Computing and Informatics, Vol. 30, 2011, 657–679

DEADLINE MISSING PREDICTION THROUGH THE USE OF MILESTONES

Patricia Della Méa Plentz, Carlos Montez
Rômulo Silva de Oliveira

Department of Informatics and Statistics (INE)
Department of Automation and Systems (DAS)
Federal University of Santa Catarina (UFSC)
88040-900, Florianópolis, SC, Brazil
e-mail: plentz@inf.ufsc.br, {montez, romulo}@das.ufsc.br

Communicated by Jiří Šafařík

Abstract. Distributed Real-Time Thread is an important concept for distributed real-time systems. Distributed Threads are schedulable entities with an end-to-end deadline that transpose nodes, carrying their scheduling context. In each node, the thread will be locally scheduled according to a local deadline, which is defined by a deadline partitioning algorithm. Mechanisms for predicting the missing of deadlines are fundamental if corrective actions are incorporated for improving system quality of service. In this work, a mechanism for predicting missing deadlines is proposed and evaluated through simulation. In order to illustrate the main characteristics of the proposed mechanism, experiments will be presented taking into account different scenarios of normal load and overload. Simulations show that the deadline missing prediction mechanism proposed presents good results for improving the overall performance and availability of distributed systems.

Keywords: En-to-end deadline, distributed threads, prediction mechanisms

1 INTRODUCTION

In distributed real-time systems, like those used in factory automation, the end-to-end deadline of tasks is a timing constraint imposed by the system application. Mechanisms for predicting missing deadlines are fundamental if corrective actions are incorporated for improving the quality of services of the system.

This kind of system can be implemented by Distributed Threads (DTs) [1], which is an abstraction of distributed tasks that transpose physical nodes boundaries carrying out remote calls. This abstraction is powerful because it demands low overhead in its execution. Distributed threads can be used to represent control and supervision tasks in factory automation systems, visiting several nodes to collect information for the purpose of analyzes and diagnoses. These tasks usually have firm deadlines and they are distributed.

This paper extends previous work [12] defining deadline missing prediction mechanisms for systems based on distributed threads. These mechanisms use information such as available slack time of a distributed thread and define a probability of a distributed thread meeting its deadline. To achieve this goal, a system architecture that supports the distributed thread abstraction with timing constraints [12] is used in this work. This system architecture is present in each system node and is composed by local tasks, an interceptor and an aperiodic server which services DTs. This architecture meets deadlines of hard periodic local tasks while trying to improve response time of DTs.

The rest of the paper is organized as follows: Section 2 presents related work with distributed threads, end-to-end deadline partitioning methods and deadline missing prediction mechanisms. Section 3 describes main concepts about distributed threads and deadline partitioning methods found in the literature. Section 4 presents the proposed model used in this work and explains probabilistic known itineraries for DTs used in this work. The deadline missing prediction mechanisms introduced in this work are presented in Section 5. Applicability and performance results of the proposed prediction mechanisms are shown in Section 6. Section 7 contains the final remarks.

2 RELATED WORK

Some works in the literature address distributed thread implementations [1, 3, 4, 18, 2, 11]. In [1], the Alpha's kernel programming model is described, which is based on distributed threads that transpose system's nodes carrying its timing constraints in order to make system resource management possible. In the context of Real-Time CORBA [7], the works [3, 4] take into account some aspects related to DTs. The work described in [3] addresses the distributed thread-scheduling problem according to the Real-Time CORBA approach at Tempus real-time middleware. The main contribution of that work is scheduling of DTs that are subject to arbitrarily-shaped Jensen's time-utility functions (TUF) time constraints. In [4] an approach for integrating release guards synchronization protocol [8] is described with CORBA DTs ensuring appropriate release times for subtasks along an end-to-end computation. Moreover, that work presents performance comparison between DTs and federated event channel for both random workloads and different canonical communication topologies.

In [2, 18, 11] DTs are studied in the Java context. The technique introduced in [2] to implement DTs in a portable middleware is based on byte-code transformation of subroutines, instead of the entire client application. They do not address aspects related to timing constraints of DTs. In [18], a scheduling algorithm for DTs and a protocol for ensuring thread integrity are proposed. It is shown that the scheduling algorithm in conjunction with the proposed protocol ensures that handlers of threads encountering failures during their execution complete within a bounded time, yielding bounded thread cleanup time. In [11] DTs are implemented through Real-Time Specification for Java (RTSJ) and a system architecture is proposed which is flexible enough to accommodate to a hybrid task set as well as several scheduling policies. The main contribution of that work was to implement DTs without modifying the Java Virtual Machine with the creation of new classes and interfaces, as opposed to many other works in the literature. In this work, we continue exploring the flexibility of this system architecture by the introduction of the deadline missing prediction mechanisms.

In distributed systems, the end-to-end deadline of a task that can transpose the system nodes needs to be partitioned in local deadlines, which will be used by local schedulers. Works presented in the literature propose different methods for partitioning end-to-end deadlines [16, 17, 9]. The critical path concept is introduced in [16], which minimizes the overall laxity of the path. This concept is used for assigning slices (execution windows with static positions in time) to subtasks. Two laxity ratio metrics were proposed in that work. The first one assigns a subtask deadline based on its execution time, the other one assigns a subtask deadline based on the number of subtasks in the critical path. That work is in the hard distributed real-time systems context and the proposed method is optimal in the sense that it maximizes the minimum task laxity in the application. The work proposed in [17] is a refinement of [16] in the sense that it circumvents difficulties associated with deadline distribution over subtasks with unknown initial task assignment. Threshold Laxity Ratio (THRES) and Adaptive Laxity Ratio (ADAPT) metrics was proposed in [17] for minimizing the maximum task lateness. Evaluations show that proposed techniques present better results in relation to those presented in [16].

Deadline assignment algorithms are also studied in the multi-hop networks field, as part of admission control. In [9], the authors propose two deadline assignment algorithms: Fair Laxity Distribution (FLD) and Unfair Laxity Distribution (ULD). FLD algorithm distributes the laxity in a fair way between the visited nodes. On the other hand, ULD algorithm distributes the laxity proportional to the minimum sojourn time that can be guaranteed. In the performance comparison between two algorithms, with intermediate values of the end-to-end deadline, ULD with NP-EDF scheduling and FLD with FIFO scheduling present better results than classical methods, such as those in [16].

Using local information, such as response times and local deadlines, for example, it is possible to predict the future behavior of the system. In [5, 6] the authors deal with the prediction of response times. The first one introduces algorithms to estimate the probability of a deadline to be met in embedded systems. This is done

through the prediction of the response time of a service, in which past executions of this service and the historical data about past executions of all the services supported by the program are used. In that work an implementation of an application as proof of concept was carried out, showing the feasibility of the proposed approach. The goal of the work described in [6] is to minimize the number of exceptions thrown during process execution and to reduce the cost associated with them when these exceptions cannot be avoided. The proposed early prediction mechanism is based on the observation that, when exceptions are inevitable, it may be beneficial to throw them during the execution of some earlier activity, which has not missed its deadline yet. The mechanism proposed exploits information about exception throw costs, which are supplied by the business analyst, as well as information provided by existing workflow systems, such as statistical measurements based on past process executions and current status data.

3 DISTRIBUTED THREADS

In many computing problems, applications may be distributed over multiple machines to achieve desired levels of performance, scalability and robustness. In this case a control flow that moves itself during the remote call from one virtual machine to another and returns to the original machine when the method returns is adequate to implement distributed applications. This control flow abstraction is known as *Distributed Threads* [3]. It has a globally unique identifier and can share physical resources (e.g. processor, disk, I/O) and logical resources (e.g. locks), which can be subject to mutual exclusion constraints. The distributed thread concept was initially introduced in the Alpha distributed real-time OS kernel [1].

In the field of real-time systems, a distributed thread can carry its execution context as it transposes machine boundaries, including its scheduling parameters (e.g., time constraints, execution time), identity and security credentials [18]. Distributed applications consist of many distributed threads executing concurrently and asynchronously. Local scheduler carries out the scheduling of these distributed threads.

In each node that a distributed thread executes, a local segment of this distributed thread is created, being implemented by a local thread provided by the operating system. With Figure 1 it is possible to visualize three distributed threads being composed by local segments of execution.

When a distributed thread is created, the object and the operation that it will execute are defined and it begins its execution by invoking an object operation. This node where it began is called *source node*. A distributed thread has one execution point in the whole system, which is called *head node*. All nodes that host part of the execution of a distributed thread are known as *segment nodes*. A distributed thread that is executing can create a new distributed thread (e.g. through one-way invocations). This new distributed thread may begin executing in another node of the system. However, this kind of invocation is not considered in this work.

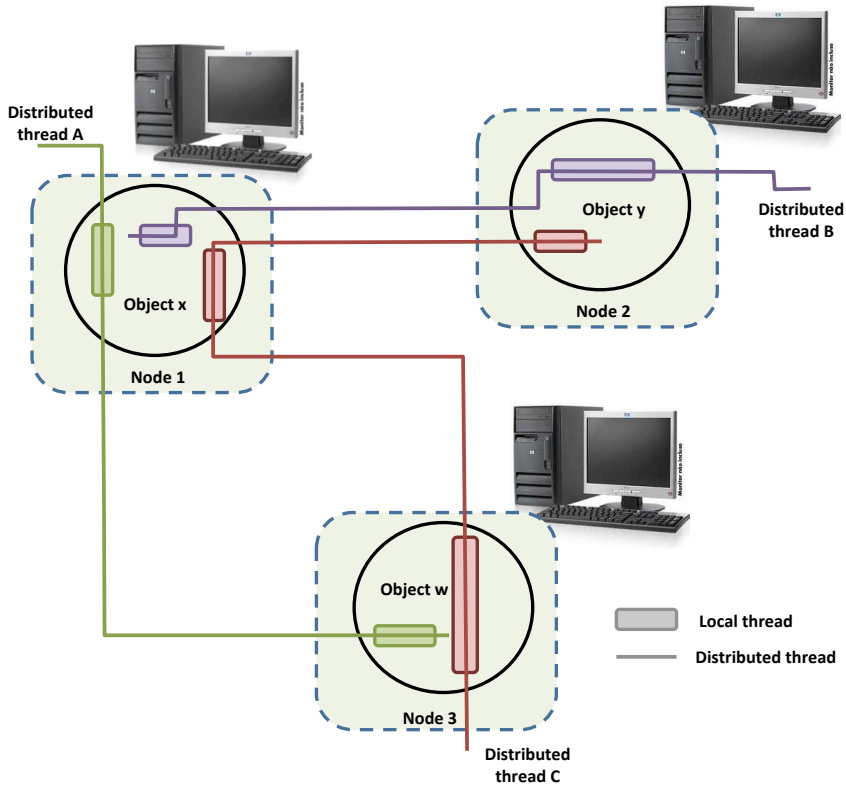

Fig. 1. Distributed threads and local segments

This abstraction can be implemented as part of the operating system (e.g., CMU Alpha, OSF MK7. 3a), as part of middleware (e.g., Real-Time Corba/Dynamic Scheduling) and as part of the programming language (e.g., Java).

3.1 Methods for End-to-End Deadline Partitioning

Distributed real-time systems are composed of local and distributed tasks. Usually, an end-to-end deadline tells the system the urgency of a distributed task. Scheduling policies are used to schedule these tasks according to their deadlines. The partition of the end-to-end deadline in local deadlines is a common practice for local scheduling. These local deadlines are used by local schedulers in a scheduling policy.

When considering a distributed system composed of distributed threads, the end-to-end deadline can be split out using their local segments. A distributed thread can execute computations before a remote call only or also after a remote call (that is, when it returns to the node). For example, if a distributed system with five pipelined nodes is considered, a distributed thread may have five local segments

because it executes computations only before a remote call, or it may have ten local segments if it executes computations before and after a remote call.

As described in [14] there are two main ways to carry out the end-to-end deadline partition. The first one is a static way where the partitioning occurs before the distributed task begins its execution. All the local deadlines are defined and they do not suffer from changes during distributed task execution. The second one is a dynamic way where the local deadlines are redefined each time the distributed task arrives at a node. This kind of end-to-end deadline partitioning takes into account current system load and generates additional overhead.

The literature presents some end-to-end deadline partitioning methods [17, 9, 16, 10]. In [10] three methods are defined that do not deal with system load (Effective Deadline – ED, Equal Slack – ES, Equal Flexibility – EQF). They create estimated local deadlines that consider just the end-to-end deadline, arrival and execution times of distributed threads. As exposed in [14], ED is a simple method that considers just the estimated execution time of local segments in the definition of local deadlines, the EQS method divides the distributed thread's total slack equally among all its local segments and method EQF divides the distributed thread's total slack proportionally to execution time among all its local segments.

4 PROPOSED MODEL

In this work, we are supposing a distributed real-time system like those in factory automation. Only one application executes in all nodes of the system at a given moment and it is implemented by Real-Time Distributed Threads (DTs). This system has a dynamic load because DTs can be created at runtime. Therefore, there may be moments when the system is overloaded.

The proposed model is the same as that used in [14] which is composed of local periodic tasks with hard deadlines and distributed aperiodic tasks with firm deadlines. The distributed aperiodic tasks are represented by Distributed Threads. These DTs have timing constraints and they are recurrent, that is, they can be activated more than once on the system. Furthermore, in this work it is considered that DTs execute operations before and after a remote call. An end-to-end deadline and an estimated execution time are defined when a distributed thread is created. These timing constraints are carried by each real-time distributed thread as they transpose system nodes.

Figure 2 shows details of the system architecture (which was introduced in [12]). Distributed threads are serviced by interceptors, which are local threads with execution times previously assigned. It means that these interceptors are considered as one additional periodic task of the system. Each interceptor maintains a list of distributed threads that execute (or executed) in each node. When a distributed thread arrives at a node (head node), the local interceptor services it. Then, the interceptor verifies if a local segment of this distributed thread already exists. In affirmative case, this local segment is activated to execute on behalf of the dis-

tributed thread that arrived. Otherwise, a distributed thread local segment is created by the interceptor, which will execute on behalf of its parent distributed thread.

In both cases, all the distributed thread properties (such as identifier and timing constraints) are inherited by the distributed thread local segment. Each distributed thread has a history, which records these properties, as well as its last activations.

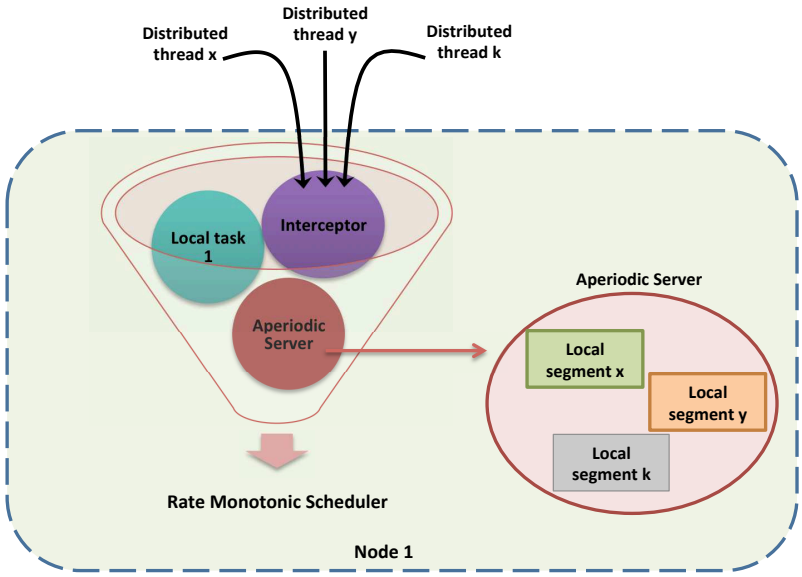

Fig. 2. System architecture

In each system node there is an aperiodic server responsible for the execution of distributed thread's local segments. The interceptor puts in the aperiodic server's queue the distributed threads local segments that arrive at the node. In this work the use of the Sporadic Server [15] is proposed for the scheduling of the distributed thread local segments and the Earliest Deadline First (EDF) algorithm [15] for preemptively scheduling the aperiodic server's queue. The Rate Monotonic (RM) algorithm [15] will be used in each system node, which will preemptively schedule the hard periodic local tasks, interceptor and aperiodic server. As hard periodic local tasks are considered critical for the application, their scheduling must not be jeopardized by the scheduling of DTs with firm deadlines. The scheduling approach used in this work is composed of two stages: partitioning of the end-to-end deadline and local scheduling. This is not an uncommon procedure in the real-time literature [8]. The distributed thread's timing attributes influence the scheduling only, and the local schedulers are considered independent. They do not collaborate with each other in an explicit way. The aim of this architecture is twofold: to guarantee

the deadlines of the hard local periodic tasks and to reduce the response time of the firm distributed aperiodic tasks in order to meet the distributed threads end-to-end deadlines.

During its execution in the system, a distributed thread defines its control flow through the remote calls executions at different nodes, as Figure 3 shows. In this figure, it is possible to visualize distributed thread executing methods A, B and C in nodes 1, 2 and 3, respectively. According to application behavior, the distributed thread can go to node 2 (method B) or node 3 (method C). For this reason, a distributed thread can be considered autonomous and its remote calls itinerary can be recorded in a history. In each new activation, the distributed thread may follow the previous itinerary or a new itinerary.

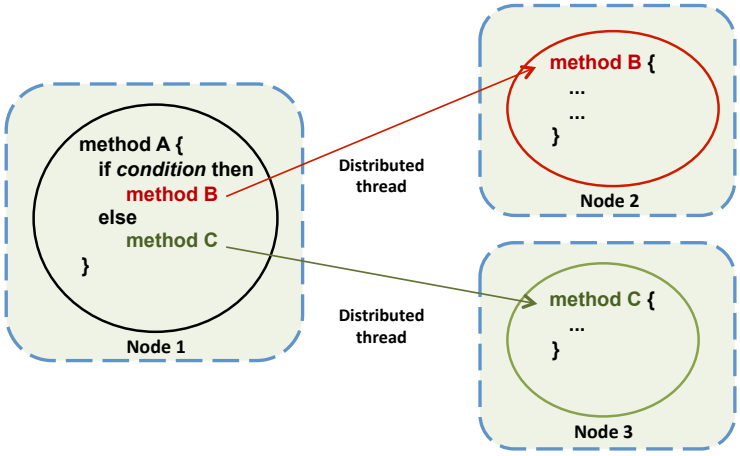

Fig. 3. Distributed thread execution flows alternatives

Because of its history and autonomous behavior we can say that in its next activation a distributed thread can follow an itinerary that is not completely known or completely random. This itinerary can be defined based on the distributed thread's history in a probabilistic way [13].

4.1 Probabilistic Known Itineraries for Distributed Threads

Due to its autonomous nature, a distributed thread can define many different itineraries, which can be kept stored in a log. In each new activation, a distributed thread will follow a different itinerary. Through the scanning of this log it is possible to identify four main itineraries that a distributed thread would follow: Longest Itinerary, Shortest Itinerary, Most Likely Itinerary and Average Itinerary. These itineraries can be visualized in Figure 4 and are described as follows.

As described in [14], the first itinerary considers the greater path that a distributed thread may follow. The second one considers the lesser path that it

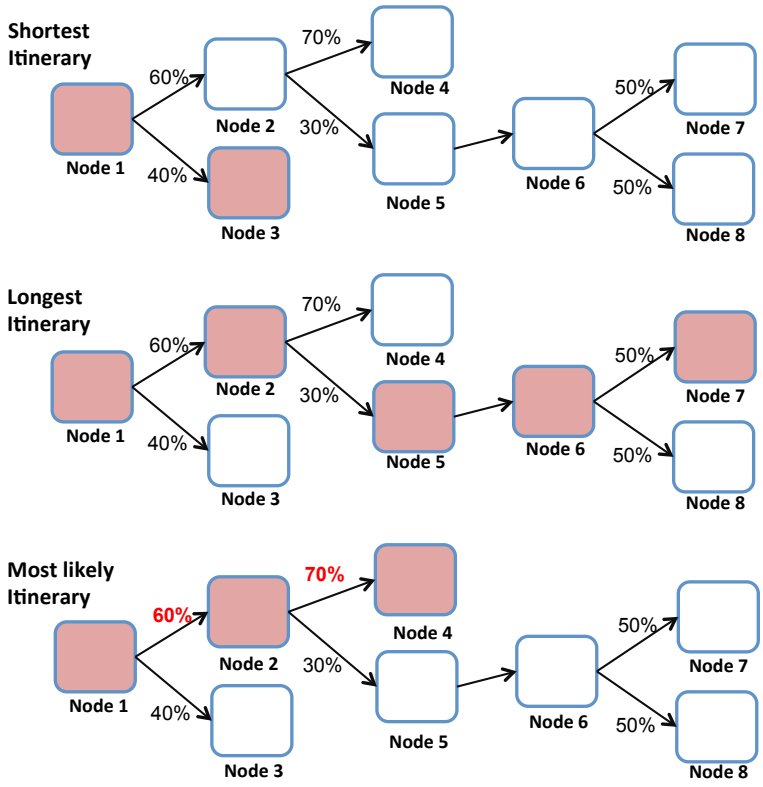

Fig. 4. Possible itineraries of a distributed thread

may follow and the third one identifies in the distributed thread log which path was followed more times by a distributed thread. The Average Itinerary computes an average of the itineraries kept stored in the distributed thread log considering the different probabilities of a distributed thread following one or another itinerary.

At each new distributed thread activation (before it begins to execute) a deadline partitioning method can be chosen to define its local deadlines according to some distributed thread itinerary. In this work the EQF deadline partitioning method is used because of its good results presented in [10]. Because of its autonomous nature, a distributed thread may not follow this pre-determined itinerary or may follow it partially. In that case, a new partitioning needs to be computed. When the system observes that the distributed thread is in a node that does not belong to the original itinerary, a new deadline definition is made through the same itinerary last used starting from the present node.

5 PROPOSED DEADLINE MISSING PREDICTION MECHANISMS

The performance of non-critical distributed real-time systems can be improved by the implementation of mechanisms for early detection of deadline missing. A deadline missing prediction mechanism can be used to determine the probability of a distributed thread to miss its end-to-end deadline. Remedial actions related to deadline missing should be carried out in time to improve system performance.

End-to-end deadline missing prediction mechanisms can be launched at start nodes, end nodes or at intermediary nodes of the distributed system. In the first case, it can compute bad results because the distributed thread is only beginning its execution. In the second case, maybe it will be too late for taking any remedial action. Considering these questions, our prediction mechanisms are launched at the node which represents the middle of the distributed thread execution, that is, the prediction is made when the response time of an ongoing distributed thread is equal to or greater than half of its end-to-end deadline.

In this work we propose deadline missing prediction mechanisms that define estimated local response times, called *milestones*. These milestones are used as base in the calculation of the probability of a distributed thread meeting its end-to-end deadline.

5.1 Mechanisms Based on Milestones

The deadline missing mechanisms based on milestones consider only the distributed thread information previously known for the definition of a estimated response time that is used in the calculations of the probability of a distributed thread meeting its deadline. Three different ways to generate the milestones are proposed, all of them are based on the end-to-end deadline partitioning methods proposed in [10]:

MilestoneED (MED): based on the *Effective Deadline* partitioning method [10];

MilestoneEQS (MEQS): based on the *Equal Slack* partitioning method [10];

MilestoneEQF (MEQF): based on the *Equal Flexibility* partitioning method [10].

As described previously, the prediction mechanisms run at the node which represents the middle of the distributed thread execution, that is, the prediction is made when the response time of an ongoing distributed thread is equal to or greater than half of its end-to-end deadline. Because of this observation, the milestones mechanisms define estimated local response times for all nodes that belong to the distributed thread possible itineraries.

The itineraries described in Section 4.1 can be used in conjunction with the milestones predictors. With the combination of these itineraries and the milestones predictors, it is possible to define 12 milestones predictors:

- MED – Longest (MED-Lt);
- MED – Shortest (MED-St);

- MED – Average (MED-Av);
- MED – Most Likely (MED-ML);
- MEQS – Longest (MEQS-Lt);
- MEQS – Shortest (MEQS-St);
- MEQS – Average (MEQS-Av);
- MEQS – Most Likely (MEQS-ML);
- MEQF – Longest (MEQF-Lt);
- MEQF – Shortest (MEQF-St);
- MEQF – Average (MEQF-Av);
- MEQF – Most Likely (MEQF-ML).

The generation of the milestones is carried out in the source node, before the distributed thread begins its execution. For each *predictor-itinerary* pair, the prediction is carried out considering the execution time of the remote calls that compose an itinerary. The generated milestones are carried by the distributed thread, in a dynamic structure that is updated as the distributed thread transposes the system nodes.

The next subsections describe the predictors *MilestoneED*, *MilestoneEQS* and *MilestoneEQF*.

5.2 MilestoneED – MED

This mechanism defines estimated local response times for each distributed thread' local segment (milestone), using Equation (1) [10]. These MED milestones are used in the definition of the index P_k, in the following way:

$$\begin{aligned} dl(si) &= d(TD) - \sum_{j=1+1}^{m} pex(sj) \\ Slack &= (MED(s_i) - rl(s_i))/Dff_k; \\ P_k(MED) &= slack + 0.5 \end{aligned} \tag{1}$$

where $MED(s_i$ is the milestone of the DT_k' local segment i defined through 1, $rl(s_i)$ is the response time of the DT_k'local segment s_i, and Dff_k is the end-to-end deadline of the DT_k.

It is defined the probability of the DT_k meets its deadline in the following way:

$$Prob_k(MED) = \begin{cases} 0 & P_k(MED) < 0 \\ P_k(MED) & 0 \leq P_k(MED) \leq 1 \\ 1 & P_k(MED) > 1 \end{cases}$$

where $Prob_k(MED)$ represents the probability of DT_k to meet its end-to-end deadline according to the MED prediction mechanism.

5.3 MilestoneEQS – MEQS

This mechanism defines estimated local response times for each distributed thread local segment (milestones), using Equation (2) [10]. These milestones MEQS are used in the definition of the index P_k, in the following way:

$$\begin{aligned} dl(si) &= ar(si) + pex(si) + (d(TD) - ar(si) \\ &\quad - \sum_{j=1}^{m} pex(sj)) \setminus (m - i + 1) \\ Slack &= (MEQS(s_i) - rl(s_i))/Dff_k; \\ P_k(MEQS) &= slack + 0.5 \end{aligned} \tag{2}$$

where $MEQS(s_i)$ is the milestone of DT_k' local segment i defined through 2. It is defined the probability of DT_k meets its deadline in the following way:

$$Prob_k(MEQS) = \begin{cases} 0 & P_k(MEQS) < 0 \\ P_k(MEQS) & 0 \leq P_k(MEQS) \leq 1 \\ 1 & P_k(MEQS) > 1 \end{cases}$$

where $Prob_k(MEQS)$ represents the probability of DT_k to meet its end-to-end deadline according to the MEQS prediction mechanism.

5.4 MilestoneEQF – MEQF

This mechanism defines estimated local response times for each distributed thread local segment (milestones), using Equation (3) [10]. These milestones MEQF are used in the definition of the index P_k, in the following way:

$$\begin{aligned} dl(si) &= ar(si) + pex(si) + (d(TD) - ar(si) \\ &\quad - \sum_{j=1}^{m} pex(sj)) * \left(\frac{pex(si)}{\sum_{j=1}^{m} pex(sj)} \right) \\ Slack &= (MEQF(s_i) - rl(s_i))/Dff_k; \\ P_k(MEQF) &= slack + 0.5 \end{aligned} \tag{3}$$

where $MEQF(s_i)$ is the milestone of the DT_k' local segment i defined through 3. It is defined the probability of DT_k meets its deadline in the following way:

$$Prob_k(MEQF) = \begin{cases} 0 & P_k(MEQF) < 0 \\ P_k(MEQF) & 0 \leq P_k(MEQF) \leq 1 \\ 1 & P_k(MEQF) > 1 \end{cases}$$

where $Prob_k(MEQF)$ represents the probability of the DT_k meeting its end-to-end deadline according to the MEQF prediction mechanism.

6 SIMULATIONS

Simulations were carried out with the objective of comparing the performance of the milestone mechanisms. The *Relative Error Rate* ($E(z)$) and *Correct Prediction Rate* (CP(z)) metrics were used to compare the results of the proposed mechanisms. The first one shows the distance of the resulting prediction in relation to an exact prediction and it is defined as [5]:

$$E_k(z) = \begin{cases} 1 - Prob_k(z) & R_k \leq D_k \\ Prob_k(z) & R_k > D_k \end{cases}$$

where R_k and D_k are the response time and the end-to-end deadline of a DT_k, respectively. With this metric, we have a measure of the capacity of each mechanism in doing correct predictions as the value of k increases.

The second metric just considers if the resulting prediction is greater or less than 0.5 and if the distributed thread met or missed its deadline. If the deadline meeting prediction of a DT_k ($Prob_k(z)$) is equal to or greater than 0.5, and the distributed thread actually meets its deadline, then the $CP(z)$ metrics accounts as correct prediction of the mechanism z. If the deadline meeting prediction of a distributed thread is less than 0.5 and the distributed thread misses its deadline, then the $CP(z)$ metrics also accounts as correct prediction of the mechanism z. Formally, $CP(z)$ metrics is defined as follows:

If ($Prob_k(z) < 0.5$) and ($R_k > D_k$) then
    CorrectPrediction(z)+ = 1;
If ($Prob_k(z) \geq 0.5$) and ($R_k \leq D_k$) then
    CorrectPrediction(z)+ = 1;
$CP(z)$ = CorrectPrediction(z)/NPredictions(z);

where NPredictions(z) is the total amount of the predictions carried out by mechanism z.

Three different load scenarios were simulated and analyzed in this work. The simulation conditions as well as results are described below.

6.1 Simulation Conditions

In this system there are 16 interconnected nodes, in each node there are four hard periodic local tasks, whose periods are 10 ut, 20 ut, 40 ut and 80 ut. Those tasks have relative deadlines equal to their periods. The interceptor task and aperiodic server task are one of the periodic local tasks. The capacity of the Sporadic Server is 5 ut and its period is 10 ut.

A set with 90 different kinds of DTs was used for the simulations. This set is composed by 30 pipelines DTs, 30 balanced DTs and 30 non-balanced DTs (Figure 5). 100 different configurations have been generated, each configuration is composed of 9 DTs randomly chosen from the distributed thread's set. For these 9 DTs, 3 DTs are of pipeline type, 3 DTs are of balanced type and 3 DTs are of non-balanced

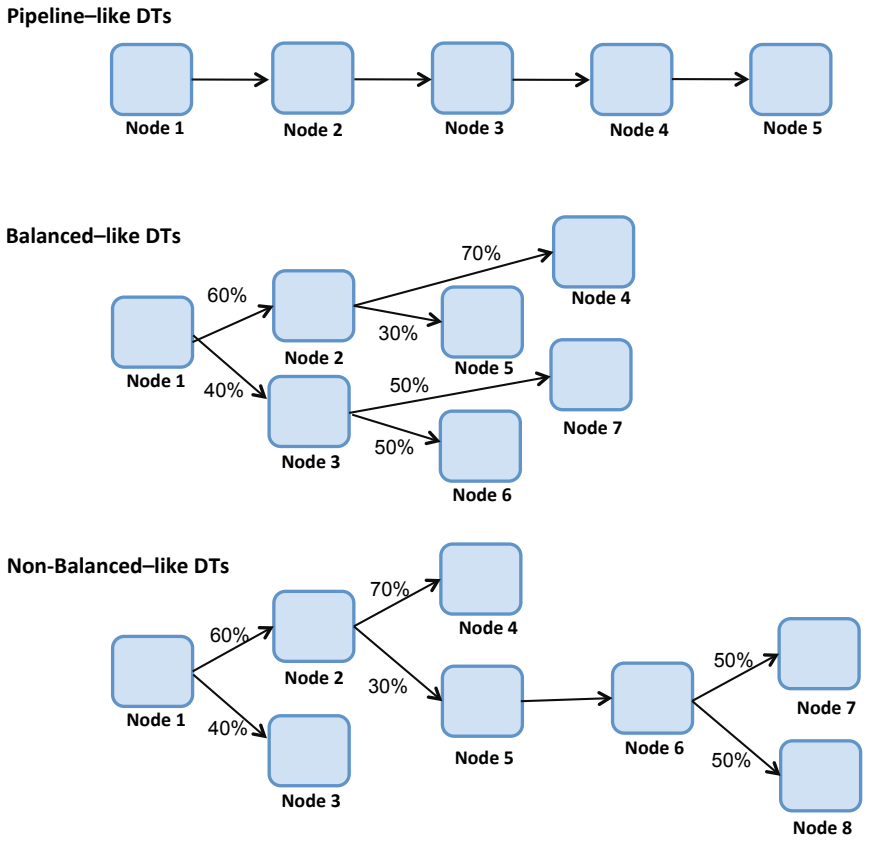

Fig. 5. Types of distributed threads

type. The average executing time of each distributed thread local segment varies from 5 ut to 200 ut.

In each configuration, 5 different values of deadlines were used in the range from 100 ut to 900 ut where 100 ut represents a tight deadline, 300 ut, 500 ut and 700 ut represent fair deadlines and 900 ut represents a loose deadline. The DTs arrival time rate follows an exponential distribution with an average of 700 ut between arrivals. Furthermore, communication network delay has uniform distribution between 1 ut and 2 ut and it is assumed that there is no network partition. The simulation time was equal to 20 000 ut for each configuration.

6.2 Simulation Results

The milestones predictors (MED, MEQS and MEQF) were used in conjunction with the itineraries Longest (Lt), Shortest (St), Average (Av.) and Most Likely (ML.), defining 12 predictors (Section 5.1).

Table 1 shows the results of the MED, MEQS and MEQF predictors, through the Relative Error Rate ($E(z)$) metrics. Each configuration was run for 20 000 ut. Since each configuration is composed of 9 distributed threads, we have around 250 predictions during each run of this 20 000 ut simulation. For each configuration we computed the average Relative Error Rate. Table 1 shows the average Relative Error Rate of the 100 different configurations simulated. The values presented in this table include the confidence interval (column *CI*) to a level of confidence of the 95 %, considering a sample of size 100. The best result for each deadline was represented as boldface.

	Deadline, Error Rate and Confidence Interval (CI)									
	100		300		500		700		900	
Prediction Mechanisms	*Error*	*CI*	*Error*	*CI*	*Error*	*CI*	*Error*	*CI*	*Error*	*CI*
MED-Lt	**0.047**	±0.005	0.234	±0.010	0.184	±0.013	**0.110**	±0.012	**0.074**	±0.012
MED-St	0.085	±0.008	0.266	±0.011	0.191	±0.014	0.114	±0.013	0.077	±0.012
MED-Av	0.055	±0.006	0.245	±0.010	0.186	±0.013	0.111	±0.012	0.075	±0.012
MED-ML	0.065	±0.007	0.242	±0.010	0.185	±0.013	0.111	±0.012	0.075	±0.012
MEQS-Lt	0.052	±0.005	0.203	±0.011	0.194	±0.010	0.135	±0.011	0.091	±0.011
MEQS-St	0.076	±0.006	0.221	±0.011	0.193	±0.010	0.132	±0.011	0.089	±0.011
MEQS-Av	0.061	±0.005	0.206	±0.011	0.193	±0.010	0.135	±0.011	0.091	±0.011
MEQS-ML	0.062	±0.006	0.209	±0.011	0.191	±0.010	0.133	±0.011	0.091	±0.011
MEQF-Lt	0.049	±0.005	**0.194**	±0.010	0.175	±0.009	0.120	±0.010	0.082	±0.011
MEQF-St	0.078	±0.006	0.223	±0.010	0.181	±0011	0.118	±0.011	0.080	±0.011
MEQF-Av	0.059	±0.005	0.198	±0.010	0.175	±0.009	0.119	±0.010	0.082	±0.011
MEQF-ML	0.062	±0.006	0.201	±0.010	**0.173**	±0.009	0.118	±0.010	0.082	±0.011

Table 1. Relative error rate of the prediction mechanisms

It is possible to visualize that none of the milestones predictors presented better results in all deadlines. In the deadlines equal to 100, 700 and 900 the MED-Lt was the better predictor. In the deadlines equal to 300 and 500, the MEQF-Lt and MEQF-ML predictors were the better predictors. The MEQS predictor did not obtain better results in none of the simulated deadlines. Through these results we can say that MEQF is better than the other milestone predictors in the Relative Error Rate metrics because it is more difficult to compute a correct prediction with fair deadlines.

The Longest (Lt) itinerary reaches best results in almost all simulated deadlines (with exception of the deadline 500). Figures 6, 7 and 8 show the values of Table 1. The MED, MEQS and MEQF are plotted with the four possible distributed thread itineraries: Lt, St, Av and ML.

Because the Longest (Lt) itinerary showed better results that the other itineraries, it will be plotted in the graphic of Figure 9, with each *Milestone* predictor. This graphic shows the good results of the MEQF predictor in the fair deadlines.

Table 2 shows the results of the milestones predictors, through the Correct Prediction Rate ($CP(z)$) metrics. Again we have around 250 predictions during each run of simulation. For each configuration we computed the average Correct Predic-

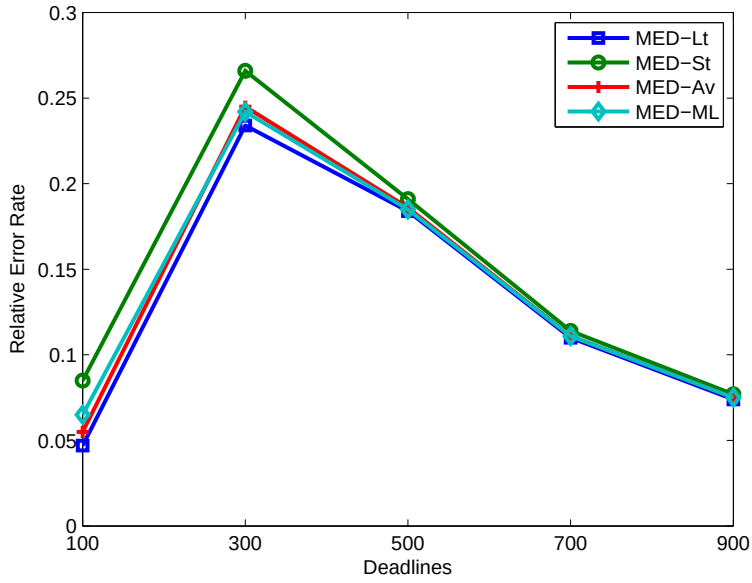

Fig. 6. Relative error rate of the MED predictor

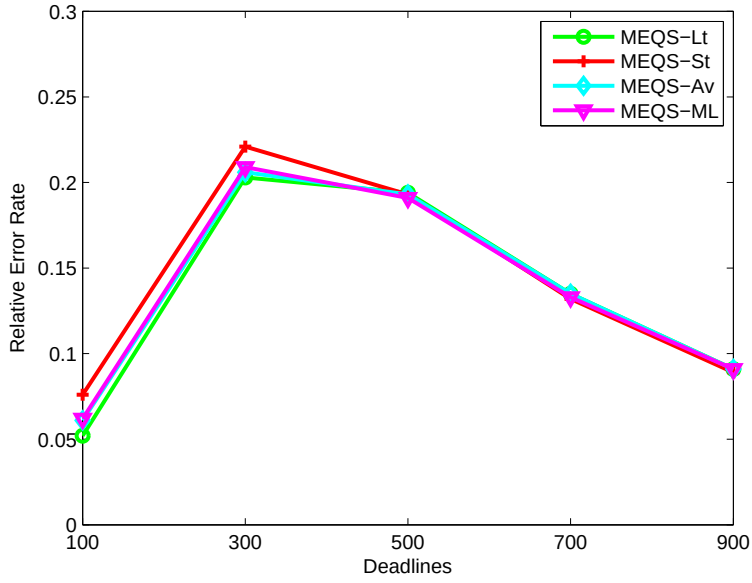

Fig. 7. Relative error rate of the MEQS predictor

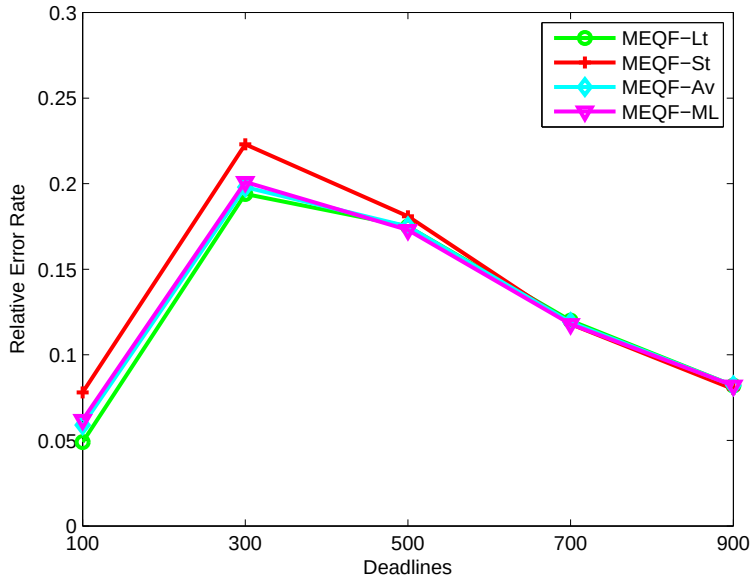

Fig. 8. Relative error rate of the MEQF predictor

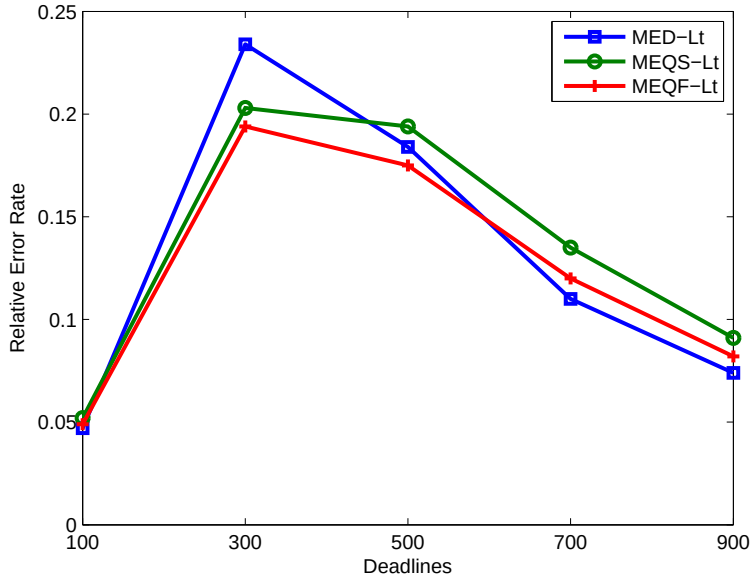

Fig. 9. Relative error rate of the MED, MEQS and MEQF mechanisms

tion Rate. Table 2 shows the average Correct Prediction Rate of the 100 different configurations simulated. The confidence interval (column CI) to a level of confidence of the 95 % considers a sample of size 100.

The results show that MEQF predictor presented best results in all simulated deadlines. Lt itinerary reached better results in the deadline 100 while the St itinerary was better for deadline 900. Av itinerary presented best results in deadlines 100, 300, 700 and 900. ML itinerary presented best results in deadlines 500, 700 and 900.

	Deadline, Correct Prediction Rate (CP) e Confidence Interval (CI)									
	100		300		500		700		900	
Prediction Mechanisms	*CP*	*CI*	*CP*	*CI*	*CP*	*CI*	*CP*	*CI*	*CP*	*CI*
MED-Lt	0.979	±0.028	0.823	±0.075	0.859	±0.068	0.932	±0.049	0.960	±0.038
MED-St	0.945	±0.045	0.760	±0.084	0.845	±0.071	0.924	±0.052	0.955	±0.041
MED-Av	0.977	±0.029	0.803	±0.078	0.853	±0.069	0.930	±0.050	0.959	±0.039
MED-ML	0.958	±0.039	0.805	±0.078	0.856	±0.069	0.930	±0.050	0.958	±0.039
MEQS-Lt	0.987	±0.022	0.871	±0.066	0.873	±0.065	0.918	±0.054	0.953	±0.041
MEQS-St	0.981	±0.027	0.863	±0.067	0.875	±0.065	0.924	±0.052	0.958	±0.039
MEQS-Av	0.987	±0.022	0.873	±0.065	0.877	±0.064	0.920	±0.053	0.955	±0.041
MEQS-ML	0.983	±0.025	0.869	±0.066	0.879	±0.064	0.922	±0.053	0.955	±0.040
MEQF-Lt	**0.989**	±0.021	0.913	±0.055	0.921	±0.053	0.949	±0.043	0.966	±0.035
MEQF-St	0.981	±0.027	0.876	±0.065	0.904	±0.058	0.948	±0.044	**0.967**	±0.035
MEQF-Av	**0.989**	±0.020	**0.917**	±0.054	0.924	±0.052	**0.951**	±0.042	**0.967**	±0.035
MEQF-ML	0.984	±0.024	0.908	±0.057	**0.925**	±0.052	**0.951**	±0.042	**0.967**	±0.035

Table 2. Correct Prediction Rate of the Deadline Missing Prediction Mechanisms

Through these results, it is possible to say that none of the itineraries presented better results than the others in all simulated deadlines. Figures 10, 11 and 12 show the values of Table 2. The MED, MEQS and MEQF are plotted with the four distributed thread possible itineraries: Lt, St, Av and ML.

With the aim of analyzing the results of the Longest (Lt) itinerary with each Milestone predictor, Figure 13 shows a graph of MED, MEQS and MEQF predictors with Lt itinerary. It shows that MEQF predictor presents better results in all simulated deadlines and this confirms its good behavior as a deadline prediction mechanism.

7 CONCLUSIONS

In this paper we have focused on distributed real-time systems, such as those used in factory automation. Distributed threads may implement this kind of system and they have probabilistic knowledge about their remote calls itineraries. An end-to-end deadline partitioning method was applied considering this probabilistic knowledge, and through it local deadlines were defined which are used by local scheduling policies.

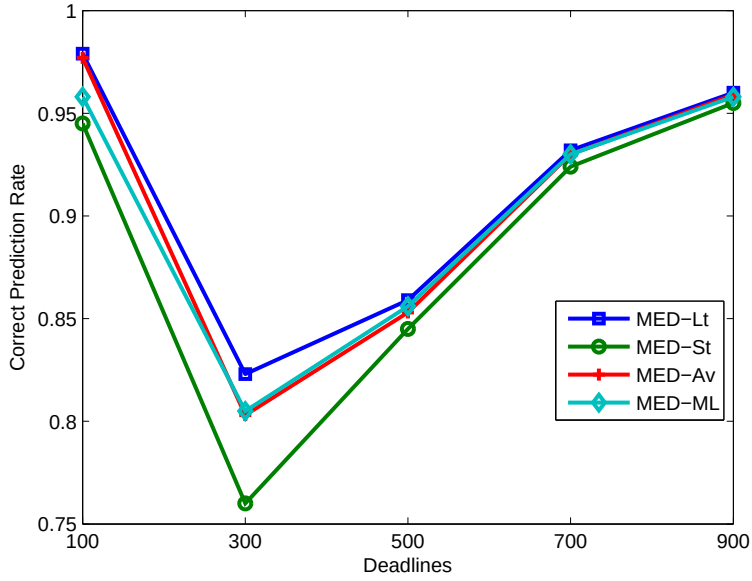

Fig. 10. Correct prediction rate of the MED predictor

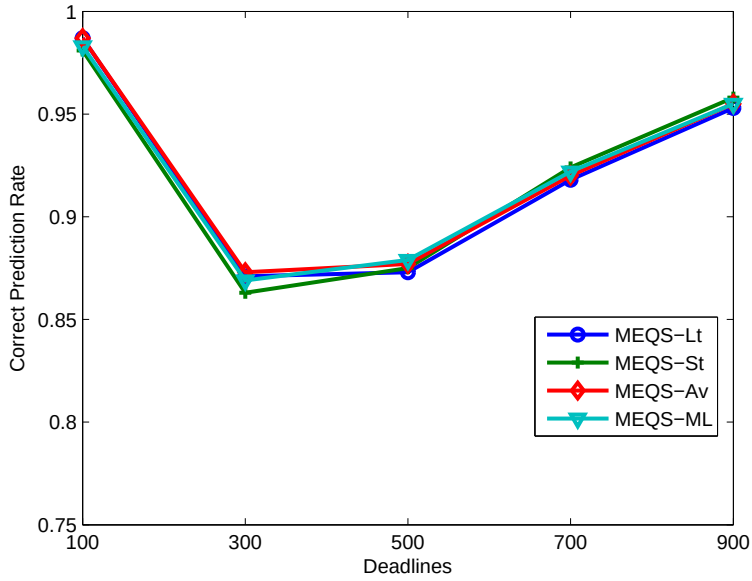

Fig. 11. Correct prediction rate of the MEQS predictor

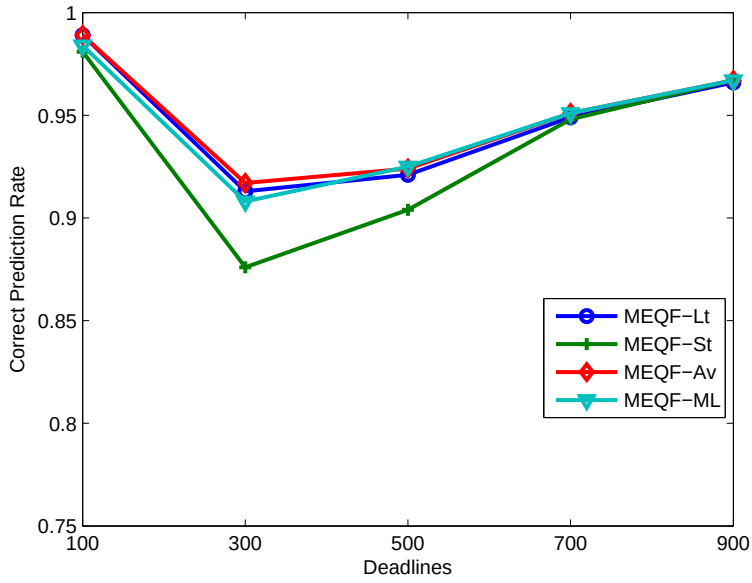

Fig. 12. Correct prediction rate of the MEQF predictor

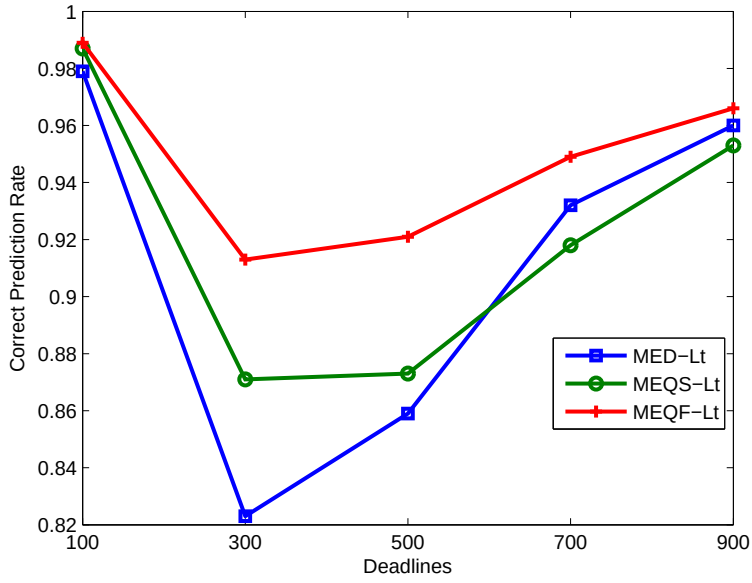

Fig. 13. Correct prediction rate of the MED, MEQS and MEQF mechanisms

We proposed a deadline missing prediction mechanism based on milestones. A milestone is an estimated local response time that is used to define the probability of a distributed thread to meet its deadline. Three milestone predictors – MED, MEQS and MEQF – were defined through the end-to-end deadline partitioning method described in [10].

Simulation results have shown that MEQF presents better results as a milestone generator, when compared with other deadline missing predictors. These results are an improvement of the results found in [12]. It should be noticed that this mechanism is very simple and does not take the dynamics of the distributed system into account, as each node load. Only the estimated computation time of each distributed thread is considered.

As in any analysis through simulation, the conclusions are mainly valid for scenarios similar to those simulated. The set of experiments considered many different scenarios with respect to deadline tightness. Also, by varying the format of the distributed threads, we tried not to draw conclusions from a very limited set of situations. As a general rule, the more variable the behavior of the system, the more difficult is to make any prediction about meeting deadlines. Nonetheless, the comparison between the many heuristics was reasonably stable among the many conditions tested.

In [14] we presented the ASQ deadline prediction mechanism that takes into account the dynamics of the distributed system. Although this mechanism has presented better results than milestone predictors, it is more complex and requires a distributed thread to gather information at each node where it arrives and to carry this information back to the source node. On the other hand, milestone predictors are lighter because all estimated response times are defined before a distributed thread begins its execution. At each node it arrives it is necessary to verify this partial response time and if it is equal to or greater than half of its end-to-end deadline. If it is true the distributed thread computes the probability of meeting its deadline. Milestone predictors are adequate for distributed systems which have restricted computational resources.

The next step of this research is to improve the way deadline missing prediction mechanisms are launched. We intend to develop a launcher that can monitor the system load and to decide the best moment to launch the prediction mechanism. An open question is how to use our simulation program to check the effectivity of the corrective actions that follows the deadline missing prediction. This is a difficult task since the value of the corrective actions is highly dependent of the application semantics.

REFERENCES

[1] Clark, R. K.—Jensen, D.—Reynolds, F. D.: An Architectural Overview of the Alpha Real-Time Distributed Kernel. Winter USENIX Conference, April 1993, pp. 127–146.

[2] Tilevich, E.—Smaragdakis, Y.: Portable and Efficient Distributed Threads for Java. Middleware '04, Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware 2004, Toronto, Canada, pp. 478–490.

[3] Li, P.—Ravindran, B.—Cho, H.—Douglas Jensen, E.: Scheduling Distributable Real-Time Threads in Tempus Middleware. 10th International Conference on Parallel and Distributed Systems, July 2004, New Port Beach (California), pp. 187.

[4] Zhang, Y.—Thrall, B.—Torri, S.—Gill, C.—Lu, C.: A Real-Time Performance Comparison of Distributable Threads and Event Channels. Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications, Symposium, March 2005, San Francisco (California), pp. 497–506.

[5] Tatibana, C. Y.—Montez, C.—de Oliveira, R. S.: Soft Real-Time Task Response Time Prediction in Dynamic Embedded Systems. Proceedings of the 5th IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, Santorini Island, Greece, May 2007, pp. 1–10.

[6] Panagos, E.—Rabinovich, M.: Reducing Escalation-Related Costs in WFMSs-NATO Advanced Study Institute on Workflow Management Systems and Interoperability, Istambul 1997.

[7] Real-Time CORBA Specification, Version 1.2, OMG (Object Management Group), January 2005.

[8] Jun Sun: Fixed-Priority End-to-End Scheduling in Distributed Real-Time Systems. Ph. D. Thesis, University of Illinois at Urbana-Champaign, 1997.

[9] Marinca, D.—Minet, P.—George, L.: Analysis of Deadline Assignment Methods in Distributed Real-Time Systems. Computer Comunications, September 2004, Vol. 24, No. 15, pp. 1412–1423.

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

[11] Plentz, P. D. M.—de Oliveira, R. S.—Montez, C.: Scheduling of the Distributed Thread Abstraction with Timing Constraints using RTSJ. 10th IEEE International Conference on Emerging Technologies and Factory Automation, Catania (Italy), September 2005, pp. 2005.

[12] Plentz, P. D. M.—Montez, C.—Silva de Oliveira, R.: Prediction of End-to-End Deadline Missing in Distributed Threads Systems. Proceedings of the 12th IEEE International Conference on Emerging Technologies and Factory Automation, Patras (Greece), September 2007, pp. 25–32.

[13] Plentz, P. D. M.—Montez, C.—Silva de Oliveira, R.: Deadline Missing Prediction in Systems based on Distributed Threads. IEEE Latin American Robotic Symposium 2008, Salvador (Bahia),October 2008, pp. 190–195.

[14] Plentz, P. D. M.—Montez, C.—Silva de Oliveira, S.: Deadline Missing Predictor Based on Aperiodic Server Queue Length for Distributed Systems. Computer Communications 2008, Vol. 31, No. 17, pp. 4167–4175.

[15] Liu, J. W. S.: Real-Time Systems. Prentice Hall 2000.

[16] di Natale, M.—Stankovic, J. A.: Dynamic End-to-End Guarantees in Distributed Real-Time Systems. Proceedings of Real-Time Systems Symposium, San Juan, Puerto Rico 1994, Vol. 7, No. 9, pp. 216–227.

[17] Jonsson, J.: A Robust Adaptive Metric for Deadline Assignment in Heterogeneous Distributed Real-Time Systems. Proceedings of the 13th International Symposium on Parallel Processing and the 10th Symposium on Parallel and Distributed Processing, IEEE Computer Society, Washington, DC (USA) 1999, pp. 678–687.

[18] Ravindran, B.—Curley, E.—Anderson, J. S.—Douglas Jensen, E.: Best-Effort Real-Time Assurances for Recovering from Distributed Thread Failures in Distributed Real-Time Systems. 10th IEEE International Symposium on Object and Deadline Missing Prediction Through the Use of Milestones, 21st Component-Oriented Real-Time Distributed Computing (ISORC), Santorini Island (Greece), May 2007, pp. 344–353.

Patricia Della Méa Plentz has degrees in Computer Science from the University of Cruz Alta (1999), Masters in Computer Science (2001) and Ph. D. in Electrical Engineering (2008) both from Universidade Federal de Santa Catarina (UFSC). She is currently an Assistant Professor in the Department of Informatics and Statistics (INE) of UFSC. Her main areas of research include distributed threads, partitioning of deadlines and forecast of end-end deadline missing.

Carlos Montez has degrees in Computer Science from the Universdade Federal do Rio de Janeiro (1989), Masters in Computer Science (1995) and Ph. D. in Electrical Engineering (1999) both from Universidade Federal de Santa Catarina (UFSC). He is currently an Assistant Professor in the Department of Automation and Systems of UFSC. His main areas of research are real-time systems, adaptive scheduling and wireless sensor networks.

Rômulo Silva de Oliveira has degrees in Electrical Engineering from Pontificia Universidade Catolica do Rio Grande do Sul (1983), Masters in Computer Science from Universidade Federal do Rio Grande do Sul (1987) and Ph. D. in Electrical Engineering from Universidade Federal de Santa Catarina (1997). He is currently an Associate Professor in the Department of Automation and Systems, Federal University of Santa Catarina. His main topics of interest include real-time systems, scheduling and operating systems.