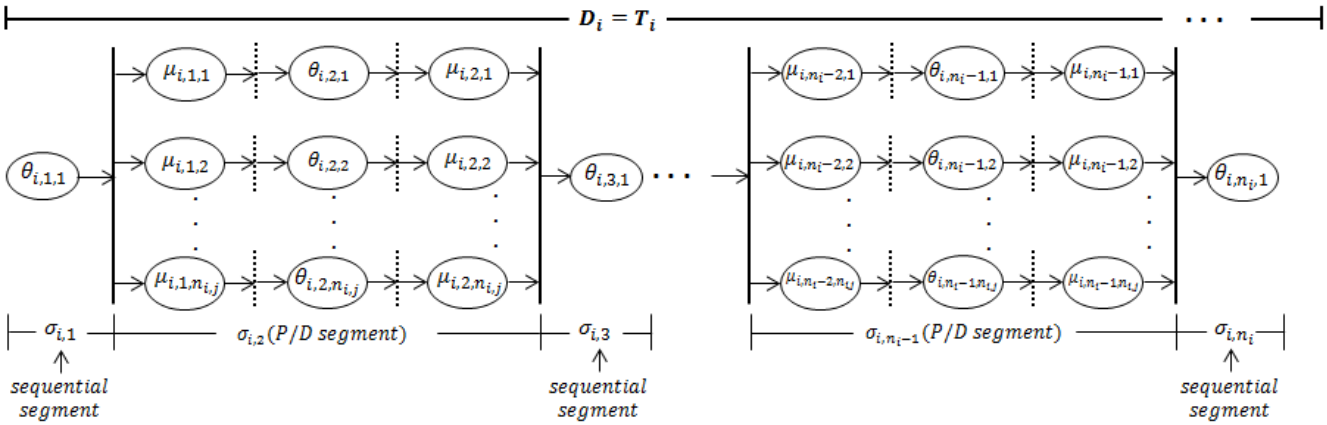

Figure 2. The fork-join parallel/distributed periodic real-time tasks (*P/D task*) model.

Definition 1. (Master thread). *The master thread of a P/D task τ_i is the collection of all threads $\theta_{i,j,1}$ belonging to all segments $\sigma_{i,j}$. A master thread can be represented as:*

$$\tau_i^{master} = \{\theta_{i,1,1}, \theta_{i,2,1}, \theta_{i,3,1}, \dots, \theta_{i,n_i-1,1}, \theta_{i,n_i,1}\}$$

Definition 2. (Parallel execution length). *The parallel execution length is the sum of the WCET of all P/D threads within the master thread:*

$$P_i = \sum_{j=1}^{\frac{n_i-1}{2}} P_{i,2j,1} \tag{1}$$

Definition 3. (Minimum execution length). *The minimum execution length η_i represents the minimum execution time a P/D task τ_i needs to execute when all P/D threads are executed in parallel. This is equal to the sum of the WCET of all the threads described in the master thread:*

$$\eta_i = \left(\sum_{j=0}^{\frac{n_i-1}{2}} C_{i,2j+1}\right) + P_i \tag{2}$$

Definition 4. (Maximum execution length). *The maximum execution length C_i, represent the maximum execution time a P/D task τ_i needs to execute when all P/D threads are executed sequentially on the invoker processor. This is equal to the sum of WCET of all threads in a task τ_i:*

$$C_i = \left(\sum_{j=0}^{\frac{n_i-1}{2}} C_{i,2j+1}\right) + P_i \times m_i \tag{3}$$

Please note, that the messages $\mu_{i,j,k}$ are not considered in Eq. (2) and (3), since all inter-process communications are internal to the invoker processor. The synchronization cost between the sequential and P/D threads can therefore be considered negligible. Figure 1, shows an example in which some messages ($\mu_{i,1,1}$ and $\mu_{i,2,1}$) are realizing a transmission within the same local processor, but its cost is negligible.

Definition 5. (Slack time). *The positive slack time L_i is the temporal difference between the task's deadline D_i and the minimum execution length η_i:*

$$L_i = D_i - \eta_i \tag{4}$$

If the slack L_i is a negative number, it means that the minimum execution length η_i is larger than its deadline ($T_i = D_i$). Therefore, such a task is not schedulable on any number of processors with a speed of 1.

Definition 6. (Capacity). *The capacity f_i is defined as the capacity of the master thread of a task τ_i to execute extra P/D threads from all P/D segments without missing its deadline:*

$$f_i = \frac{L_i}{P_i} \tag{5}$$

IV. Distributed Stretch Transformation

Our task transformation model is called *Distributed Stretch Transformation* (DST), and it has been inspired by the *Task Stretch Transformation* [2] (TST) model and the *Segment Stretch Transformation* [3, 4] (SST) model. The DST model is designed specifically for distributed systems where real-time tasks and messages need to be processed and transmitted by processors and a real-time network, respectively. Therefore, the main difference from DST, when compared with TST and SST, is that the two previous transformation algorithms were conceived for multicore processors, thus not considering the transmission delays inherent to the synchronization between threads executing on different processors, as in the case of distributed systems.

The TST and SST transformations consider that tasks are scheduled by a partitioned preemptive fixed-priority algorithm, executed in a multicore processor. In our model, we consider tasks to be scheduled with the preemptive fixed priority algorithm deadline monotonic (DM) on each processor. On the other hand, messages to be transmitted within the real-time network are scheduled with a non-preemptive version of DM algorithm. This is due to the fact that the transmission of a message cannot be interrupted once initiated.

A. The Task Stretch Transformation and Segment Stretch Transfomation Models

In this subsection, we study the TST and the SST transformation models, with the intention of showing the similarities and main differences with our DST model.

In the TST model [2], Lakshmanan et al. show that *"FJ task sets on multiprocessor systems can have schedulable utilization bounds slightly greater than and arbitrarily close to uniprocessor schedulable utilization bounds"*, thus, it is desirable to avoid fork-join structures as much as possible. The main objective of the TST model is to convert the master thread into a fully stretched string in which the execution length of the master thread becomes equal to its period T_i. The transformation is done by inserting (or coalescing) threads (or part of them) into the master thread while paying attention to respect their precedence constraints. Thus, a subset of parallel threads executes with the master thread while the rest of them are partitioned among the cores using the partitioning heuristic Fisher-Baruah-Baker First-Fit-Decreasing (FBB-FFD) [15]. The authors showed that their scheduling algorithm has a resource augmentation bound of 3.42.

The main disadvantage of the TST is that it forces to stretch a master thread completely. In some cases, it may not be possible to fit complete threads within the master thread. This provokes a migration of the remaining part of such a thread for being executed in another processor. This migration can lead to extra considerations when implemented in a real platform. For this reason, the authors of [3] proposed the SST model, which also tries to convert the parallel threads into sequential ones by creating a master thread, but with the difference that the coalescing operation is performed only when parallel threads can be fully inserted within the master thread. Thus, creating a master thread that can be fully stretched (with a WCET of the master thread equal to its period) or partially stretched (the WCET of the master thread is smaller or equal to its period). In a similar manner than in [2], the remaining parallel threads are scheduled with the partitioned scheduling algorithm FBB-FFD [15]. Later, the same authors [4] proved that SST has the same resource augmentation bound of 3.42 than TST, although, it cannot be claimed that one of both algorithms dominates the other [3].

B. The Distributed Stretch Transform Model

Our work is inspired by the SST approach. Since *"FJ task sets on multiprocessor systems can have schedulable utilization bounds slightly greater than and arbitrarily close to uniprocessor schedulable utilization bounds"*, we opt for the formation of a stretched master thread $\tau_i^{stretched}$ for each P/D task τ_i. However, we need to address some specific constraints that are related to distributed systems. In that case, when performing a D-Fork operation, it implies that some messages will be transmitted within the network that may affect the execution length of the P/D tasks.

Let us illustrate the DST transformation with an example. Consider two tasks: $\tau_1 = ((1,1,2,1,1),3,8)$, and $\tau_2 = ((1,1,3,1,1),3,10)$ to be scheduled on 3 processors. Figure 3(a), shows the execution of a task to be scheduled under global DM scheduling. It is possible to see that task τ_2 having the lowest priority misses its deadline at time 10. This is due to the suffered interference provoked by threads of task τ_1 that have higher priority. Also, notice the presence of a high source of interference in the network, for example, the P/D thread $\theta_{2,2,3}$ with WCET $P_{2,2,3}$, is ready for execution at time 1, but due to the network interference it is only released for execution in processor 3 at time 7, therefore drastically increasing its response time.

Now let us consider the DST transformation explained below and illustrated in Figure 3(b). By calculating the maximum execution length of tasks τ_1 and τ_2 (see Definition 4), we obtain $C_1 = 8$ and $C_2 = 11$. Then, by looking at Figure 3(b) it is possible to observe two cases:

1. $C_i \leq T_i$. This is the case of τ_1 in our example; whenever such a case appears for a task τ_i, the task τ_i is fully stretched into a master thread and handled as a sequential task with execution time equal to C_i, a task period of T_i, and an implicit deadline equal to D_i. That is, all threads of the tasks are executed sequentially on a unique processor.
2. $C_i > T_i$. This is this case of τ_2 in our example; for such tasks, the DST transformation inserts (coalesces) as many P/D threads of τ_i into the master thread as possible. To do so, it is needed to calculate the available slack and capacity of task τ_i as indicated in Eq. (4) and (5). For τ_2, it gives $L_2 = 10 - 5 = 5$ and, $f_2 = 5/3$. Thus, the number of P/D threads that each P/D segment can fully insert into the master thread without causing τ_i to miss its deadline is given by:

$$i_{i,2j} = \lfloor f_i \rfloor \tag{6}$$

For example, in the case of τ_2, $i_{2,2} = \lfloor f_2 \rfloor = 1$. It can indeed be seen on Figure 3(b) that τ_2 executes two P/D threads per P/D segment on the invoker processor rather than only one when considering the non-stretched master thread.

In the DST only P/D threads that fit completely (since $i_{i,2j} = \lfloor f_i \rfloor$) can be inserted into the master thread. A master thread is assigned to be executed in its own processor and the remaining subset of P/D threads, have to be executed on other nodes in the system. The dispatching of the remaining P/D threads to the processors is performed according to the FBB-FFD algorithm [15].

The number $q_{i,2j}$ of the remaining P/D threads that have not been coalesced into the master thread is given by:

$$q_{i,2j} = m_i - i_{i,2j} \tag{7}$$

The slack f_i of task τ_i is equally distributed between all the P/D segments of a P/D task τ_i. This distribution can be considered as the available scheduling length for the execution of threads and transmission of messages in each P/D segment on a remote processor.

Thus, the maximum scheduling length for the subset of P/D threads and their respective messages is determined by defining a set of P/D intermediate deadlines $d_{i,2j}$:

$$d_{i,2j} = (f_i + 1) \times P_{i,2j} \;\; \forall\, 1 \leq j \leq \frac{n_i - 1}{2} \tag{8}$$

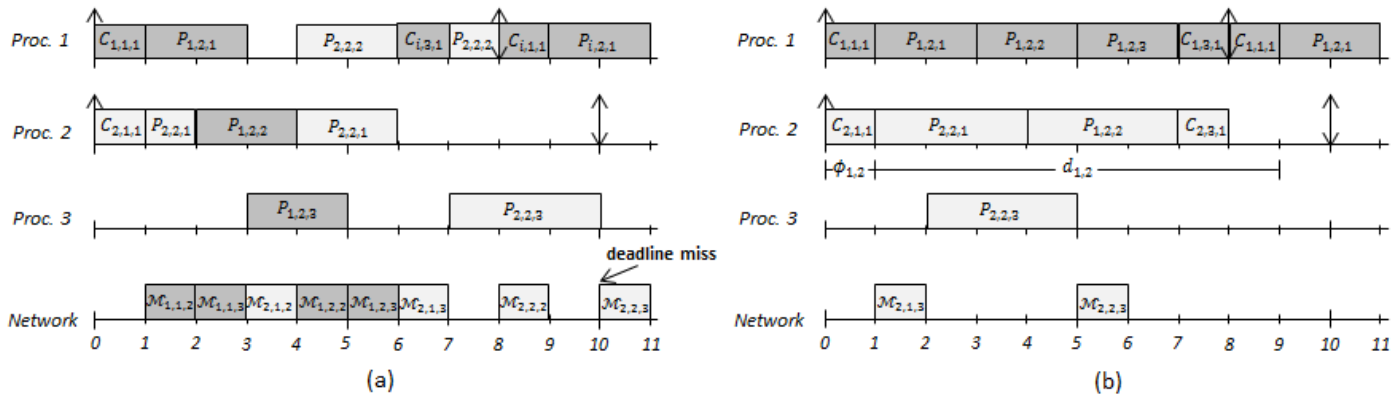

Figure 3. A P/D tasks: (a) scheduled with global scheduling, (b) scheduled after the DST transformation.

In the case of task τ_2, $d_{2,2} = 3(5/3+1) = 8$. Also, each P/D segment $\sigma_{i,2j}$ has a static offset $\phi_{i,2j}$ defined as:

$$\phi_{i,2j} = \sum_{j=0}^{\frac{n_i-1}{2}} C_{i,2j+1,1} + \sum_{j=1}^{\frac{n_i-1}{2}} d_{i,jk} \quad (9)$$

Thus, at the end of the DST transformation, a P/D task τ_i will be composed of a single stretched master thread $\tau_i^{stretched}$ and a set of constrained deadline P/D threads $\{\tau_i^{cd}\}$ (and their respective constrained deadline messages $\{\mu_i^{cd}\}$) per each P/D segment $\sigma_{i,2j}$.

The P/D segments' offsets $\phi_{i,2j}$ and the P/D segments' deadlines $d_{i,2j}$, define the scheduling window, in which the remaining $q_{i,2j}$ P/D threads and its corresponding messages have to complete their execution (and transmission, respectively) in order for a task τ_i to respect its deadline. That is, the following inequality must be respected:

$$r_{\mu_{i,2j-1,k}} + r_{\theta_{i,2j,k}} + r_{\mu_{i,2j,k}} \leq d_{i,2j} \forall\, \theta_{i,2j,k} \notin master\ thread \quad (10)$$

where, $r_{\mu_{i,2j-1,k}}$, $r_{\mu_{i,2j,k}}$ and $r_{\theta_{i,2j,k}}$ are the Worst Case Response Time (WCRT) of messages $\mu_{i,2j-1,k}$ and $\mu_{i,2j,k}$, and the thread $\theta_{i,2j,k}$, respectively.

C. End-to-end delay computation in distributed systems

In this section we summarize some results for the calculation of the response time for the execution and transmission of threads and messages respectively.

From [16] we know that for periodic fixed priority preemptive tasks, the following recursive equation can be used to calculate the response time of a threads $\theta_{i,j,k}$:

$$r_{\theta_{i,j,k}}^{n+1} = C_{i,j,k} + \sum_{\theta_{i,j,l} \in hp(\theta_{i,j,k})} \left\lceil \frac{r_{\theta_{i,j,k}}^{n}}{T_{i,j,l}} \right\rceil C_{i,j,l}, \quad (11)$$

where $r_{\theta_{i,j,k}}$ is the worst case execution time of a thread $\theta_{i,j,k}$ and $hp(\theta_{i,j,k})$ is the set of all threads $\theta_{i,j,l}$ with higher priority than $\theta_{i,j,k}$ that execute on the same processor π_i. The recursion ends when $r_{\theta_{i,j,k}}^{n+1} = r_{\theta_{i,j,k}}^{n} = r_{\theta_{i,j,k}}$ and can be solved by successive iterations starting from $r_{\theta_{i,j,k}}^{1} = \theta_{i,j,k}$. The series is non-decreasing, and therefore converges if $\sum_{\theta_{i,j,l} \in hp(\theta_{i,j,k}) \cup \theta_{i,j,k}} \frac{C_{i,j,l}}{T_{i,j,l}} \leq 1$. If the condition of convergence is not respected threads $\theta_{i,j,k}$ are not schedulable.

For the case of messages $\mu_{i,j,k}$, the calculation of the WCRT needs to consider the non-preemptability of messages on the network. Thus, for periodic fixed-priority non-preemptive messages, the following recursive equation can be used to calculate the worst-case response time [17]:

$$r_{\mu_{i,j,k}}^{n+1} = \mathcal{M}_{i,j,k} + \sum_{\mu_{i,j,l} \in hp(\mu_{i,j,k})} \left\lceil \frac{r_{\mu_{i,j,k}}^{n}}{T_{\mu_{i,j,l}}} \right\rceil \mathcal{M}_{i,j,l} + \max_{\mu_{i,j,l} \in lp(\mu_{i,j,k})} \{\mathcal{M}_{i,j,l}\}, \quad (12)$$

where, the third term on the right hand side of Eq. (12), accounts for the maximum possible suffered interference of a higher priority message $\mu_{i,j,k}$, caused by lower priority message $\mu_{i,j,l}$, contained in the set of lower priority messages $lp(\mu_{i,j,k})$.

V. THE P/D-DMS ALGORITHM

The P/D-DMS algorithm is the dispatching algorithm for partitioning the set τ of tasks τ_i onto the elements of the distributed system. The P/D-DMS algorithm realizes the dispatching by: (i) applying the DST to each P/D taks τ_i in τ. Two possible cases can appear (Section IV-B): (1) $C_i \leq T_i$; the task is fully stretched in a single sequential thread and added to a list $\mathcal{L}$, or (2) the task τ_i is converted into a master thread τ_i^{master} and a subset of sequential P/D threads $\{\tau_i^{cd}\}$ with their respective messages $\{\mu_i^{cd}\}$. The master thread τ_i^{master} is allocated to its own processor and the subset of sequential P/D threads is added to the list $\mathcal{L}$, and (ii) the set of threads in $\mathcal{L}$, are partitioned onto processors according to the FBB-FFD algorithm [15]. Messages $\{\mu_i^{cd}\}$ are assigned to the single real-time network.

In the following subsection, we analyse and prove the demand bound function of a P/D task τ_i and provide the resource augmentation bound for the P/D-DMS algorithm.

A. Demand Bound Function

Definition 7. (Demand Bound Function (DBF) [18]). *The DBF is defined as the largest cumulative execution requirement of all jobs that can be generated by τ_i to have both their arrival times and their deadlines within a contiguous interval of length t.*

For a sequential task τ_i with a total execution time of C_i, period T_i, and a deadline $D_i \leq T_i$, the DBF function is given by:

$$DBF(\tau_i, t) = max(0, (\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1)C_i) \quad (13)$$

Theorem 1. *The DBF function of a stretched task $\tau_i^{stretched}$ that has been transformed by the DST algorithm is bounded from above by:*

$$DBF(\tau_i^{stretched}, t) \leq \max_j \{ \frac{C_i}{T_i - \eta_i - \frac{\left(r_{\mu_{i,2j-1,k}} + r_{\mu_{i,2j,k}}\right) \times P_i}{P_{i,2j}}} \} t \quad (14)$$

Proof: we generalize the concept of DBF for the case of P/D tasks τ_i composed of a master thread τ_i^{master} and a sequence of sequential P/D threads $\{\tau_i^{cd}\}$ and their respective messages $\{\mu_i^{cd}\}$. We consider the two only possible cases when applying the DST algorithm to a P/D task τ_i (see Section IV-B):

1. **Case $C_i \leq T_i$.** In that case, a P/D task τ_i is fully stretched after applying the DST into a single sequential thread with a total execution time of $C_i^{master} \leq C_i$, period T_i, and a deadline $D_i^{master} = T_i$, therefore, the DBF function (Definition 7) can be used without any modifications:

$$DBF(\tau_i^{stretched}, t) = DBF(\tau_i^{master}, t)$$

$$DBF(\tau_i^{stretched}, t) = \max\{0, (\left\lfloor \frac{t - D_i^{master}}{T_i} \right\rfloor + 1)C_i^{master}\}$$

$$DBF(\tau_i^{stretched}, t) = \max\{0, (\left\lfloor \frac{t}{T_i} \right\rfloor)C_i^{master}\} \leq \frac{C_i}{T_i} t$$

$$\leq \frac{C_i}{T_i - \eta_i} t \leq \max_j \{ \frac{C_i}{T_i - \eta_i - \frac{(r_{\mu_{i,2j-1,k}} + r_{\mu_{i,2j,k}}) \times P_i}{P_{i,2j}}} \} t \quad (15)$$

where $0 \leq \eta_i \leq T_i$.

2. **$C_i > T_i$.** In the second case, after applying the DST, a P/D task τ_i has been transformed into a master thread τ_i^{master}, and a set $\{\tau_i^{cd}\}$ of constrained deadline P/D threads associated to their respective constrained deadline messages $\{\mu_i^{cd}\}$. That is,

$$\tau_i^{stretched} = \tau_i^{master} + \{\tau_i^{cd}\}$$

Thus, the DBF function can be computed as follows:

$$DBF(\tau_i^{stretched}, t) = DBF(\tau_i^{master}, t) + DBF(\{\tau_i^{cd}\}, t) \quad (16)$$

Since the master thread has been stretched, we have:

$$\eta_i + P_i \lfloor f_i \rfloor \leq C_i^{master} \leq T_i$$

$$C_i^{master} \leq T_i \Rightarrow \frac{C_i^{master}}{T_i} < 1$$

$$\Longrightarrow DBF(\tau_i^{master}, t) = \max(0, (\left\lfloor \frac{t}{T_i} \right\rfloor)C_i^{master})$$

$$\leq \frac{C_i^{master}}{T_i} t \leq t \quad (17)$$

The set $\{\tau_i^{cd}\}$ of constrained deadline P/D threads and their respective messages $\{\mu_i^{cd}\}$ consist of the P/D threads and P/D messages of all P/D segments of a task τ_i. Since P/D segments within a P/D tasks have an offset, only one P/D segment can be activated at time t and the *maximum number of P/D threads* in each P/D region is equal to $(q_{i,2j} - 1)$ where $q_{i,2j} = m_i - \lfloor f_i \rfloor$.

Therefore, the previous property guarantees that the DBF of the subset of P/D threads $\{\tau_i^{cd}\}$ over any interval of length t, does not exceed $\delta_i^{max}(q_{i,2j} - 1)t$:

$$DBF(\{\tau_i^{cd}\}, t) \leq \delta_i^{max}(q_{i,2j} - 1)t \quad (18)$$

The density of a constrained deadline tasks is given by:

$$\delta_i = \frac{C_i}{D_i}$$

The DST transformation fills the available slack L_i with $\lfloor f_i \rfloor$ P/D threads per P/D segment (remember that only complete P/D threads are inserted within the master thread, since $\lfloor f_i \rfloor$ is an integer number). In each P/D segment within τ_i, all P/D threads have the same WCET $P_{i,2j}$, and a deadline $d_{i,2j} = P_{i,2j} \times (f_i + 1)$. Due to the fact that the P/D thread is executed on a remote processor in the system, two messages per P/D thread ($\mu_{i,2j-1,k}$ and $\mu_{i,2j,k}$) are sent through the real-time network. Thus, in the worst-case the time for a P/D thread to execute is reduced to: $P_{i,2j} \times (f_i + 1) - r_{\mu_{i,2j-1,k}} - r_{\mu_{i,2j,k}}$ (see Eq.(8 and 10)).

Therefore, the maximum density of the P/D threads $\{\tau_i^{cd}\}$ can be calculated as follows:

$$\delta_i^{max} = \max_j \{ \frac{P_{i,2j}}{P_{i,2j} \times (f_i + 1) - r_{\mu_{i,2j-1,k}} - r_{\mu_{i,2j,k}}} \}$$

$$= \max_j \{ \frac{1}{(f_i + 1) - \frac{r_{\mu_{i,2j-1,k}} + r_{\mu_{i,2j,k}}}{P_{i,2j}}} \} \quad (19)$$

By substituting Eq. (19) in Eq. (18), the DBF of the P/D threads $\{\tau_i^{cd}\}$ can be calculated as:

$$DBF(\{\tau_i^{cd}\}, t) \leq \max_j \{ \frac{1}{(f_i + 1) - \frac{r_{\mu_{i,2j-1,k}} + r_{\mu_{i,2j,k}}}{P_{i,2j}}} \} (q_{i,2j} - 1)t$$

and since $q_{i,2j} = m_i - \lfloor f_i \rfloor$, we get

$$DBF(\{\tau_i^{cd}\}, t) \leq \max_j \{ \frac{m_i - \lfloor f_i \rfloor - 1}{(f_i + 1) - \frac{(r_{\mu_{i,2j-1,k}} + r_{\mu_{i,2j,k}})}{P_{i,2j}}} \} t$$

$$\leq \max_j\{\frac{m_i - \lfloor f_i \rfloor - 1}{f_i - \frac{(r_{\mu_{i,2j-1,k}} + r_{\mu_{i,2j,k}})}{P_{i,2j}}}\}t \quad (20)$$

By substituting inequality (17) and inequality (20) in Eq. (16), we can compute the DBF of $\tau_i^{stretched}$ as:

$$DBF(\tau_i^{stretched}, t) \leq t + \max_j\{\frac{m_i - \lfloor f_i \rfloor - 1}{f_i - \frac{(r_{\mu_{i,2j-1,k}} + r_{\mu_{i,2j,k}})}{P_{i,2j}}}\}t$$

$$\leq \max_j\{1 + \frac{m_i - \lfloor f_i \rfloor - 1}{f_i - \frac{(r_{\mu_{i,2j-1,k}} + r_{\mu_{i,2j,k}})P_i}{P_{i,2j}}}\}t$$

$$\leq \max_j\{\frac{f_i - \frac{(r_{\mu_{i,2j-1,k}} + r_{\mu_{i,2j,k}})}{P_{i,2j}} + m_i - \lfloor f_i \rfloor - 1}{f_i - \frac{(r_{\mu_{i,2j-1,k}} + r_{\mu_{i,2j,k}})}{P_{i,2j}}}\}t$$

$$\leq \max_j\{\frac{m_i - \frac{(r_{\mu_{i,2j-1,k}} + r_{\mu_{i,2j,k}})}{P_{i,2j}}}{f_i - \frac{(r_{\mu_{i,2j-1,k}} + r_{\mu_{i,2j,k}})}{P_{i,2j}}}\}t$$

because $f_i = \frac{T_i - \eta_i}{P_i}$ and $m_i \times P_i < C_i$ (from Eq.(3)), it results that:

$$DBF(\tau_i^{stretched}, t) \leq \max_j\{\frac{m_i \times P_i - \frac{(r_{\mu_{i,2j-1,k}} + r_{\mu_{i,2j,k}})P_i}{P_{i,2j}}}{T_i - \eta_i - \frac{(r_{\mu_{i,2j-1,k}} + r_{\mu_{i,2j,k}})P_i}{P_{i,2j}}}\}t$$

$$\leq \max_j\{\frac{C_i}{T_i - \eta_i - \frac{(r_{\mu_{i,2j-1,k}} + r_{\mu_{i,2j,k}})P_i}{P_{i,2j}}}\}t \quad (21)$$

Then, in the two possible cases, the DBF of a task $\tau_i^{stretched}$ resulting of the application the DST transformation is bounded by the same value (Eq. (15) and Eq. (21)). ■

B. Resource Augmentation Bound

Now we provide a resource augmentation bound for the Distributed-DMS partition algorithm. We will use the results of Theorem 2 from Fisher *et al.* [15].

Theorem 2 (from [15]**).** *Any constrained sporadic task system τ is successfully schedulable by FBB-FFD on m unit-capacity processors if:*

$$m \geq \frac{\delta_{sum} + u_{sum} - \delta_{max}}{1 - \delta_{max}} \quad (22)$$

where

$$\delta_{sum} = \max_{t>0}\left(\frac{\sum_{i=1}^{n} DBF(\tau_i^{stretched}, t)}{t}\right) \quad (23)$$

Using equations (22) and (23), we now provide a resource augmentation bound for the Distributed-DMS partition algorithm.

Theorem 3. *If any set τ of P/D tasks τ_i is feasible on m unit-speed processors (and messages are feasible on a single unit-speed real-time network), then the Distributed-DMS partition algorithm is guaranteed to successfully schedule this task set on m processors and one real-time network that are 4 times faster.*

Proof: The set τ of P/D tasks is feasible on m unit-speed processors:

$$u_{sum} \stackrel{\text{def}}{=} \sum_{i=1}^{n} \frac{C_i}{T_i} \leq m, \quad (24)$$

and because the minimum response time of a thread is its execution time, Equations (8) and (9) imply that the task set τ is feasible if and only if:

$$r_{\mu_{i,2j-1,k}} + P_{i,2j} + r_{\mu_{i,2j,k}} \leq (f_i + 1) \times P_{i,2j}$$
$$\Leftrightarrow \quad r_{\mu_{i,2j-1,k}} + r_{\mu_{i,2j,k}} \leq f_i \times P_{i,2j} \quad (25)$$

Let us consider the *minimum execution length* η_i of any task τ_i. It must respect that:

$$\forall\, 1 \leq i \leq n \;\; \eta_i \leq T_i \quad (26)$$

Otherwise, τ_i would be unschedulable on a unit-speed processor. On a processor that is v times faster, the *minimum execution length* η_i^v is given by:

$$\forall\, 1 \leq i \leq n \;\; \eta_i^v = \frac{\eta_i}{v} \leq \frac{T_i}{v} \quad (27)$$

For each task τ_i, it was proven by Theorem 1 that:

$$DBF(\tau_i^{stretched}, t) \leq \max_j\{\frac{C_i}{T_i - \eta_i - \frac{(r_{\mu_{i,2j-1,k}} + r_{\mu_{i,2j,k}})P_i}{P_{i,2j}}}\}t$$

Using the above inequality together with Eq. (22) we have:

$$\delta_{sum}^v \leq \sum_{i=1}^{n} \max_j\{\frac{C_i^v}{T_i - \eta_i^v - \frac{(r_{\mu_{i,2j-1,k}}^v + r_{\mu_{i,2j,k}}^v)P_i^v}{P_{i,2j}^v}}\}$$

using inequality (27):

$$\delta_{sum}^v \leq \sum_{i=1}^{n} \max_j\{\frac{C_i^v}{T_i\left(1 - \frac{1}{v}\right) - \frac{(r_{\mu_{i,2j-1,k}}^v + r_{\mu_{i,2j,k}}^v)P_i^v}{P_{i,2j}^v}}\}$$

$$\leq \frac{1}{v}\sum_{i=1}^{n} \max_j\{\frac{C_i}{T_i\left(1 - \frac{1}{v}\right) - \frac{1}{v}\frac{(r_{\mu_{i,2j-1,k}} + r_{\mu_{i,2j-1,k}})P_i}{P_{i,2j}}}\}$$

From inequality (25) and then (5):

$$\delta_{sum}^v \leq \frac{1}{v}\sum_{i=1}^{n} \frac{C_i}{T_i\left(1 - \frac{1}{v}\right) - \frac{1}{v}f_i P_i}$$

$$\leq \frac{1}{v}\sum_{i=1}^{n} \frac{C_i}{T_i\left(1 - \frac{1}{v}\right) - \frac{T_i}{v}}$$

$$\leq \frac{1}{v}\sum_{i=1}^{n} \frac{C_i}{T_i\left(1 - \frac{1}{v}\right) - \frac{T_i}{v}} \leq \frac{1}{v-2}\sum_{i=1}^{n} \frac{C_i}{T_i}$$

$$\Leftrightarrow \delta_{sum}^{v} \leq \frac{1}{v-2} u_{sum}$$

Also on v speed processors, $u_{sum}^{v} = \frac{u_{sum}}{v}$ and $\delta_{max}^{v} = \frac{\delta_{max}}{v}$. Using Eq. (22), the task set τ is schedulable on m processors of speed v if:

$$m \geq \frac{\delta_{sum}^{v} + u_{sum}^{v} - \delta_{max}^{v}}{1 - \delta_{max}^{v}}$$

$$\geq \frac{\frac{u_{sum}}{v-2} + \frac{u_{sum}}{v} - \frac{\delta_{max}}{v}}{1 - \frac{\delta_{max}}{v}}$$

The right-hand side of the inequality above is an increasing function of δ_{max} for $m \geq \frac{v(v-2)}{2v-2}$.

Since $\delta_i = \frac{C_i}{D_i}$ and because the task set τ is feasible if and only if $C_i \leq D_i$ for all tasks τ_i, we have that the greatest possible density for a feasible task set is given by $\delta_i^{max} \leq 1$.

Thus, when $m \geq \frac{v(v-2)}{2v-2}$, the schedulability is guaranteed if:

$$m \geq \frac{\frac{m}{v-2} + \frac{m}{v} - \frac{1}{v}}{1 - \frac{1}{v}}$$

$$m(1 - \frac{1}{v}) \geq \frac{m}{v-2} + \frac{m}{v} - \frac{1}{v}$$

$$v - \frac{2}{v-2} \geq 3 - \frac{1}{m}$$

This inequality is respected with $v = 4$ and $m \geq 2$.

Hence, any feasible P/D task set τ feasible on $m \geq 2$ unit-speed processors and a unit-speed network, is guaranteed to be schedulable by the P/D-DMS algorithm on m processors and a single real-time network with speed $v = 4$. ■

VI. SIMULATIONS

In this section we present the simulation results that validate the resource augmentation bound of the P/D-DMS algorithm after applying the DST transformation presented in Sections V and IV, respectively.

To generate feasible P/D task sets, we follow the guidelines presented in [19] for generating random task sets for multiprocessor systems, using the Stafford's Randfixedsum algorithm [20]. 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 P/D tasks with a fixed total density $\delta_{tot} = \sum \delta_i$. For a given total density δ_{tot}, the Randfixedsum algorithm returns n P/D tasks with density δ_i. For generating the P/D threads' densities we use again the Randfixedsum algorithm taking as an input the previous generated densities $\delta_i = \sum \delta_{i,j,k}$, obtaining a set of values $\delta_{i,j,k}$ for each P/D thread. The WCETs and end-to-end deadlines D_i are generated as recommended in [19]. Once all P/D threads are generated, their respective messages are generated and inserted within a P/D task by preserving their execution order. The total message density $\delta_{tot}^{messages}$, represents the utilization of the network. Thus, when a total message density is given, the Randfixedsum algorithm returns n messages of densities $\delta_i^{messages}$ for each task τ_i. For generating the messages' densities $\delta_i^{messages} = \sum \delta_{i,j,k}^{message}$ we use again the Randfixedsum algorithm taking as an input the previous generated densities $\delta_i^{messages}$. We consider that applications have implicit end-to-end deadlines ($D_i = T_i$) following a uniform distribution between the values $D_i^{min} = 100$ and $D_i^{max} = 10000$.

Figure 4(a) shows the number of accepted task sets over 1000 experiments for different given total message densities $\delta_{tot}^{messages}$. We simulate 4 P/D tasks that are partitioned by the P/D-DMS algorithm in a computing platform of 8 processors and 1 network. Thus the total utilization U_{tot} for these experiments is fixed to 8. Three different total message densities are analysed: (i) $\delta_{tot}^{messages}$ equal to the 10% of the fixed total utilization U_{tot}; $\delta_{tot}^{messages} = 0.8$, (ii) $\delta_{tot}^{messages}$ equal to the 5% of U_{tot}; $\delta_{tot}^{messages} = 0.4$, and (iii) $\delta_{tot}^{messages}$ equal to 1% of the total utilization; $\delta_{tot}^{messages} = 0.08$. It is possible to see that when $\delta_{tot}^{messages}$ increases, more speed v is required by the processors and the network to be able to schedule 100% of the task sets. This effect is modelled by Eq. 10 in our analysis.

In Figure 4(b) we show the number of accepted task sets for 1000 experiments. 4 P/D tasks have to execute in a computing platform composed of 1 network and 8 distributed processors. The total density δ_{tot} is fixed to 8. In this case we analyse the variations in respect of different individual thread density $\delta_{i,j,k}^{min}$ and $\delta_{i,j,k}^{max}$. Three different variations are compared: (i) $\delta_{i,j,k}^{min} = 0.1$ and $\delta_{i,j,k}^{max} = 0.2$, (ii) $\delta_{i,j,k}^{min} = 0.05$ and $\delta_{i,j,k}^{max} = 0.1$, and (iii) $\delta_{i,j,k}^{min} = 0.01$ and $\delta_{i,j,k}^{max} = 0.05$. $\delta_{tot}^{messages}$ is fixed to 5% of the δ_{tot}. It is possible to observe that if densities of tasks are larger, the more speed is needed to successfully schedule 100% of the task sets.

Figure 4(c) shows the number of accepted task sets over 1000 experiments, in which we vary the number of P/D tasks with a fixed total density $U_{tot} = \delta_{tot} = 8$ to be scheduled in a computing platform of 8 processors and 1 network. The total message density $\delta_{tot}^{messages}$ is fixed to 5% of U_{tot}. We compared three possible variations: (i) 4 P/D tasks, (ii) 6 P/D tasks and (iii) 8 P/D tasks. It is possible to see that when generating fewer P/D tasks for the same density δ_{tot}, the more speed v is required by the processors and the network to successfully schedule 100% of the task sets. Thus, whenever P/D densities δ_i increase, the probability of finding a schedulable partitioning with the P/D-DMS algorithm for P/D tasks, decreases.

Therefore, it is possible to observe though Figures 4(a-c) that in all cases; the P/D-DMS algorithm is able to find a schedulable partition by respecting its resource augmentation bound of 4.

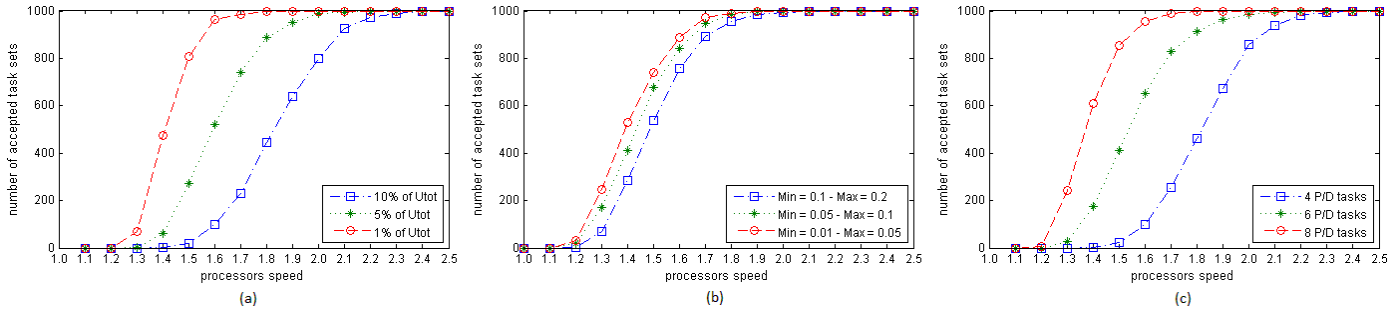

Figure 4. 1000 generated task sets varying (a) the total message density $\delta_{tot}^{messages}$, (b) the minimum thread density $\delta_{i,j,k}^{min}$ and maximum thread density $\delta_{i,j,k}^{max}$, and (c) the number of P/D tasks in the set τ.

VII. Conclusions and Future Work

This paper presented the P/D-DMS algorithm. The P/D-DMS algorithm makes use of the DST model for scheduling parallel/distributed fixed-priority fork-join real-time tasks. The P/D-DMS algorithm is shown to have a resource augmentation bound of 4. The DST is designed with two main objectives. The first one is to eliminate as many messages of a P/D task as possible by stretching a master thread, since a master thread is executed locally on its own processor. And the second objective is to reduce the possible interference in the network and in the processors by forcing P/D threads to execute within the master thread.

We are currently working on finding more efficient partitioning algorithms for P/D threads and messages by considering the specific structure (threads' offsets and intermediate deadlines) of distributed fixed-priority fork-join real-time tasks.

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 project FCOMP-01-0124-FEDER-037281 (CISTER); by FCT and the EU ARTEMIS JU funding, within projects ENCOURAGE (ARTEMIS/0002/2010, JU grant nr. 269354), ARROWHEAD (ARTEMIS/0001/2012, JU grant nr. 332987), CONCERTO (ARTEMIS/0003/2012, JU grant nr. 333053); by FCT and ESF (European Social Fund) through POPH (Portuguese Human Potential Operational Program), under PhD grant SFRH/BD/71562/2010.

References

[1] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comp. Surv.,* vol. 43, no. 35, p. 1–44, 2011.

[2] K. Lakshmanan, S. Kato and R. Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," in *Proc. of the IEEE 31st Real-Time Systems Symposium (RTSS 2010)*, 2010.

[3] F. Fauberteau, M. Qamhieh and S. Midonnet, "Partitioned Scheduling of Parallel Real-time Tasks on Multiprocessor Systems," in *ACM SIGBED Review, 8(Special Issue on Work-in-Progress (WiP) session of the 23rd Euromicro Conference on Real-Time System)*, 2011.

[4] M. Qamhieh, F. Fauberteau and S. Midonnet, "Performance Analysis for Segment Stretch Transformation of Parallel Real-time Tasks," in *Proceedings of the 5th Junior Researcher Workshop on Real-Time Computing (JRWRTC 2011)*, 2011.

[5] A. Saifullah, K. Agrawal, C. Lu and C. Gill, "Multi-core Real-Time Scheduling for Generalized Parallel Task Models," in *Proc. of the IEEE 32st Real-Time Systems Symposium (RTSS 2011)*, 2011.

[6] Y. Nimmagadda, K. Kumar, L. Yung-Hsiang and C. S. G. Lee, "Real-time moving object recognition and tracking using computation offloading," in *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010)*, 2010.

[7] OpenMP Architecture Review Board, "OpenMP application program interface V3.1 July 2011," www.openmp.org/wp/openmp-specifications/, online: last accessed April 2014.

[8] Message Passing Interface Forum, "MPI: A Message-Passing Interface standard version 2.2," online: http://www.mpi-forum.org/docs/docs.html, online: last accessed April 2014.

[9] R. Garibay-Martínez, L. L. Ferreira and L. M. Pinho, "A framework for the development of parallel and distributed real-time embedded systems," in *Proc. of 38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA'12)*, 2012.

[10] R. Garibay-Martínez, L. Ferreira, C. Maia and L. Pinho, "Towards transparent parallel/distributed support for real-time embedded applications," in *Proc. of the 8th IEEE International Symposium on Industrial Embedded Systems (SIES'13)*, 2013.

[11] J. J. Gutiérrez-García, J. C. P. Gutierrez and M. Gonzalez-Harbour, "Schedulability analysis of distributed hard real-time systems with multiple-event synchronization," in *Proc. of the12th Euromicro Conference on Real-Time Systems (ECRTS'00)*, 2000.

[12] 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.

[13] Q. Zhu, Y. Yang, M. Di Natale, E. Scholte and A. Sangiovanni-Vincentelli, "Optimizing the Software Architecture for Extensibility in Hard Real-Time Distributed Systems," *IEEE Transactions on Industrial Informatics,* vol. 6 , no. 4, pp. 621-636, Nov. 2010.

[14] P. Axer, S. Quinton, M. Neukirchner and R. Ernst, "Response-Time Analysis of Parallel Fork-Join Workloads with Real-Time Constraints," in *Proc. IEEE 25th Euromicro Conference on Real-Time Systems (ECRTS'13)*, 2013.

[15] N. Fisher, S. Baruah and T. P. Baker, "The partitioned scheduling of sporadic tasks according to static-priorities," in *Proc. of the IEEE 18th Euromicro Conference on Real-Time Systems (ECRTS'06)*, 2006.

[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] G. Laurent, N. Rivierre and M. Spuri, "Preemptive and non-preemptive real-time uniprocessor scheduling," 1996.

[18] S. K. Baruah, A. K.-L. Mok and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Proc. of the 11th IEEE Real-Time Systems Symposium (RTSS'90)*, Orlando, Florida, USA, 1990.

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

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