Non-Preemptive Scheduling of Periodic Mixed-Criticality Real-Time Systems Literature Review

Real-time systems play an important role within today's society and technologies. Real-time systems have to have logical correctness and temporal correctness [1]. We must be able to rely on these systems to ensure they can be reliable within real world timing constraints. This means that not only does it have to do the right thing, but needs to do the right thing at the right time. Real-time systems prioritize predictability more than their performance [1]. The paper we are doing a literature review on focuses on the importance of predictability. We are looking into non-preemptive scheduling of periodic mixed-criticality real-time systems, as an offline analysis. Non-preemptive scheduling is a technique used in real-time systems in which a schedule executes a process completely before giving control of the processor to another process [1]. When defining periodic tasks, it is describing tasks that have a minimum amount of time between job releases. Another focus from the paper is mixed-criticality real-time systems. This refers to systems that attempt to cope with the conflict between timing safety and resource efficiency through mode-based approach [2]. An important term to note throughout is the term criticality, which is a property to a task which defines its priority or importance. With mixed-criticality systems, they operate by switching between various criticality modes depending on the resource requirements by the executing tasks [2]. These systems have to function at different levels of their criticalities. A mixed-criticality real-time system has a mode which chooses to prioritize whichever tasks are placing a higher demand for them to be executed: low criticality tasks or high criticality tasks. Criticality mode switches occur at the moment that tasks with larger execution times need to execute so that high criticality tasks are prioritized over the lower criticality jobs. A system can be provably schedulable in all modes discussed, if all the tasks that have been given a criticality mode meet their timing constraints. To accomplish this, the probabilistic approach to real-time systems can provide an optimal solution to schedulable mixed-criticality systems. We can do so by using probabilistic worst case execution times of all tasks. We will use a probabilistic distribution as an upper bound to the execution times of all tasks in the probabilistic real-time system to predict which criticality mode is the optimal one to use at any given time. A reason we have come to learn on why we would want to use this approach is because we want to increase the efficiency of the resources being used in the system. With our mixed-criticality model our goal is to reduce extremities with executions and make decisions on which tasks to execute at the right moment. We need to consider how many times the real-time system can fail at run-time to better improve its performance in future usage of the system. This is why the usage of probability with real-time systems allows us to reinforce the system every time a failure occurs at runtime. We talked about mode switches on how it is a system-wide property of the real-time system. We have a type of mode switch known as novel job-level mode switching. With this approach, we aim to have greater efficiency with the jobs. With novel job-level mode switching, whenever a job exceeds the worst-case execution time assumed for a given mode, only that particular job switches mode (changes run-time criticality), and a more pessimistic worst-case execution time (appropriate for a higher criticality) is assumed [2]. A con to using novel job-level mode switching is that scheduling tasks using this approach becomes more difficult. However, we are able to limit the number of tasks we need to drop which leads to a more optimal schedule. In classical approaches, when a task exceeds its worst-case execution time threshold, all tasks through the system wide mode switch are assumed to have more pessimistic worst-case execution times [2]. Using novel job-level mode switching, we are able to adjust the probabilistic worst-case execution time of the task that exceeded its threshold. This can lead to less tasks missing their deadlines.

To create schedules that are independent of their probabilistic worst case execution times, an important thing is to do is to quantify how many times a criticality mode changes in a multi-criticality probabilistic real-time system. The idea is keep track of criticality mode changes while ensuring that the max number of low criticality jobs are executed. Our mixed-criticality scheduling will all be done on a uniprocessor. To reiterate on the approach to using our approach with mixed-criticality mode is that we are able to analyze the jobs and modes that will help us schedule a resource-efficient and optimal schedule for the jobs. With this, we are also taking into consideration the criticality level of each job. We also used probabilistic worst-case execution times to predict whether or not a job is entering a higher criticality mode. For every jobs criticality level modes, we create a schedule from the point of that mode, all the way until we finish executing, searching for the optimal system utilization that results from all the paths taken. The probabilities we get are independent from what decisions we make on the system process. Overall, this
mixed-criticality probabilistic real-time approach to scheduling is unique in the sense that it not only ensures that priorities are being properly assigned to all tasks, but this approach also ensures that we optimize the schedule while we assign these priorities to each task. The system has switches between modes in order to maximize the efficiency of the execution of all tasks in any given point in time in the real-time system. A system with low criticality execution would be considered the complex model, while the safe model would be the system with high criticality execution. This allows us to ensure our schedules are optimized and safely schedulable. In this paper, we come to learn that there are many different equations and notations to take into consideration. In the section of notations, equations, and assumptions we come to learn the meaning and description of all of them used in this work. One of the first things to go over is the background on the probabilistic equations. We will have some random variable C that represents the worst-case execution time of the assignment and has many worst-case values, each correlated with a probability associated. The probability density function (PDF) of C is usually associated with continuous distributions gives us some probability that C will have a value between a and b for the following equation. \[ P(a \leq C \leq b) = \int_{a}^{b} f(x) \, dx \] with \[ \int_{0}^{\infty} f(x) \, dx = 1 \] [2]. We also have a equation for discrete distributions which is called probability mass function (PMF): \[ f(x) = P(C = X) \] with \[ \sum_{0}^{x} f(x) = 1 \] [2]. There is also a cumulative distribution function (CDF) \( F(x) \) used for continuous distributions which represents the cumulative probability that some C value is less than x as: \[ F(x) = P(C \leq x) = \sum_{0}^{x} f(x) \]. The CDF has a complement with the following equation \[ F(x) = P(C > x) = 1 - F(x) \], which works for continuous and discrete distributions. One thing I learned was that now we can use the information prior to come up with a computational model. We have a set of periodic tasks that are performed in a system. A normal instance of a task is referred to as a job and is represented by \( J \). The random variable C illustrates how a job is executed. There is some function \( f(x) \) that is its probabilistic worst-case execution time and C is a worst-case execution value that is not likely to surpass some probability \( P \) at runtime. We are going to essentially set up a computational model and will layout the variables and information required to do as such. \( P \) will be the cumulative chance that C is derived as \( P = P(C > C) = F_{C}(x) \). J will arrive at some time \( a \) and some deadline \( d \). L is the job's level of criticality. The job inherits the degree of criticality of the assignment to which it belongs and assume that the first task of each assignment always arrives at time zero. Due to this approach not considering system criticality levels, we delegate criticality levels to a single job and adjust each task's criticality mode. We will have some hyperperiod variable H. The paper talks about hyperperiod job sets, which from class I can assume it is a list of jobs that are done from a time span of 0 to the hyperperiod value H. We also have a model of criticality which we will consider three levels of criticality for a job: LO, MI, HI [2]. We will represent the worst-case execution time thresholds for the three levels of criticality as C with a LO, MI and HI as their superscripts. The probability of these execution limits being surpassed is derived from the probabilistic worst-case execution time. All of these C values have equations that say the probability that each one will exceed LO, MI, or HI. For C LO we have \[ P = 1 - \sum_{0}^{x_{C_{LO}}} f(x) \], C MI: \[ P = 1 - \sum_{0}^{x_{C_{MI}}} f(x) \], and for C HI it will be 0 since it is the highest threshold of execution that cannot be met [2]. With these three equations we are able to determine whether a job's execution time will exceed some criticality level using probability. We come to learn that a work's criticality is its property that corresponds to its value in the method and a job's criticality reflects the degree of effect on the operating protection of the device when its target is met. The change in criticality is done at run-time. Jobs can be split, which in this case we do it into three disjointed sets. We have a set of LO criticality jobs, MI criticality jobs, and HI criticality jobs. They are all together in union. We assume that finite probabilistic worst-case execution time distributions are often confined to the worst-case execution time with a set upper limit. The next thing we come to learn is about mixed-criticality probabilistic real-time models. This is when the jobs run, it will take more time for any task to complete execution and hence reach a higher criticality mode. This applies more to the MI and HI criticality jobs. We suggest a model of a mixed-criticality scheme based on a graph structure in this section. It is possible to depict a network of possible events that can occur in the device at runtime with the graphs. Then we
Non-Preemptive Scheduling of Periodic Mixed-Criticality Real-Time Systems

Literature Review

Nestor Vallejos
Eduardo Nogueria

can execute explorations in a room with all possible schedules by graphs and aim at optimality. We do so by trees derived from the model of the graph that shape a forest to be explored. To describe how mixed-criticality probabilistic real time systems work we describe them through a directed graph. In each hyperperiod, the graph reflects the system’s potential work schedules. We use a directed graph to depict the possible job schedules which is defined as a tuple of $G = \{V, E\}$, where we use $V$ as a finite set of elements called nodes and some $E$ which is the finite set of ordered pairs of elements of $V$. We have nodes which each node $J$ is an element of $V$. In addition, there is a subgraph within each node that reflects the modifications of the job's criticality mode during execution. We have to define each node from $V$ which is if $J$ is HI criticality then the node $J$ contains a subgraph with three nodes representing the job criticality modes, if its MI then it has two nodes, and if its LO it does not contain a subgraph because the job does not enter higher modes [2]. It should be remembered that the labeled probabilities are just for representation and that is, the chance of accessing the node of HI criticality is continuous and not conditional on the MI node of the work. Using graphs we have arcs, which correspond to the possible sequence relations of jobs. When we have an arc we have $\{J, J’\}$ in the set of $E$ which represents the possible ordering of jobs. In the graph we look at, the graph is directed and no self loops exist or any arcs connect to themselves. Since we have nodes in our graph, they are connected by labelled arcs representing possible transitions of the job criticality modes with some probability [2]. When a job needs more execution time, it will need more higher criticality and as a consequence of taking the idea of criticality mode transition to work level, these two requirements are imposed in the model. In LO-criticality mode, a job still starts running and then passes to higher criticalities, therefore the arcs from other jobs will only reach the LO-criticality node. The arcs will go to higher criticality nodes which may only come from the lower criticality nodes with the same work within the subgraph. The arcs can exit from any node, which means that in every criticality mode the job can finish execution. In order to achieve a timetable that is suitable for resource use, we need to explore all the possible combinations of work sequences. To do so, the above mentioned graph is unfolded into trees that are graphs without cycles that are directed. The trees represent all feasible work which call trees of discovery. A node $J$ in the tree does not exist if it already resides between the root and the desired addition stage. In the schedulability analysis section, the trees mentioned in this section are used to successfully schedule tasks using job scheduling. The scheduling choice is dependent on particular criteria that are introduced. First, the trees are investigated based off of paths using that criteria. The paper also presents a strategy for reducing the complexity of the offline tree construction process. From the probabilities derived from the job probabilistic worst-case execution times, the likelihood of the system entering MI or HI criticality gets calculated. A significant thing to remember is that the probability does not change based on scheduling because the probabilistic worst-case execution time is not derived by taking scheduling into account. However since the probability is obtained by selecting an execution threshold on the probabilistic worst-case execution time, this alternative tells us whether or not a job enters a high criticality. Alterations in the specified task parameters becomes required if the probability is greater than expected. In the tree metrics subsection, it begins by discussing the graph representation that was shown in the prior section. An important factor to consider when it comes to tree metrics is response time. A job's response time is the difference between the time it takes for the job to complete and the time of the job’s arrival. The value of the completion time depends on the time it takes to execute the job and the moment when the job starts to be executed. The worst-case finishing time value depends on both the execution of prior jobs as well as the option that was chosen for scheduling only when arrival time is given. The Worst Case Response Time (WCRT) was formally defined in this section as: Given a node $J''$ in the subgraph of job $J''$ in a path $(J,J'')$ for some $J'' \in \{J',J''\} \in \text{path}(J,J'')$ the Worst Case Response Time for node $J''$ in L criticality, $L = \{LO,MI,HI\}$ is: $\text{WCRT}(J''^L) = \max(0, \text{WCRT}(J''^{L'} - a'' + C''L)$ where $a''$ is the time of arrival and $C''L$ is the worst-case execution time at L criticality mode of job $J''$[2]. As a result, the WCRT is a function of the path in the model proposed, which varies among several differing criticality levels for tasks as well as differing schedules. Implementing the aforementioned definition, the deadline miss for a job is defined as: If $\text{WCRT}(J''^L) > d''$, $J'' \in \text{path}(J,J'')$ for any L in $\{LO,MI,HI\}$, a job represented by its node $J''$ in a path $(J,J'')$ is said to have missed a deadline [2]. Another tree metric mentioned in this section is path utilization metric. The path utilization is the job’s demand for the processor in the path’s schedule. It is formally
Non-Preemptive Scheduling of Periodic Mixed-Criticality Real-Time Systems Literature Review

Eduardo Noguera
Nestor Vallejos

defined as \( U(path(J,J)) = \frac{SC}{HT} \) \( \forall J \in \text{path}(J,J) \) where \( a \) is the time it takes for the job to arrive, and \( H \) is the schedule's hyperperiod [2]. Another tree metric that is being utilized is the System Criticality Mode Probability. Using System Criticality Mode probability, the probability will mainly be obtained from the probabilistic worst-case execution times of the jobs. There is a possibility that a job will take a certain amount of time to complete and as a result, will fit in a particular criticality mode. Since the probabilistic worst-case execution time is not dependent on the schedule, job reordering will not affect the resulting probabilistic worst-case execution times. Since the probabilities are obtained from the probabilistic worst-case execution times, the schedule does not change these probabilities. As a result, the only thing that can be extracted from this is that the probability that at a given point in time, the system will achieve a higher criticality. This paper recommends using the probabilistic worst-case execution times to be able to derive the probability of the system entering a higher criticality in order to relate the different methods where system criticality would be utilized.

If at least one of the jobs in its direction reaches MI criticality, the system in its entirety enters MI criticality. For any two occurrences A and B with \( P() \) giving their probability of occurrence, \( P(A \cup B) = P(A) + P(B) - P(A \cap B) \) [2]. In terms of System Criticality Probability, For a certain path \( (J,J'') \) for some \( J \), the probability that the system enters HI criticality is given as \( P_{\text{HI}}(\text{path}(J,J'')) = 1 - (1 - P_{\text{sys}}^{\text{HI}}) \) \( \forall J' \in \text{path}(J,J'') \) & \( J'' = \Lambda^\text{HI} \) OR \( J' \in \Lambda^\text{HI} \), \( J'' \) is a leaf node. In a similar manner, the probability that HI criticality is achieved in the system is \( P_{\text{sys}}^{\text{HI}}(\text{path}(J,J'')) = 1 - \prod (1 - P_{\text{sys}}) \) \( \forall J' \in \text{path}(J,J'') \) & \( J'' = \Lambda^\text{HI} \). \( J'' \) is a leaf node [2]. It should be mentioned that this probability is not a property of paths, but a property of a system. For the schedule, the probability remains unchanged because the overall number of jobs in the hyperperiod stays the same. If this probability is greater than a certain permissible probability maximum, it cannot be minimized by scheduling and must be enhanced at the design level before the probabilistic worst-case execution times are calculated. In the System scheduling subsection, the tree we’ve been discussing is reduced to achieve a schedule by deleting all but one optimal path for each job in each criticality. In the tree, we start by identifying a valid path. Defined formally, A valid path is a path \( (J,J'') \) for a node \( J \) and a leaf node \( J'' \) in a tree is said to be a valid path iff: WCRT(\( J'' \)) \( \leq d' \) \( \forall J' \in \text{path}(J,J'') \) and \( \exists J' \in \text{path}(J,J'') \) \( \forall J' \in \Lambda^\text{HI} \) [2]. To go through all the LO-criticality jobs, a valid path is not needed. As a result, there are paths that can have just jobs with no MI or LO criticality and jobs with HI criticality. A path that is not valid is defined in this section as the path \( (J,J'') \). By making all jobs meet their deadlines, but not containing all the HI-criticality jobs, and vice versa, a non-valid path can be present. A valid path \( (J,J'') \) is said to be a dangerous path if: \( \exists J' \in \text{path}(J,J'') : J' \in \text{path}(J,J''), L > L' \) [2]. A dangerous path is not inherently a path which is not valid. In MI criticality, a job may reach its deadline and as a result may appear schedulable. On the other side, if it hits HI criticality, the same job could miss its deadline. A valid path \( (J,J'') \) is optimal if all the finite possible paths for a node \( J \in T \), a leaf node \( J'' \), if the following conditions are satisfied in the order of priority: In the path \( (J,J'') \), the number of MI-criticality jobs is maximum, in the path \( (J,J'') \), the number of LO-criticality jobs is maximum, and in the path \( (J,J'') \) has maximum usage. In each of its criticalities, it searches for some optimum path for each job. Using this strategy, the outcome of this tree, \( T_{\text{schd}} \), results in \( T_{\text{schd}} = \text{Optimize}(T - \text{path}(J,J'') - \text{path}(J,J)) \) \( \forall J' \in J, \forall J \in \Lambda \) [2]. The \text{Optimize()} function eliminates all except one valid and optimal path for each job. The case in which there is at least one path \( (J,J'') \) for any \( J \) with maximum utilization \( U(path(J,J'')) \), the tree \( T \in F \) with any root node \( \bar{J} \in S \) is picked. The goal that this tree has in mind is to find a schedule that is considered valid for the job. The following is obtained at the conclusion of the analysis. A valid path can be found using a schedule that contains jobs that only are in LO-criticality mode. If said schedule does not exist, or there is no such path can be found to be valid, the schedulability test is not satisfied and adjustments are done to the parameters of the task. For every optimal schedule of the remaining scheduled jobs from that point forward, an optimal valid path is found for each MI-criticality and HI-criticality task. All the MI and HI criticality jobs and the maximum amount of LO criticality jobs are found in this schedule. In terms of complexity reduction, whenever a new node \( J' \) is added to the tree, we check the condition WCRT(\( J' \)) \( \leq d' \) is satisfied. When the condition is no longer satisfied, the tree will not be constructed in that direction. The implication being that WCRT(\( J' \)) \( < d' \) classifies \( J' \) as a dangerous path when this occurs. Since the node being inserted has already missed its deadline, it is not appropriate to schedule the system in this path. Using this approach, the
Non-Preemptive Scheduling of Periodic Mixed-Criticality Real-Time Systems Literature Review

Eduardo Noguera
Nestor Vallejos

Tree is minimized when it's constructed. This reduces the overall complexity process of creating the tree. When it comes to the use of online scheduling, the paper begins to talk about the offline analysis, specifically how a scheduled tree $T_{schd}$ is found for some task set $\Gamma$. The schedule is optimal in the utilization of the resource and assures the scheduling of all jobs that have a high criticality. The root node $\bar{J} \in S T_{schd}$ is scheduled as the first task. The order of jobs at $T_{schd}$ is then abided by. The online schedule is set in such a way that the schedules for each of its criticalities and for each job are fixed. The schedule, however, is flexible in the sense that the change in the criticality mode of the job chooses the schedule that is fixed from that specific moment onwards. A job will take more time to perform at run-time to achieve a higher criticality. By switching to the subgraph of the job with a higher criticality, the path taken by the system is affected. In the $T_{schd}$ tree, the optimized path from that node onwards is already apparent. For all jobs created by the task set, the system continues to function in this way. The online process proceeds until the leaf node located at the end of the hyper-period is reached. In this manner, the offline schedule has been made to be the most efficient and the online scheduler merely needs to proceed scheduling the remaining series of jobs. Using this method, the idea of system criticality does not exist since the research focus is in the job’s criticality mode. In their LO-criticality mode, all jobs start to execute and can then switch to a higher criticality. The scheduling tree can continue to change criticality modes in a safe and optimal manner. In the experiments section, it is suggested to utilize a test case that is practical and non-exhaustive to illustrate all the contributions that were done. Complexity is also talked about in great detail when it comes to these test cases. It introduces the task sets and the scheduling derived from their graph trees. By optimizing the number of carried out LO-criticality jobs while those jobs reach higher criticality modes, it can be ensured that the method that is taken will be the most optimal. By finding an ideal schedule, this is proven. The solution that is mentioned is contrary to the classical way to solve this where as the system reaches high criticality, but all the LO-criticality workers are dropped. In this section, a table is shown where there are five tasks, a HI-criticality task, 2 MI-criticality tasks and 2 LO-criticality tasks. Each task had their own respective execution times $C^L_i$, $C^M_i$ and $C^H_i$ as well as their own periods. Units is the measurement used for all times. In the hyper-period that has a size of 30 time units, the task set has 10 jobs. For instance, if the first job of the first task gets executed, the second job itself or the first job of the rest of the jobs can move forward with execution. The graph for the task set is given. It has 134 nodes. It is part of a greater expansion of trees contained within a forest. $J_{11}$ is the root of a section of one of the trees. The tree contains a total of 502,063 nodes. The following scheduling is recommended from the derived valid paths in the LO-criticality mode of the jobs: $J_{11}$, $J_{12}$, $J_{13}$, $J_{31}$, $J_{32}$, $J_{33}$, $J_{34}$, $J_{35}$ with a utilization of 0.65 [2]. The risk of entering MI criticality is 0.22. The risk of entering HI criticality is 0.099. The second table shows the schedule that is optimal for the jobs in a higher criticality mode. It also indicates the subsequent usage of the remaining schedule. The table further illustrates that both the MI and HI-criticality jobs can be scheduled. It can be seen that the $J_{13}$ job is HI-critical, and the $J_{21}$ job stops its execution as a result of $J_{13}$ entering a higher criticality. The $J_{21}$ job is the last hyper-periodic job. Therefore, for both cases there is no schedule which shows the benefit of the method being used. If $J_{13}$ reaches a MI-criticality but not a HI-criticality, there is space for $J_{21}$ to execute. It is only when $J_{13}$ reaches a HI-criticality that $J_{21}$ has to stop executing. In order to maximize the value of this method, apart from the findings that were acquired, another task set is observed in a third table. The table has four tasks in this scenario, 1 HI-criticality task, 1 MI-criticality task and 2 LO-criticality tasks. Each task had their own respective execution times $C^L_i$, $C^M_i$, and $C^H_i$ as well as their own periods. The hyper-period is 30 time units and is made up of a task set that has 7 jobs. The suggested LO-criticality schedule for this case is: $J_{11}$, $J_{41}$, $J_{31}$, $J_{32}$, $J_{33}$, $J_{34}$, $J_{35}$ [2]. The schedule’s utilization is 0.966. In the HI-criticality mode, where the first task has an execution time of 8 and a period of 10, we see that the first task almost misses it’s deadline. The first task begins in LO-mode. However if the $J_{11}$ job reaches HI-criticality, the correct optimal HI-criticality path from the $J_{11}$ node is the one containing all the HI-criticality jobs that reach their deadlines followed by the maximum number of MI-criticality jobs, and lastly by the maximum number of LO-criticality jobs. There's a path from the tree that was found: $J_{11}$ from LO to MI to HI, then $J_{21}$ from LO to MI, continued by $J_{12}$ from LO to MIHI, and lastly $J_{13}$ from LO to MI to HI. As a result, the HI-criticality schedule for $J_{11}$ is: $J_{11}$, $J_{21}$, $J_{31}$, $J_{32}$ [2]. When both the
HI-criticality and MI-criticality jobs are in the highest mode, the total utilization of the schedule is 0.933. The utilization would have been 0.8 had it not been that we include all LO-criticality jobs. As mentioned, though, there is space to execute $J_{21}$. In HI-criticality, $J_{41}$ is not included in the schedule after $J_{11}$. This is because the $J_{11}$ node path that passes via $J_{41}$ does not encounter the $J_{13}$ node in a HI-criticality, leading it to be a dangerous path. The problem in this situation is that a utilization maximum greater than 1 is evidently not preferred.

In this section complexity is mentioned. The complexity of constructing a tree is determined by the number of jobs. The amount of nodes on the graph grows when the number of MI and HI jobs also grows. Then it relies on the probabilistic worst-case execution time and worst-case execution time values obtained from $c_i^{LO}$, $c_i^{MI}$, and $c_i^{HI}$ as well as from the job deadlines. The closer the worst-case execution time values are to the date, the smaller amount of exploration branches need to be searched. This also suggests that there are fewer potential schedules. This is because the number of valid paths is lower. The fourth table in this section shows the node amount versus the potential complexity given by a certain job amount. When all jobs appear at the same time, the worst-case complexity is presented. A more accurate indicator of complexity will be a value that is lower than the values that were presented. To conclude this review, it can be said that a method involving graphs was used to find schedules for Mixed Criticality probabilistic Real-Time Systems. All jobs were able to get scheduled in each criticality level. All the schedules ensured that the utilization was preserved in the most efficient way possible and that jobs were able to be scheduled in the most secure way they can be scheduled while reducing the complexity of the exploration process. Also, there were probabilities that were able to be acquired by analyzing the system and it’s schedulability. The resulting research in the paper anticipates to utilize response times in mixed criticality systems in an application focused environment, where decisions can be made based on probabilities. It goes on to mention that task dependence is another topic that the authors desire to further explore.
References
