Availability Enhancement and Analysis for Mixed-Criticality Systems on Multi-core

***Abstract*—In the critical systems domain, Mixed Criticality Systems (MCS) improve considerably the usage of computation resources by running tasks with different levels of criticality on multi-core processors. To ensure the safety of MCS, services provided by low criticality tasks are degraded or stopped whenever high criticality tasks need more computation time than initially credited. The evaluation of this degradation is hardly considered in the literature although low criticality services are of prime importance for the quality of service (QoS) of critical systems.**

In this paper, we propose a method to evaluate the availability of low criticality services, *i.e.* how often these services are delivered in a MCS. We also propose a task model that improves this availability, demonstrated thanks to our evaluation method on an illustrative example of MCS.

I. Introduction

Most critical systems are composed of tasks with different levels of criticality: high criticality tasks ensure the safety of the system, while low criticality tasks provide additional services with low impact on safety. These systems are monitored and controlled by real-time software applications to guarantee that high criticality tasks meet predefined deadlines even under pessimistic assumptions on their execution time. On the other hand, recent works on Mixed-Criticality Systems (MCS) advocate for resource sharing between low and high criticality tasks to improve performance while ensuring safety [1]. This improvement is more consequent on multi-core architectures thanks to parallelism of tasks' execution.

With MCS, tasks can be executed in different execution modes: in the nominal mode, high and low criticality tasks are both executed with an optimistic timing budget. When the system detects a timing failure event (TFE), *i.e.* a task did not complete its execution within its optimistic timing budget, the system switches to a degraded mode. In this mode, high criticality task are executed with their pessimistic timing budget, discarding [2] low criticality tasks or degrading them [3] (*i.e.* reducing their execution frequency). This difference in timing budgets is introduced due to the multi-core processor non-determinism: execution time of tasks greatly varies in function of the state of the processor (*i.e.* caches, memory bus, *etc*).

Even if low criticality tasks have low or no impact on the system safety, their execution is important for the system's usability. A satellite is an example of critical system with different execution modes. When the satellite has a major failure it enters a survival mode where the solar panels are facing the Sun to save energy and its software applications execute maintenance operations in order to restart the remaining services. In this degraded mode, high criticality tasks ensure that the satellite remains in orbit and does not get lost. However, the low critical tasks (*e.g.* sending communication signals) are what is really useful and interesting in the satellite.

In this context, MCS raise an important problem: in order to ensure the schedulability of high criticality tasks, MCS degrade the availability of low criticality ones, where availability is defined as the ratio of completed against attempted executions of a task. Indeed most works on MCS ignore the **resilience** of the system to transient timing failures. In the seminal work of MCS [2], when a TFE occurs in a low criticality task, all the low criticality tasks are discarded even if the TFE does not impact high criticality tasks. In a recent study, authors of [4] propose a degree of *allowance* to mitigate small delays on tasks' execution. Other approaches advocate for the degradation of low criticality tasks by increasing their periods and/or reducing timing budgets on the high criticality mode [3]. The evaluation of these degradations in terms of availability of low criticality tasks is not considered by these works. In addition, MCS also neglect the fact that real-time systems are often composed of tasks with *weakly hard* real-time constraints: a number of deadline misses is allowed for a number of consecutive executions [5].

In this paper, we propose a Mixed-Criticality (MC) model to improve the availability of low criticality tasks. We also propose a method and tool to evaluate availability of these tasks. The remainder of this paper is structured as follows: our task and fault models are presented in Section II. Enhancements for the availability of low criticality services are described in Section III. Section IV presents the method used to compute the availability rates of MCS. Our contribution is evaluated in Section V. We discuss related works on Section VI and conclude in Section VII.

II. Task and Fault Model

In this section, we present the task model we rely on in order to improve and analyse the availability of low criticality tasks of a MCS deployed on multi-core architectures. We present the task model and its timing properties (subsection II-A) and a model for fault sources (subsection II-B).

A. Task Model

We consider MCS with two operational modes noted HI (high criticality) and LO (low criticality). When the system is in LO mode (initial mode), all HI and LO tasks can be executed on the platform until their optimistic timing budget in LO mode (noted $C_i(LO)$ for a task τ_i). For each task τ_i, $C_i(LO) \leq C_i(HI)$ ($C_i(HI)$ is the timing budget of τ_i in HI

mode). A TFE occurs when a task τ_i runs for its $C_i(LO)$ but has not produced its results.

To represent parallelism and execution dependencies among tasks we consider applications of our MCS are modeled by Directed Acyclic Graphs (DAG). Our model, noted MC-DAG, is composed of tasks represented by vertices in the graph. Each task τ_i is characterized by a criticality level $\chi_i \in$ LO, HI, a timing budget in LO mode $C_i(LO)$ and a timing budget in HI mode $C_i(HI)$ ($C_i(HI) = 0$ if $\chi_i \in LO$). Edges in the DAG materialize precedence constraints among tasks.

Some tasks of the DAG are identified as output tasks: they produce the outputs of the DAG. We consider a service is delivered when the corresponding output task of the DAG has finished its execution. We note LO outputs the outputs produced by LO tasks. Thus, computing the availability of low criticality services (noted LO services) boils to compute the availability of LO outputs.

B. Modeling fault sources

To evaluate the availability of low criticality services in MCS, we need to quantify the occurrence of TFEs in terms of probabilities and time. In our study, we assume that the probability of a TFE are provided: for each task τ_i, a TFE probability p_i is given. Measured Based Time Analysis (MBTA) can be used to obtain a distribution of the execution time for a task, as it is shown in [6]. This technique, among others ([7], [8]) can be used to compute the probability of a TFE. When no fault resilience is considered for low criticality tasks of the MCS, (*i.e.* considering MCS as defined in the seminal work on this topic [2]) obtaining an availability rate for low criticality tasks could be straightforward. Considering the TFE probabilities of tasks and a predefined termination order of these tasks, the availability $A(\tau_i)$ of a task τ_i is defined by the following formula, where $pred(\tau_i)$ is the set of tasks terminating before τ_i:

$$A(\tau_i) = 1 - (p_{\tau_i} + \Sigma_{\tau_j \in pred(\tau_i)} p_{\tau_j}). \quad (1)$$

This statement advocates for the use of static scheduling tables in order to have a unique execution order of tasks. Indeed, with a dynamic scheduling algorithm on a multi-core architecture the number of execution orders among tasks grows rapidly and makes the availability computation impractical. For this reason, we limit the scope of our study to static scheduling algorithms: tasks are dispatched according to predefined scheduling tables (one for each execution mode). Each task τ_i is dispatched in predefined windows with a start date and end date. If task τ_i is able to complete its execution before the end of its time window, the remaining of the window is not re-allocated, *i.e.* the core executing τ_i would be idle until the end of the time window. Static tables also justify the adoption of the discard approach for LO tasks: varying periods and budgets for LO tasks would require to compute a large number of scheduling tables. We assume the static scheduling tables are also given as an input for our method. Results presented in [9], [10] can be used to obtain such scheduling tables.

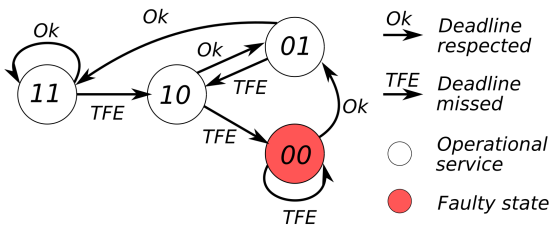


Fig. 1: $(1-2)$-firm task representation

Even under these hypothesis, computing the availability of LO services is not trivial: the resilience of tasks to transient timing failures has to be considered. In addition, faults propagation must be controlled in order to enhance availability of LO services. For this reason, we propose the fault propagation model described in the next section.

III. Enhancing availability on LO tasks

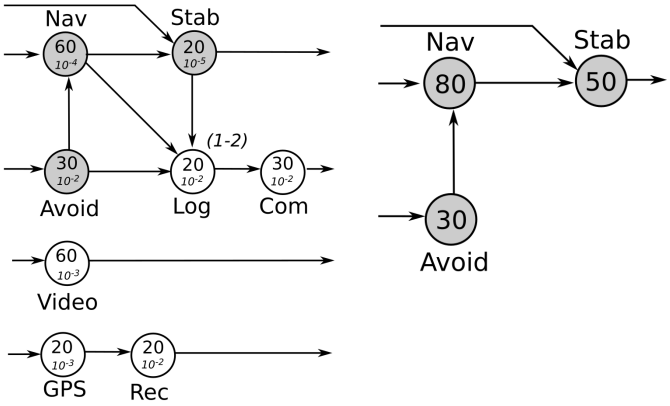
To enhance the availability of LO services we propose to apply three strategies: 1) As long as they appear in LO tasks, **TFE are only propagated to compromised tasks in the MC-DAG**. 2) A recovery mechanism reintroduces LO tasks in the next iteration of the scheduling table, so the scheduler always (re)starts in LO mode. 3) Resilience of tasks to TFE is modeled using **weakly hard real-time constraints**. These strategies are presented in subsection III-A. We illustrate them on a simple example in subsection III-B.

A. Modeling faults propagation

When a TFE occurs in a HI task, the system switches to the HI mode. LO tasks are canceled and the timing budget of HI tasks is extended to complete their execution.

In our contribution we are interested by providing a degree of allowance on the system and mitigate faults that occur in LO tasks. For this reason, when a TFE occurs in a LO task, we propose to limit its propagation in the MCS. Instead of performing a mode transition to the HI criticality mode, only the faulty task is canceled and its successors in the MC-DAG are not executed. Since the application is modeled by a DAG, we are able to determine which LO tasks are affected by the TFE: we know which part of the graph will not be able to execute. As opposed to existing works on MCS, tasks that do not depend on the failing task are allowed to complete their execution.

To enhance even further the availability of LO services, we model the resilience of real-time systems to transient timing failures thanks to weakly hard real-time tasks [5]. These are often modeled with a $(m-k)$ firm constraint: TFEs can occur without interrupting the services as long as m out of k consecutive executions do not miss their deadlines. Fig. 1 illustrates on a state machine the behavior of a $(1-2)$ firm task: out of two consecutive executions, at least one needs to respect the deadline to consider the service can rely on produced or existing results. On this figure, the history of two consecutive executions is represented with a binary encoding for each state. Correct executions are marked with a 1, and deadline misses are marked with a 0: 11 means that both the



(a) LO Mode (b) HI Mode

Fig. 2: Architecture example

i^{th} and $(i+1)^{th}$ iterations were non-faulty while 01 means that the i^{th} iteration was faulty while the $(i+1)^{th}$ iteration was non-faulty. The status of task executions are represented by transitions among states. In the remainder of this paper, LO tasks enhanced with $(m-k)$-firm constraints are referred as **LO* tasks**.

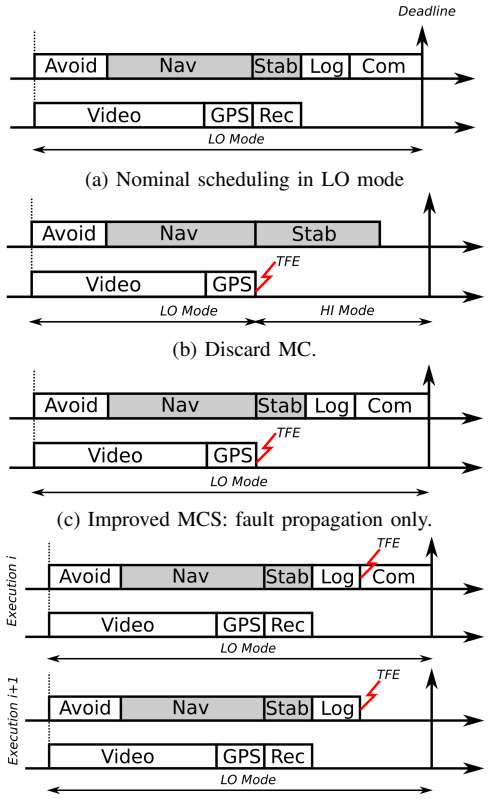
If the system switched to HI mode, the recovery of LO tasks is performed as follows: once the system has finished executing the scheduling table in HI mode, at the next execution of the scheduling table, the system goes back to LO mode.

B. Illustrative example

An example of a MC-DAG is illustrated in Fig. 2. The system represents an Unmanned Air Vehicle (UAV). HI tasks are illustrated by gray vertices while LO tasks are the white vertices. Each node is annotated with its timing budget for the LO and HI modes. In Fig. 2a the system in LO mode is represented, for example task GPS has a $C_i(LO) = 20\ TU$ with a TFE probability of 10^{-3}. Some of these tasks can also be $(m-k)$ firm: task Log in the example is a $(1-2)$ firm as identified by its label. The HI mode is illustrated by Fig. 2b.

The navigation system of the UAV is composed of tasks $Avoid$ (Avoidance), Nav (Navigation) and $Stab$ (Stability) which are HI tasks. These ensure that the UAV will not crash. Nonetheless, if the UAV is used on an exploration mission, what is interesting for the end users is the record Rec of the GPS coordinates and the $Video$ transmissions. These services (Rec and $Video$ are outputs of the graph) are independent: it would be interesting to deliver them whenever it is possible.

The following scenarii show how the MCS reacts to TFEs that occur on different tasks of the MCS in Fig. 2. For space limitation we assume there is only one DAG in the MCS with a single period/deadline. As long as scheduling tables are provided the analysis remains the same even with various DAGs and different periods/deadlines, since we consider the hyper-period for the analysis. Fig. 3a shows the nominal scheduling tables of the MC-DAG in LO mode. When a TFE occurs on task GPS in a discard MC model, a mode transition occurs, as shown in Fig. 3b. A TFE in the GPS task provokes



(a) Nominal scheduling in LO mode

(b) Discard MC.

(c) Improved MCS: fault propagation only.

(d) $(1-2)$ firm: mode transition in the second execution.

Fig. 3: Examples of TFE on different tasks.

a mode switch and discards the execution of the Log and Com tasks. However, there is no dependency between the GPS task on the one hand, and the Log or Com task on the other hand. With our fault propagation model (Fig. 3c) this is no longer the case, and tasks Log and Com can complete their execution if a TFE occurs in the GPS task. Finally, Fig. 3d shows how the system is able to tolerate a TFE on the Log task if it is $(1-2)$ firm: two consecutive faults on the task need to take place to discard its successors.

As shown on these scenarii, we expect our fault propagation model to enhance the availability of LO services in MCS. To determine if these enhancements are significant in terms of availability, we propose a method to compute availability in the next section.

IV. Computing Availability Rates

We consider that services of the MC-DAG are delivered when output tasks are executed. We thus focus on the availability of LO outputs of the MC-DAG. To compute this availability, when LO* tasks are deployed in the MCS, Eq. 1 does not hold: the availability analysis is too complex due to the behavior of LO* tasks. By using Eq. 2, the availability value can obtained by simulating the system and counting the number of times the output was produced, divided by the total executions of the graphs in LO mode (LO_{exec}) and HI mode (HI_{exec}).

$$A(out_i) = Num\_prod_i/(LO_{exec} + HI_{exec}). \quad (2)$$

We propose a method to compute the availability of LO outputs by translating the MC-DAG and scheduling tables into probabilistic automata and evaluation formulas used by the PRISM simulator [11]. In this section we first present an overview of probabilistic automata (PA) used in our translation (subsection IV-A). We then describe the algorithm to translate scheduling tables to PA (subsection IV-B).

A. Probabilistic automata in PRISM

PRISM is a probabilistic model checker capable of analyzing systems that exhibit complex probabilistic behaviors. It has been used across many fields with great success: security and embedded systems are some examples. Our fault model with TFEs being probabilistic justifies the utilization of this tool to perform our availability analysis. A PRISM model is made of a set of PA, and each automaton is characterized by: *states* to represent tasks' execution, *local variables* for our fault propagation model, *probabilistic transitions* to represent TFEs, *rewards* to count output productions and *deterministic transitions* to incorporate our recovery mechanism.

B. Probabilistic Automaton for Scheduling Tables

The idea behind the translation of scheduling tables to a PA is to represent the state of the processor, *i.e.* which task is running and in which mode.

While the scheduling tables are computed for a multi-core architecture, we check if all tasks completed their execution sequentially. Having parallel automata in PRISM increases significantly the simulation time due to the large number of possible states for the model. Since all tasks need to be checked, doing it in a sequential manner does not change the final result but significantly reduces the simulation time.

To check completion of tasks, we sort the tasks of the MC-DAG by their end of time window in the LO scheduling table, and create an ordered list of tasks in LO mode. For each task τ_i in this list (where i is the index of this task in the list) we create a state L_i and a boolean variable b_i. Due to parallelism obtainable thanks to multi-core processors, tasks can have the same end of time window. This has to be taken into consideration for the availability computation since mode transitions can be triggered by HI tasks but LO outputs can be available at the same time slot. Ties in the list need to be broken with the following rules: 1) LO tasks that are exit nodes of the graph go first: we need to know if their output is available. 2) HI tasks go afterwards, a mode transition occurs when they have a TFE. 3) Other LO tasks are checked at the end, no mode transition occurs even if they fail.

After generating the states representing the execution of tasks in LO mode, we connect them with the following rules:

1) From a state L_i: if it represents a LO task, we add to next state L_{i+1}. One representing the occurrence of a TFE (with a probability p_i) and updates b_i to false, whereas the other (with a probability $1 - p_i$) updates b_i to true.

2) If L_i is a LO* task, we add an intermediary state L_i^* and a set of k variables to represent the execution history of k executions of τ_i. We connect L_i^* to L_i with two probabilistic transitions. In each transition, the execution history is updated to account for a new correct or incorrect execution of τ_i. L_i^* is then connected to L_{i+1} thanks to two deterministic transitions. The guard of the first transition checks if at least m out of k executions of the LO* task were correct. If that is the case boolean, b_i is set to true. The second transition represents a failure of the LO* task (*i.e.* less than m out of k executions were correct), and boolean b_i is set to false.

3) If L_i is a LO output task, we also need an intermediary state, noted L_i^{out}, to check if all the predecessors of L_i executed normally. We also add a reward out_i to count for the number of cycles where the output is produced (used for Eq. 2). Then, L_i is connected to L_i^{out} with two probabilistic transitions. One with a probability p_i updates b_i to false, the other with a probability $1 - p_i$ updates b_i to true. L_i^{out} is then linked to L_{i+1} with two deterministic transitions. The first one represents the production of the output (*i.e.* service delivery) and is guarded by a boolean conjunction of the boolean variables representing the execution of τ_i and all its predecessors in the MC-DAG. When firing this transition, the reward out_i is incremented. For the output Rec, this transition is guarded by $b_{GPS} \wedge b_{Rec}$ and reward out_{Rec} is incremented. The second transition represents the non production of the output (*i.e.* the conjunction is false).

4) If L_i is a HI task, we add a probabilistic transition from L_i to L_{i+1} with probability $1-p_i$. This represents τ_i executing within its timing budget. Since L_i is a HI task, the system switches to HI mode in case of TFE. We explain how the execution in HI mode is represented in the generated PA in the remainder of this subsection.

Note that if L_i is a HI task and an output task, a combination of rules 1 and 3 is applied. This is not detailed for space limitation reasons. Besides, when L_i is the last state of the list, L_{i+1} becomes L_0: representing the cyclic execution of the MC-DAG.

To translate the scheduling table in HI mode, we also sort the nodes of the MC-DAG by their end of time window in the HI scheduling table. For each task τ_i in the ordered list we create a state H_i in the PA we generate. H_i represents the execution of τ_i in HI mode. To represent a mode switch in case of TFE in a HI task τ_i, we add a probabilistic transition from L_i to H_i with a probability of p_i (probability of TFE in τ_i). We then connect each state H_i with its successor in the list using deterministic transitions. Note here that if H_i is the last state of the list, it is actually connected to L_0 with a deterministic transition: this represents the recovery mechanism.

Fig. 4 illustrates the PA obtained by our translation algorithm with the MC-DAG of Fig. 2. Dotted lines represent probabilistic transitions. As we can see they are annotated with TFE probabilities: p_{τ_i}. Intermediary states to account for the production of LO outputs are $L_{Video}^{out}, L_{Rec}^{out}$ and L_{Com}^{out} for tasks $Video, Rec$ and Com. Deterministic transitions come from these states (full lines in the figure) and are guarded by the boolean conjunctions of the output's predecessors. L_{Log}^* is the intermediary state introduced in order to test the $(m-k)$ firm behavior of task Log.

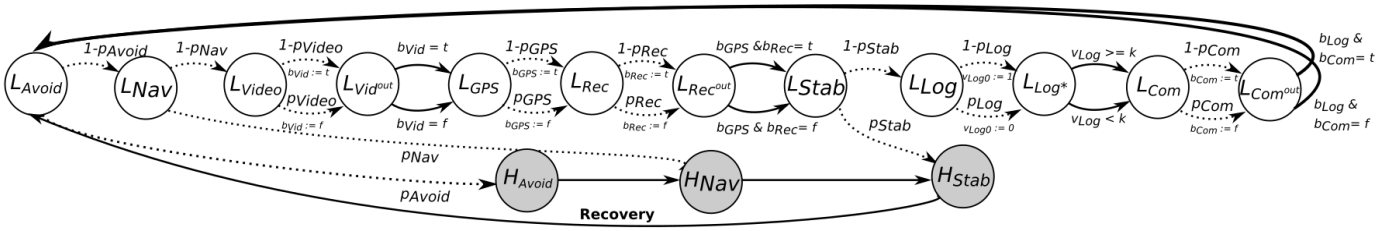


Fig. 4: Translation of the UAV system

Once the PA is generated, we use a reward operator (noted R in PRISM) to compute the cumulative value, for a number of simulation steps (each step triggers one transition for the automaton), of rewards associated to outputs of the MC-DAG. A similar mechanism is used to compute the number of executions of the scheduling tables in LO and HI mode. This is finally translated in a formula to compute availability of outputs as expressed in Eq. 2.

We proceed to test our translation algorithm in the next section where we consider different architectures to compute their availability rate.

V. Experimental Results and Discussion

An implementation of our translation of MC-DAG to PRISM automata is available as an open source project[1]. We present in this section the results we obtained when using this tool to analyse the availability of low criticality services of the UAV system illustrated on Fig. 2. We also present the threats to validity of these results.

A. Experimental setup and results

We perform the availability analysis under three different execution models : (i) the discard MC one (a TFE triggers a mode transition), (ii) our enhanced MC model and (iii) our enhancement enriched with LO* tasks.

We compute the availability rates for services $Video$, Rec and Com thanks to our tool. We set the maximum number of steps simulated in PRISM: each step triggers one transition of the automaton. We simulated the execution of our system for $5 * 10^6$ steps to obtain a stable value of availability with a precision of 10^{-7}. Such precision is typically required when a system is supposed to be used for years: in this case, a difference of availability at scale 10^{-7} can make a difference. For instance, an avionics function at level B of criticality (second most critical) must have a probability of failure of 10^{-7} per hour of flight (and 10^{-5} for level C). In our experiments, the number of steps we used to reach this precision was set experimentally, observing the stabilization of the availability value during the simulation of the model.

Table I provides the results obtained when computing availability rates of outputs of the UAV example. The availability values of outputs Rec and Com are improved when we consider allowance on the MCS. The first improvement can be seen when our enhanced mode transition condition is used: Rec and Com are more available than when discard MC model is considered. The fact that mode switches are not triggered at every TFE and that faults in LO tasks are only propagated to dependent tasks has a positive impact on availability. This is improved even further for the Com output when the Log task is a LO* task with a $(1-2)$ firm behavior: the task is capable of mitigating some TFEs that can occur. We observe a significant gain of availability for the Com service: more than 2% when considering Log as an $(1-2)$ firm task, and more than 1% otherwise. This is a very significant difference for a critical system, specially when a precision of 10^{-7} is required.

TABLE I: Experimental results

	Availability Rate		
Outputs	Discard MC	Enhanced MC	Enhanced MC w/ WHRT
$Video$	98.90099%	98.90099%	98.90099%
Rec	97.80331%	97.90121%	97.90121%
Com	95.85703%	97.01922%	97.98941%

These experimental results show the efficiency of our method, as well as the effectiveness of the fault propagation model we propose: it significantly improves the availability of low criticality services in critical systems. We discuss the threats to validity of these results in next subsection.

B. Threats to Validity

The threats to validity of our experimental results are formulated as follows: does the method scale for more complex systems? Is the availability improvement measured on the example repeatable?

In terms of complexity, we expect the scalability of our method to be influenced by the complexity of the input model (*i.e.* number of tasks, number of LO* tasks, number of outputs, etc.) and the required precision of the availability: the number of simulation steps increases when high precision is required, but also when tasks have a high probability of failure. In order to evaluate the scalability of our approach with regards to the complexity of the input model, we measure the time required to obtain the availability of low criticality services in more and more complex examples.

For the UAV example of Fig. 2, computing the availability rate for each LO output after $5 * 10^6$ steps took 10-20 seconds with an Intel®Core™ i7-5600U CPU @ 2.60GHz. We produced more complex examples by replicating the UAV of Fig 2 up to 5 times in a single MC-DAG. The computation of our availability rate grows rapidly: when 3 replicas of the DAG are considered the computation of an availability rate takes up to 4 hours. For 5 replicas the computation of the availability takes up to 26 hours. This model is made up of 40

[1]https://github.com/robertoxmed/ls_mxc

tasks (20 LO tasks, 5 LO* tasks, 15 HI tasks) and 20 outputs (5 HI, 15 LO), and failure probabilities vary from 10^{-5} to 10^{-2} (see Fig 2). We obtained an availability value with a precision of 10^{-5}, which is quite high for low criticality services.

We believe that this example is representative of a complex system with pessimistic hypothesis for our method (high precision required with high probabilities of TFE: 10^{-2} to 10^{-3} for most tasks). In this context, 26 hours is still a acceptable for an early validation of the QoS of a complex and critical system. If quicker results must be obtained, for instance to compare architectures at early stages of the development process, the precision can be set to smaller values. For example, with 4 replicas of the UAV MC-DAG, TFE probabilities of 10^{-4} for every task, and a precision of 10^{-5}, we could obtain availability of outputs in 1 hour of simulation ($2 * 10^{-5}$ steps).

In terms of repeatability, the threat to validity concerns the gain of availability measured in our experiments. We have a significant gain of availability for the Com service on the UAV example. On the one hand, we expect this gain to be smaller with more realistic probabilities of TFE. On the other hand, we expect this gain to grow if more LO tasks are added to the MC-DAG: if a discard MC model is adopted additional TFE points are introduced to the scheduler. To evaluate these statements, we performed new experiments and measured the improvement of availability for the Com service when using our approach versus discard MC model. First, using the UAV example with TFE probabilities to 10^{-4} for all tasks, the gain was of 0.04% when considering Log as an $(1-2)$ firm task, and 0.03% otherwise. Second, we increase the number of LO tasks by considering 4 replicas of the UAV, the gain was of 0.2% when considering Log as an $(1-2)$ firm task, and 0.16% otherwise. Again, all these gains a very significant when a precision of 10^{-5} or $10-7$ is required for a low criticality service, confirming our expectations and showing the repeatability of our results.

VI. Related Works

Taking into account QoS for LO tasks has recently been tackled in the MC domain. The scheduling approach presented in [3] proposes to change periods of LO criticality tasks in order to have a minimum service of LO tasks: the period is incremented to have less activations of LO tasks but they are still executed in the HI mode. Adaptations to the discard MC model have also been proposed [12], where LO tasks also have a $C_i(HI)$ smaller than their $C_i(LO)$. A scheduling algorithm is proposed in this contribution and proved to be dominant over other studies considering the same task model. Nonetheless these works consider independent task sets and mono-core architectures. Improving the QoS for LO tasks thanks to fault-tolerance for MCS has been studied in [13]. Authors consider a degraded LO mode when overruns and transient faults occur the system, allowing the execution of LO tasks even after failures. However, none of these works proposes a method to measure the improvements on the QoS in terms of availability of low criticality services.

A method to compute an availability of LO tasks in MCS was proposed in [14]. To our knowledge this is the only existing contribution considering an availability analysis for MCS with task precedence constraints. Nonetheless, no enhancement of the discard MC model is proposed in this work, and only mono-core architectures are considered.

VII. Conclusion

This paper presents two contributions: a method to compute the availability of low criticality services in mixed criticality systems, and a model to improve this availability. Experimental results show the usability of our method, as well as the enhancement of this availability with the model we proposed. Last but not least, an open-source implementation of the proposed approach is available.

In the future, we plan to extend this work to support N levels of criticality. We also plan to work on source code generation to automate the deployment of software applications modeled with the approach we proposed in this paper.

Acknowledgment

This research work has been funded by the academic and research chair Engineering of Complex Systems.

References

[1] A. Burns and R. Davis, "Mixed criticality systems-a review," *Department of Computer Science, University of York, Tech. Rep*, 2013.

[2] S. Vestal, "Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance," *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, 2007.

[3] H. Su and D. Zhu, "An elastic mixed-criticality task model and its scheduling algorithm," in *Design, Automation & Test in Europe (DATE), 2013*.

[4] F. Santy, L. George, P. Thierry, and J. Goossens, "Relaxing mixed-criticality scheduling strictness for task sets scheduled with fp," in *24th Euromicro Conference on Real-Time Systems (ECRTS), 2012*.

[5] G. Bernat, A. Burns, and A. Liamosi, "Weakly hard real-time systems," *IEEE transactions on Computers*, vol. 50, no. 4, 2001.

[6] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem; overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, May 2008.

[7] G. Bernat, A. Colin, and S. M. Petters, "Wcet analysis of probabilistic hard real-time systems," in *IEEE Real-Time Systems Symposium, 2002*.

[8] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quinones, and F. J. Cazorla, "Measurement-based probabilistic timing analysis for multi-path programs," in *Euromicro Conference on Real-Time Systems (ECRTS), 2012*.

[9] S. Baruah, "The federated scheduling of systems of mixed-criticality sporadic dag tasks," in *IEEE Real-Time Systems Symposium (RTSS) 2016*. IEEE.

[10] R. Medina, E. Borde, and L. Pautet, "Directed Acyclic Graph Scheduling for Mixed-Criticality Systems," *22nd International Conference on Reliable Software Technologies - Ada-Europe*, 2017.

[11] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *International Conference on Computer Aided Verification (CAV), 2011*.

[12] S. Baruah, A. Burns, and Z. Guo, "Scheduling mixed-criticality systems to guarantee some service under all non-erroneous behaviors," in *28th Euromicro Conference on Real-Time Systems (ECRTS), 2016*.

[13] Z. Al-bayati, J. Caplan, B. H. Meyer, and H. Zeng, "A four-mode model for efficient fault-tolerant mixed-criticality systems," in *Design, Automation & Test in Europe (DATE), 2016*.

[14] R. Medina, E. Borde, and L. Pautet, "Availability analysis for synchronous data-flow graphs in mixed-criticality systems," *11th IEEE International Symposium on Industrial Embedded Systems, SIES 2016*.