Deadline Missing Prediction in Systems based on Distributed Threads

Patricia Della Méa Plentz, Carlos Montez, Rômulo Silva de Oliveira
Federal University of Santa Catarina (UFSC)
Programa de Pós-Graduação em Engenharia Elétrica (PGEEL)
88040-900 - Florianópolis - SC - Brazil
plentz,montez,romulo@das.ufsc.br

***Abstract*— The Distributed Real-Time Thread is an emergent concept for distributed real-time systems. Distributed Threads are schedulable entities with an end-to-end deadline that transpose nodes, carrying their scheduling context. Mechanisms for predicting the missing of deadlines are fundamental because corrective actions can be incorporated for improving system quality of service. In this work we propose an end-to-end deadline missing prediction mechanism called Available Remaining Slack (ARS). Simulations show that the proposed prediction mechanism presents good results for improving the overall performance and availability of distributed real-time systems.**

I. 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. According to a commonly used definition [1], a deadline can be hard, soft or firm. A deadline is hard if the failure to meet it is considered to be a fatal fault. In contrast, the late completion of a task that has a soft deadline is undesirable. However, a few misses of soft deadlines do no serious harm; only the system's overall performance becomes poorer. A deadline is firm if the failure to meet it is not considered to be a fatal fault, but its late completion does not has value to the system application. Mechanisms for predicting missing of soft and firm deadlines are fundamental because corrective actions can be carry out for improving the system performance.

Applications for this kind of system can be implemented with Distributed Threads (DTs) [2], which are abstractions of distributed tasks that traverse physical nodes boundaries carrying out remote calls. This abstraction is powerful because it demands low overhead in its execution. DTs can be used to represent control and supervision tasks in factory automation systems. These tasks usually have firm deadlines and they are distributed - visiting several nodes to collect information for analyzing and diagnose purposes. In [3] presents a real-time control system based on RTAI-Linux operating system and developed for coupling of an advanced end-effector. [4] develops a system of robot open control based on a the reference model OSACA. [5] proposes an real-time architecture for robot control system development based in real-time operating system for embedded systems, RTOS.

In the field of autonomous mobile robots [6], a high-level coordination and control system is very important because it allows achieving the desired overall system behavior in a distributed thread framework. Adequate computational architectures may be used to support the effective behaviors execution of autonomous robots, controlling various robotic components and subsystems. These architectures incorporate hardware and software components. In the context of software architecture, the concept of distributed threads can be used to implement applications that manage robotic coordination and control systems.

The aim of this work is to define a deadline missing prediction mechanism for systems based on DTs. This mechanism uses information such as available remaining slack of a DT.

The remaining of the paper is organized as follows: section II presents related work with DTs and response time prediction mechanisms. Section III describes main concepts about DTs and deadline partitioning methods. Section IV presents the proposed model and describes probabilistic known itineraries for DTs. The deadline missing prediction mechanism introduced in this work is presented in section V and its applicability and performance results are shown in section VI. Section VII contains the final remarks.

II. Related Work

Many work in the literature address DT implementations [2], [7], [8]. The seminal work is [2] where the Alpha's kernel programming model is described. That kernel is based on DTs that traverse

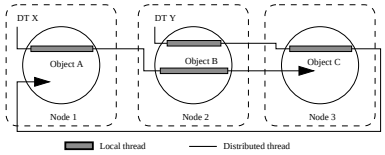

Fig. 1. Distributed threads and local segments.

system's nodes carrying its timing constraints in order to make possible system resource management. In [8] DTs are implemented through Real-Time Specification for Java (RTSJ) and a system architecture is proposed which is flexible enough to accommodate 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 that system architecture by the introduction of the deadline missing prediction mechanisms.

Using local information, such as response times and local deadlines, for example, it is possible to predict the future behavior of the system. In [9] the authors deal with the prediction of response times. That work 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 are used past executions of this service and the historical data about past executions of all the services supported by the program.

III. Distributed Threads

A Distributed Thread (DT) is an end-to-end control flow abstraction with a globally unique identifier that transparently extends and retracts through an arbitrary number of local and remote objects [10]. Usually, a DT is implemented through a sequencing of executions of local threads. Figure 1 shows the execution of two DTs (DT X and DT Y) where it is possible to visualize DTs being composed by local segments of execution. In each node that a DT executes, a local segment of this DT is created, being mapped into a local operating system thread.

This abstraction carries its execution context as it transits node boundaries, including its scheduling parameters (e.g., time constraints, execution time), identity and security credentials. Also, DTs can share physical resources (e.g. processor, disk, I/O) and logical resources (e.g. locks), which can be subject of mutual exclusion constraints.

A DT begins its execution by invoking an object operation. The object and the operation are specified when the DT is created. Its execution may terminate in the same node where it began, or in another node of the system. A DT that is executing can create a new real-time DT (e.g. through one-way invocations). This new real-time DT may begin executing in another node of the system. However, this kind of invocation is not considered in this work. An application consists of multiple DTs executing concurrently and asynchronously and the scheduling of them is carried out by local schedulers.

A. Methods for End-to-End Deadline Partitioning

One of the timing constraints usually imposed in the requirements of a distributed real-time application is an end-to-end deadline. A common practice for local scheduling is to partition this end-to-end deadline in local deadlines, which will be used in a scheduling policy by local schedulers.

In the context of DTs, this partitioning can be carried out considering their local segments, which are adequate points to define local deadlines. The partitioning method can be performed in a static or dynamic way. In the first case, the partitioning occurs before the DT begins its execution. All the local deadlines are defined and do not suffer changes during DT execution. On the other hand, when the partitioning is performed in a dynamic way, the local deadlines are redefined each time the DT arrives at a node.

Several works in the literature propose different methods for partitioning deadlines [11], [12]. In this work, we are interested in methods that do not consider the system load, like those defined in [11] which are described following.

- **Equal Flexibility** (EQF): This method divides the DT's total slack proportionally to execution time among all its local segments. With this method, a DT has more chance of meeting deadlines because all local segments receive local deadlines according to its execution time.
- **Equal Slack** (EQS): This method divides the DT's total slack equally among all its local segments. This method is not as flexible as EQF because it doesn't embody the relation between execution time of the local segments and DT total slack.
- **Effective Deadline** (ED): This is a simple method that considers just the estimated execution time of local segments in the definition of local deadlines. ED assigns all DT slack to the first DT local segment. All other local segments do not receive any slack to execute and have more chances of missing their deadlines.

IV. Proposed Model

In this work we are considering distributed real-time systems like those in factory automation. Only one application executes in all nodes of the system at a given moment. This application is composed by local periodic tasks, which have hard deadlines and by distributed aperiodic tasks with firm deadlines. The distributed aperiodic tasks are represented by DTs. An end-to-end deadline and an estimated execution time are defined when a DT is created. This system has dynamic load because the arrival times of DTs are unknown. Therefore, there may be moments that the system is overloaded. DTs 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.

The system architecture presented in [13] is used in this work (Fig. 2). In each system node there is an aperiodic server responsible for the execution of DT's local segments. In this work it is proposed the use of the Sporadic Server [1] for the scheduling of the DT local segments and the Earliest Deadline First (EDF) algorithm [1] for preemptively scheduling the aperiodic server's queue.

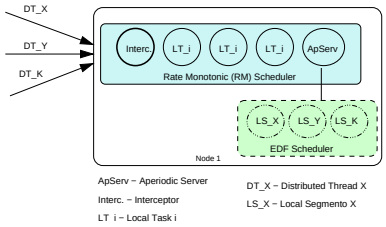

Fig. 2. System architecture.

With this architecture, we intend to guarantee the deadlines of the hard local periodic tasks. Furthermore, it should reduce the response time of the firm distributed aperiodic tasks in order to meet the end-to-end deadlines of the DTs. For these purpose, the scheduling approach used in this work is composed by two stages: partitioning of the end-to-end deadline and local scheduling. This is not an uncommon procedure in the real-time literature. The DT's timing attributes only affects the scheduling, and the local schedulers are considered independent. They do not collaborate with each other in a explicit way. Rate Monotonic (RM) algorithm [1] is used in each system node. It will preemptively schedule the hard periodic local tasks, interceptor and aperiodic server.

The DT's execution flow is defined at runtime while it traverses the system nodes executing remote calls. For this reason, a DT can be considered autonomous and its remote calls itinerary can be recorded. As shown in Figure 3, methods A, B and C are in the nodes 1, 2 and 3, respectively. If a DT begins its execution in node 1, it can go to node 2 (method B) or node 3 (method C), according to application behavior.

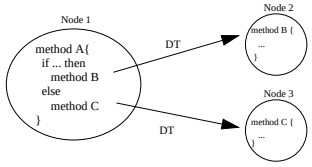

Fig. 3. DT execution flows alternatives.

A. Probabilistic Known Itineraries for Distributed Threads

A DT can follow different itineraries during the execution of remote calls. These itineraries are kept stored in the DT's log. Although a DT is of an autonomous nature and it is difficult to predict the next node that it will traverse, through the observation of its historical data it is possible to define four main itineraries that a DT would follow [13]:

- **Longest Itinerary:** it takes into account the greater path, in number of nodes, that a DT can traverse.
- **Shortest Itinerary:** it takes into account the lesser path that a DT can traverse executing remote calls.
- **Most Likely Itinerary:** it identifies in the DT's log what path was followed more times by a DT.
- **Average:** it searches in the DT's log data the many paths traversed by it and computes an average of these itineraries. The different probabilities of a DT following one or an other path are taking into account in the partitioning of the end-to-end deadline.

Deadline partitioning methods are applied at the creation of each DT activation, before it begins to execute (static way). Method EQF [13] is used in this work to partition deadlines according to some DT itinerary.

Because the DT traverses nodes according to application dynamics, it may not follow this pre-determined itinerary. Considering this situation, all DT's local segments receive a local deadline which is defined before the DT begins its execution. This is done iteratively for each DT' possible itinerary. All DT' possible itineraries can be seen as a task graph, like those showed in the Figure 4 (balanced and non-balanced-like DTs). The DT' local segments that execute the pre-determined itinerary

receive a local deadline, which is defined by EQF deadline partitioning method. The partitioning of the end-to-end deadline is governed by the constraint that the arrival time of a local segment must be equal to the absolute deadline of its immediate predecessor in the task graph.

All remaining DT' local segments that do not belong to the pre-determined itinerary must receive a local deadline, also defined by the EQF method. For this, the arrival time of each remaining local segment is equal to the absolute deadline of the immediate predecessor local segment which belongs to the pre-determined itinerary.

V. Deadline Missing Prediction Mechanisms

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

The choice of an adequate system node to run deadline missing prediction mechanisms is a very important decision. The prediction mechanism proposed in this work runs at the node which represents the middle of the DT execution, that is, the prediction is made when the response time of an ongoing DT is equal to or greater than half of its end-to-end deadline. For example, if the DT end-to-end deadline is 200 units of time - ut, the prediction mechanism is executed at the first node where the DT response time so far is equal to or greater than 100 ut when the local segment finishes.

In this work we propose a deadline missing prediction mechanism. First we briefly describe a mechanism already presented in the literature. It will be used as baseline in the comparison. Next, we present a prediction mechanism based on the DT available remaining slack.

A. Mechanism Based on Milestone

In [13] it is proposed a deadline missing prediction mechanism which uses deadline partitioning methods (ED, EQS and EQF) in order to define target local response times, called milestones. These milestones are used as response time predictors. If a DT finishes a given local segment before its milestone, it is assumed that the end-to-end deadline will probably be met. On the other hand, the missing of a local milestone is an indication that the end-to-end deadline will probably be missed.

Simulation results in that paper show that EQF presents better results as a milestone generator, when compared with other deadline partition algorithms. One should notice that this mechanism is very simple and does not take into account the dynamics of the distributed system such as the load at each node. Only the estimated computation time of each DT is considered.

B. Mechanism Based on Available Remaining Slack (ARS)

We propose in this paper the mechanism based on available remaining slack (ARS). This mechanism uses the execution time of all DT itineraries that it can follow, from the current node where the DT is executing, with the respective probability of each itinerary. It is defined as:

$$P_k(ARS) = (D_k - LResp_k - RExec_k)/RExec_k$$

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

where D_k is the DT's end-to-end deadline, $LResp_k$ is the spent execution time and waiting time by DT_k in all nodes visited up to the moment and $RExec_k$ is the sum of expected execution time of DT_k local segments that still will be executed.

The $RExec_k$ variable considers the execution time of all DT itineraries that it can follow. To carry out this calculation, ARS mechanism consults a dynamic structure, carried and updated by the DT as it transposes the system nodes. This structure has all possible DT itineraries with the respective execution time of each them. This mechanism calculates the amount of execution time that will be spent by each itinerary, considering the probability of each itinerary be followed.

Considering a DT_x of non-balanced type, as showed in Figure 4 (with end-to-end deadline equal to 140ut), there are 2 possible itineraries that it can execute: the first one composed by the nodes 2 and 4, and the second one composed by the node 3. The first itinerary has a probability of 60% to be executed by DT_x, and the second one has 40%. If it is supposed that execution time of the first and the second itinerary is, respectively, equal to 50ut and 30ut, the value of $RExec_x$ is defined as:

$$RExec_x = (50 \times 0,6) + (30 \times 0,4)$$
$$RExec_x = 42$$

Supposing that the response time of DT_x' at node 1 is equal to 70ut. The ARS mechanism is activated at this node because the DT_x reached half of its end-to-end deadline. The probability of this DT to meet its deadline is defined by the ARS in the following way:

$$P_x(ARS) = (140 - 70 - 42) \div 42$$
$$Prob_x(ARS) = 0,6$$

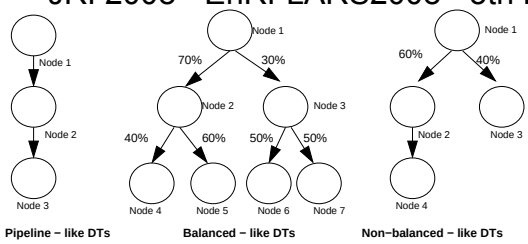

Fig. 4. Types of Distributed Threads.

VI. Simulations

Simulations were carried out with the objective of comparing the performance of the ARS mechanism with Milestone mechanism proposed in [13]. 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 [9]:

$$E_k(z) = 1 - Prob_k(z) \qquad case \qquad R_k \leq D_k$$
$$E_k(z) = Prob_k(z) \qquad case \qquad R_k > D_k$$

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 50% and if the DT met or missed its deadline. Formally, CP(z) metric is defined as:

$$If \;\; (Prob_k(z) < 50\%) \; and \; (R_k > D_k) \; then$$
$$CorrectPrediction(z) += 1;$$
$$If \;\; (Prob_k(z) \geq 50\%) \; and \; (R_k \leq D_k) \; then$$
$$CorrectPrediction(z) += 1;$$
$$CP(z) = CorrectPrediction(z) \div NPredictions(z);$$

where $NPredictions(z)$ is the total amount of 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 bellow.

A. Simulation Conditions

In this system there are 16 interconnected nodes, in each node there are four hard periodic local tasks, whose periods are 10ut, 20ut, 40ut and 80ut. 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 5ut and its period is 10ut.

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 (Fig. 4). It was generated 100 different configurations, each configuration is composed by 9 DTs randomly chosen from the DT'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 type. The average executing time of each DT local segment varies from 5ut to 200ut.

In each configuration, 5 different values of deadlines were used in the range from 100ut to 900ut where 100ut represents a tight deadline, 300ut, 500ut and 700ut represent fair deadlines and 900ut represents a loose deadline. The DTs arrival time rate follows an exponential distribution with an average of 700ut between arrivals. Furthermore, communication network delay has uniform distribution between 1ut and 2ut and it is assumed that there is no network partition. The simulation time was equal to 20000ut for each configuration.

B. 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:

- MED-Lt, MED-St, MED-Av and MED-ML;
- MEQS-Lt, MEQS-St, MEQS-Av and MEQS-ML;
- MEQF-Lt, MEQF-St, MEQF-Av and MEQF-ML.

Figure 5 shows the results of the MED, MEQS, MEQF and ARS predictors, through the Relative Error Rate (E(z)) metric. Because of Longest itinerary showed better results that the other itineraries, it will be plotted in the graphic, with each *Milestone* predictor.

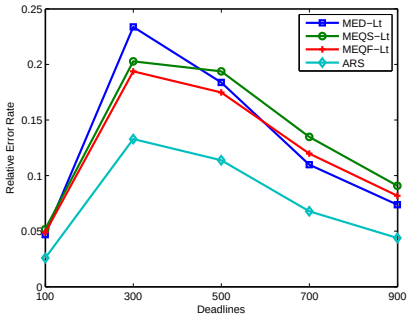

Fig. 5. Relative Error Rate.

It is possible to verify that with all deadlines, the ARS prediction mechanism presents better results than the milestone predictors. These results show that if the mechanism considers all possible itineraries that a DT can follow in the deadline

missing prediction, the calculation of the probability is more accurate.

Figure 6 shows the results of the milestones and ARS predictors, through the Correct Prediction Rate (CP(z)) metric. It is possible to visualize that predictor MEQF presents better results than the ARS predictor, in all deadlines. That is because this metric is more simple than the E(z) metric, it considers only if the probability of each mechanism is greater or less than 50% and if the DT met or no its deadline.

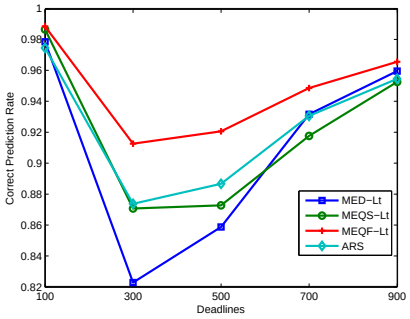

Fig. 6. Correct Prediction Rate.

Through these results, it is possible to say that MEQF predictor is better than ARS predictor if we are interested to know if a DT will meet its deadline and we are not interested to know how was the prediction value, that is to say, what distance that separates the resulting prediction of an exact prediction.

VII. Conclusions

The choice of an adequate deadline missing prediction mechanism has a strong impact in the system performance if remedial actions are to be done. The contribution of this work is the proposing of a deadline missing prediction mechanism that works well when system load is balanced and heavy.

Distributed threads with probabilistic knowledge about their remote calls itineraries has been used in this work to implement a distributed real-time application. 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.

In the performance comparisons, it is possible to say that ARS is better than MEQF because ARS defines better probability values and the results of the E(z) metric confirm this assumption. The predictions of the ARS are more accurate because it uses information about all possible itineraries that a DT can follow. The next step is to elaborate a mechanism that can provide better results than ARS, through the use of system information, like length and composition of the aperiodic server queue of each node that belongs to DT's itineraries.

This approach can be applied to construct distributed applications in many domains, in particular in the robotic field (industrial manipulators or autonomous robots). Ongoing work spreads on the implementation of networked robotic controller including distributed supervisors and distributed sensor systems.

References

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

[2] R. K. Clark, E. D. Jensen, and F. D. Reynolds, "An architectural overview of the alpha real-time distributed kernel," in *Winter USENIX Conference*, April 1993, pp. 127–146.

[3] A. Macchelli and C. Melchiorri, "Real time control system for industrial robots and control applications based on real time linux," *15th IFAC World Congress, Barcelona, Spain, July*, pp. 21–26, 2002.

[4] K. Hong, K. Choi, J. Kim, and S. Lee, "A pc-based open robot control system: Pc-orc," *Robotics and Computer Integrated Manufacturing*, vol. 17, no. 4, pp. 355–365, 2001.

[5] B. Bona, M. Indri, and N. Smaldone, "Open system real time architecture and software design for robotcontrol," *Advanced Intelligent Mechatronics, 2001. Proceedings. 2001 IEEE/ASME International Conference on*, vol. 1, 2001.

[6] J. Azevedo, B. Cunha, and L. Almeida, "Hierarchical distributed architectures for autonomous mobile robots: a case study," *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*, 2007.

[7] E. Tilevich and Y. Smaragdakis, "Portable and efficient distributed threads for java," in *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*. New York, NY, USA: Springer-Verlag New York, Inc., 2004, pp. 478–492.

[8] P. D. M. Plentz, R. S. de Oliveira, and C. Montez, "Scheduling of the distributed thread abstraction with timing constraints using rtsj," in *10th IEEE International Conference on Emerging Technologies and Factory Automation*, Catania(Italy), September 2005, pp. 23–30.

[9] C. Y. Tatibana, C. Montez, and R. S. de Oliveira, "Soft real-time task response time prediction in dynamic embedded systems," in *Proceedings of the 5th IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems*. Santorini Island, Greece: Lecture Notes in Computer Science, May 2007, pp. 1–10.

[10] P. Li, B. Ravindran, H. Cho, and E. D. Jensen, "Scheduling distributable real-time threads in tempus middleware," in *10th International Conference on Parallel and Distributed Systems*, New Port Beach(California), July 2004, p. 187.

[11] B. Kao and H. Garcia-Molina, "Deadline assignment in a distributed soft real-time system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 12, pp. 1268–1274, December 1997.

[12] D. Marinca, P. Minet, and L. George, "Analysis of deadline assignment methods in distributed real-time systems," *Computer Comunications*, vol. 27, no. 15, pp. 1412–1423, September 2004.

[13] P. D. M. Plentz, C. Montez, and R. S. de Oliveira, "Prediction of end-to-end deadline missing in distributed threads systems," in *Proceedings of the 12th IEEE International Conference on Emerging Technologies and Factory Automation*, Patras(Greece), September 2007, pp. 25–32.