July 2020

Research review on mixed-criticality scheduling

Hattan Althebeiti
University of Central Florida

Research Review

This research review focuses mainly on papers that has investigated mixed criticality scheduling. In the following review, we will discuss each paper and its contribution to this topic. First, we need to define mixed criticality, system and what are standards that most of these contributions relied on. Mixed criticality RTS is based on the Vestal model developed by Vestal at Honeywell Lab. The goal of this model is to divide tasks based on their criticality which can simply mean how sever the consequences will be if a task misses its deadline. If we took avionics system for example, high criticality task can cause disrupt entire system or cause some malfunctioning. In contrast, low criticality tasks can tolerate missing deadline which mean some features might be disabled or produce inaccurate information occasionally. This concept is crucial for the certifying authority as it needs to ensure the safety of the overall system and verify high criticality tasks ensuring they will meet their deadline.

After defining the mixed criticality concept, we must make sure that high criticality tasks will meet its deadline. The common method used is defined in the Vestal model. First, each high task will have two execution time. (C_{HI} and C_{LO}) where is always $C_{HI} > C_{LO}$. In addition, the system will have two operation modes, high operation mode and low operation mode and it works as follow:

1. The system will start in the low mode and schedule all tasks based on the chosen algorithm assuming all high task will finish within C_{LO} execution time.
2. The system continues execution in low mode until one of the high tasks exceeds its C_{LO} execution time, in which the system shifts to high mode in order to guarantee meeting the deadlines for high tasks.
3. The system will suspend (abandon) any low task in order to devote all resources and time to high tasks only.
4. The system may switch back to low mode when all the high tasks finishes by their C_{LO}.

In this model we are assuming a periodic system where the period defines the interarrival time between two consecutive jobs. Moreover, we consider the tasks to be independent of each other without sharing any resources except the processor. This assumption simplify the analysis of the model.

Although the Vestal model is considered evolutionary because it expanded the are of RTS research. However, several critics have raised regarding this approach, specifically from system designers and engineers. We summarize them as follow:

1. Low tasks are present for a reason and abort their execution completely can cause a problem. They should be allowed to make some progress and execute as long they don't affect high tasks.
2. If we are considering a system that will run for long period of time, we should guarantee that a switch mode will revert the system to its original mode which is low mode.
3. We need to make sure that no high task will exceed its C_{HI} and no low task will exceed it C_{LO}.

Now when we estimate the WCET (worst case execution time), there are normally two approaches. 1) WCET determined by simulation which tends to be optimistic and less reliant for high tasks, but it's the easier way to do it. 2) the second approach which include static analysis of the source code which is time consuming and require more detailed review which is hard. From those two approaches the concept of risk analysis emerges to determine the criticality of a task. Therefore, for high task we use the conservative approach which probably is not the common case but it guarantees the safety of the system. The simulation approach can be used for low tasks as they are less critical and can handle missing the deadline or being dropped.

The first research we will consider is "Towards A More Practical Model for Mixed Criticality System" by A. Burns and S.K Baruah. This paper focuses on major problem with the Vestal model, which is suspending all low task. Therefore, this paper try to accommodate low tasks even after the system transfer to high mode. They also propose a condition for which the system can transit from high mode to low mode. Moreover, they also propose a priority assignment technique for fixed priority with mixed criticality. Schedulability analysis for mixed criticality is considered NP-hard because it is impossible to cover all low tasks when the system transfers to high mode, which poses a great limitation to the analysis. In order to allow some progress for low tasks in high mode, we must not abandon them completely. We should guarantee some service level to them. This paper proposes three approaches for this solution:

1. Modifying the priority of these tasks to be less than any job priority in the high criticality group.
2. Reducing the execution time of those tasks to allow both high tasks and low tasks to execute without any conflict.
3. Increasing the period of low, which in turn reduce the number of jobs. Thus, giving more flexibility for high tasks.

This discussion considers the implementation on a single processor with fixed priority algorithm like RM. The first method imply that we have the capability to change the priority of low tasks online. Therefore, each low task will have two priority, P_{LO} will denote the normal priority at low mode and P_{HI} will denote the priority in high mode. However, as we stated earlier, the priority of P_{HI} must be less than the lowest priority in high tasks group. The ordering of the tasks will be the same in both modes. One drawback with this approach is that it produces a great overhead due to the re-prioritization and reordering as jobs will have to be taken out of the queue and put back again in the correct position. As stated before, we need a way to ensure that no low tasks will execute for more than its C_{LO}. This can be done by run time monitoring and enforcement for each task. Two drawbacks can be found with modifying the priority approach. First, if we allowed the priority to be modified, then this can be exploited to undermine the overall reliability of the system specifically for high tasks. Second, this method doesn't assure that low tasks with short deadline will be serviced after the mode changes as they will probably miss their deadline.

We mentioned before that the system must revert to its original mode after all high tasks finish by their C_{LO}. In order to do this, we use the simplest method where the system waits for an idle tick and switch back to low mode under the stated condition above. There are other two protocols that can handle the switch problem with a better performance defined as fixed task priority FTP protocol and fixed job priority FJP protocol. FJP dominates FTP because it is more specific as it is applied to a job. Unlike FTP which is applied to a task.

We will explain the FJP briefly as this is an important protocol that can reduce the switch time of MCS. This research produces an efficient way for the system to go back to low mode to resume low tasks. The two protocols are limited to two criticality level. We may have more than that and the protocols will work the same way. First, with FJP the system will create a variable $Q_i(t)$ for each task T(i). This variable stores the amount of time since the last job of T(i) has been released to time t. During this time if any $Q_i(t) > C_{LO}$ for T(i), then there is an overrun and the system must switch to the higher mode. Otherwise the system computes the remaining execution time at the lower level and Qi(t) will be at most C_{LO}. This value is referenced in the paper as act-rem. The system switch back to the lower mode after the following conditions are met.

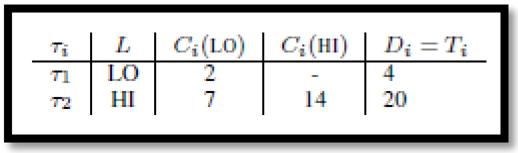
a) For each task T(i) in high mode, act-rem >= 0 (the task is not overrunning)
b) For each task T(i) in high mode, act-rem =< ref-rem.

The variable ref-rem is a value that represent the remining execution time of job j(i) of task T(i) when all jobs are executing for WCET and it is stored in an alpha-queue data structure. Thus, ref-rem will be larger than C_{LO}. And act-rem must be less than or equal to ref-rem. The proof of correctness for FJP can be found in the original paper included in the references.

Now, we consider the second approach which is reducing low tasks execution time. With this method, we make a few adjustments to our model.in addition, we will also have a schedulability test that either accept or reject a task. Then to determine the service level that will be provided to a low task that has passed the test, we use a sensitivity test. In this model, we have two execution time C_i (LO) and C_i (HI) where $C_i(LO) > C_i(HI)$. Some tasks may have $C_i(HI)$

zero which indicate that they cannot execute in high mode. In this model modifying priority is not allowed and low tasks are not allowed to execute beyond $C_i(LO)$ in low mode and beyond $C_i(HI)$ in high mode. One more issue needs to be considered in which the low jobs begin in low mode and then the system transfers to high mode. In this situation, the job may miss the deadline as its budget decreases. However, it is guaranteed to receive $C_i(HI)$ execution by its deadline. The schedulability test used for this model is based on the Adaptive Mixed Criticality scheduling which will be covered afterward. In addition, when the system return to its low mode operation, it only needs to adjust the execution time for each task to its corresponding $C_i(LO)$.

Now we consider the adaptive mixed criticality AMC. AMC is built on the idea that no low task job is released after the system shifts to high mode. The paper considers a system with fixed priority scheduling with pre-emption, non pre-emptive, and deferred pre-emption. The pre-emptive behaviour occurs when a high priority appears during the execution of low priority task. The system suspends the low priority job and handle the high job. On the other hand, the non pre-emptive approach doesn't allow any pre-emption. The low priority job must finish before the high priority job starts. The deferred pre-emption allows the pre-emption at certain point of time. Therefore, pre-emption can take place during that time only. The paper defines a specific type of deferred pre-emption called Final Non pre-emptive Region (FNPR), which has been proofed that it dominates the other two pre-emption behaviours. Moreover, the paper combines AMC with FNPR to improve the schedulability for both low and high tasks. The following example shows how AMC-NPR makes a different in meeting the deadline. the following illustrative example and figure has been taken from the original paper.



τ_i	L	$C_i(\mathrm{LO})$	$C_i(\mathrm{HI})$	$D_i = T_i$
τ_1	LO	2	-	4
τ_2	HI	7	14	20

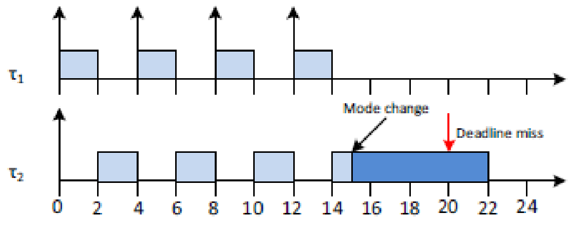
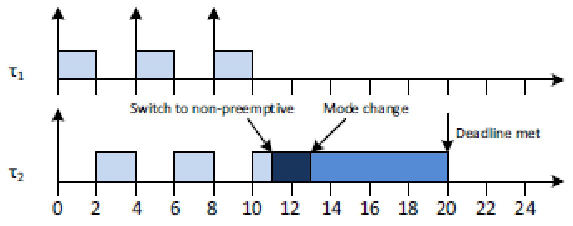


Fig. 1. Execution under AMC



The system model follows the same assumptions that we made at the begging where we have two execution time for high tasks and one for low tasks, and if high job doesn't finish by its low execution time, the system will trigger the mode switch form low to high. However, there is one more assumption we make regarding the execution time. We assume the execution time is some how derived from the criticality level of the job, the higher the criticality, the more conservative the value. Similarly, the system will revert to its low mode only if all low jobs that will be released can be guaranteed. As stated before, the first step it to compute the response time of each task in the low mode. For high mode, we are only concern with high tasks, therefore the equation considers high criticality jobs only. Exact and full analysis can be found in the original paper.

The new concept here is deferred pre-emption and the way it improves the schedulability. It does that by balancing between reducing the interference imposed on low tasks by high tasks and increasing the blocking time for high tasks. According to the research, most high tasks can tolerate some blocking as long these tasks remain schedulable. Now we will explain the optimal algorithm for mapping priorities and FNPR length to a task. The algorithm proposed by Davis and Bertogna is summarized as follow:

1. From the lowest priority level, we determine the set of tasks that can be schedulable at this level and the minimum FNPR for it to be schedulable. If a task cannot be schedulable under those conditions, we can declare it as unschedulable.
2. Among those tasks, pick the one with the lowest FNPR and assign it to that priority level. Thus, we ensure minimizing the blocking time on the high task.
3. After finishing this assignment, we move to the next higher priority level.

We use binary search to find the minimum FNPR in a task. The value of FNPR can range from 1 as the minimum up to the execution time of a task. If the task is not schedulable within this range, we conclude that the task is not a candidate for this priority level. However, one complication may arise with the presence of FNPR, which introduces some blocking time. Therefore, the blocking time may push the higher priority to the point where overrun occurs and can continue to transit over several jobs. This mean that the interarrival between two jobs will shorter with the presence of blocking time which may results in an overrun between the jobs. To alleviate this issue, we need to check that for all jobs need to be verified within the busy period at the task's priority level and find which one gives the largest response time.

After explaining the priority and FNPR assignment procedure for each task, we go back to the AMC model. For each task in the low criticality level, the task is divided into two regions, C(LO) and F(LO). The FNPR is the F(LO) region, while the C(LO) region can be executed pre-emptively or non-pre-emptively. Although we can divide the pre-emptive region into two non -pre-emptive regions. However, non of them are longer than F(LO) and only one region in the low criticality task could cause blocking to high tasks. The same concept applies to HI criticality tasks but with three regions instead of two. The three regions are C(LO), C(HI), and F(HI). In the previous discussion, we assumed that the priority and FNPR length are give. Now, we will show a different algorithm called FNR-PA that was also developed by Davis and Bertogna that handles the other case. One more thing to keep in mind that the value of F(HI) is dependent on F(LO) and there are other dependencies such as: F(HI) = F(LO) if C(HI) – C(LO)>= F(LO) or C(HI). The goal of these is to maintain the blocking time between high and low criticality tasks. We start with set of unassigned tasks that include both criticality level. The algorithm works as follow:

1. Binary search will be used to find the minimum F value between 1 and C(HI) in which the task is schedulable. If the task is high, then we fix F(HI) to the minimum of C(HI)-C(LO) and F(LO). Otherwise, the task is low and F(LO) is fixed to the minimum of C(LO) and F.
2. If the task set doesn't have a task that meet those conditions, then the set in not schedulable.
3. If tasks are found in the task set, then pick the one with smallest F and if there is more than one task, choose the low task over the high, and assign it to that priority level.

The AMC-NPR scheduling policy follows the same model explained earlier; the only difference is that it will allow low jobs that have been released before the mode switch to finish executing without guaranteeing meeting their deadlines. However, no further releases will be allowed for low criticality jobs.

$$\boxed{R_i(LO) \;=\; C_i(LO) \;+\; \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{R_i(LO)}{T_j} \right\rceil C_j(LO) \quad (1)}$$

$$\boxed{\begin{aligned} R_i(HI) \;=\; C_i(HI) \;+\; \sum_{\tau_j \in \mathbf{hpH}(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(HI) \;+ \\ \sum_{\tau_k \in \mathbf{hpL}(i)} \left\lceil \frac{R_i(HI)}{T_k} \right\rceil C_k(LO) \qquad (2) \end{aligned}}$$

The system will return to its original when it finds an idle tick and all high criticality tasks are executing within C(LO).

Now we return to the first paper. The last method that can be used to accommodate low tasks is to increase low tasks' periods, which allow the periods to be extended upon the mode changes. In this case each low task will have two periods that will be used for each mode denoted as P_i(LO) and Pi(HI) where P_i(LO) < P_i(HI). This idea is based on the elastic mixed criticality E-MC. In this model, the minimum service required by low tasks will be based om the larger period of that task. Therefore, the larger period will determine the minimum service requirement by the low task. The model also introduce a new variation of EDF called Early Release EDF (ER-EDF) which gives the chance for low tasks to be released and scheduled more frequently without affecting high tasks which according to the paper dominate EDF with virtual deadline in which the system determine the priority of a high task in high mode based on its virtual deadline. A set of tasks is said to be E-MC schedulable if the high tasks with their maximum execution requirement and low tasks with their minimum service requirement are guaranteed to be scheduled under high mode in the worst-case scenario. However, ER-EDF has some issues that need to be considered carefully. First, we need to determine the deadline of the low job if it was released early. As it implies, early release doesn't have a fixed time, the release time is arbitrary. Therefore, we need to determine the deadline for the released job. Second, we need to ensure whether it is feasible to release a job earlier as this may affect the overall system behaviour as the workload will increase which may have a negative effect on the execution of other jobs.

After finishing all approaches All three approaches have some disadvantages. The first guarantee scheduling low tasks but the schedulability is undetermined upon mode changing, there might be a deadline miss. The other two guarantee some service but it doesn't utilize the whole available capacity. In order to enhance the effectiveness of these approaches, the paper suggests such improvement through the following: 1) We combine the last two schemes together, when the system mode changes to high, low tasks will continue executing up to its C_{HI}, then it's priority lowered to Priority in the mode high P_{HI} and the task can continue to execute based on the new assigned priority. 2) The available capacity from the high tasks can be utilized by assigning them directly to low tasks. The second strategy can be achieved using any published technique such as extended priority exchange, capacity sharing or history rewriting. We will explain them within the mixed criticality context.

Assuming the system is in high mode, therefore, all high tasks have an execution time at most C_{HI}. We will define a new variable **g** which stand for gain time and it is calculated as follow ($g = C_{HI} – e$) where e is the response time. Then the gain time can be assigned to any low tasks. For this g value to be used with extended priority exchange, it will be allocated to the budget of the highest priority job that is ready to be executed. If there is no ready job, the gain time is lost. In contrast, capacity sharing saves the gain time in some sense. When a low job exhausts its C_{HI}, it uses the available gain time computed from any higher priority job. The low job is said to be Plugged to the job providing the gain time. The plug is broken when the job reaches its deadline, completed or when the gain time expires. Finally, in history rewriting, a low task that has executed for its C_{HI} budget and has not finished yet can claim the gain time of any higher priority task if the low job acquires the gain time before the deadline of that high priority job. This can be done over and over with more gain time which will be assigned to tasks with lower priority.

Next, we consider a recent paper that suggest a novel idea about graceful degradation for low task upon mode switch. As stated earlier that low tasks should not be abandon and they should be guaranteed with some level of service regardless of the high tasks' behaviour. While most of the work done in this area have struggled to make to accommodate the low jobs with many different approaches, but the implementation still doesn't guarantee anything to tasks with low criticality. This paper presents a clever way to somehow guarantee some service to low criticality jobs. The concept of graceful degradation means that even if a major portion of the system is corrupted or not working, the system will still be working with limited functionality. The paper also introduces the admission control procedure which perform a test on the low jobs, if the job passes the test, then it will have a guaranteed completion rate r(i) for a task T(i). The model will adapt EDF with virtual deadline as the scheduling algorithm and it will use the maximum length of a period to revert to low mode after the mode switch. The research will be based mainly on the Vestal model, but with a few adjustments.

The minimum cumulative admission rate R(i) for a task T(i) is defined as the minimum number of jobs that will be guaranteed to complete after the mode switch take place Therefore, for **n** jobs released after the mode switch, the system will guarantee R(i) * n jobs to be finished. The value of R(i) will not be the same probably for every task, but it must at least equal to R(i). As with previous model, we expect each job to finish within its deadline. However, the low tasks will receive at least R(i) in high mode. Although every low job will have a value of R(i), not all of them will be executable in high mode. A job must go through the admission control to determine if it fits for high mode. The algorithm for admission control defines a variable a(i) which holds the number of the executed jobs for a task. After the mode switch the algorithm will track another value b(i) that represent the newly released jobs and make sure the value of a(i) < b(i) * r(i). if the condition is satisfied, then those jobs will be schedulable in high mode. Otherwise, the job is dropped. The admission rate a(i) must never drop below the minimum cumulative admission rate R(i). This admission rate has been proofed with various tasks and it showed that it will approve the admission of several jobs within a task in high mode.

Several remarks have been made regarding the admission rate and its behaviour. 1) The procedure tries to keep the minimum cumulative admission rate and to reduce the maximum number of consecutive drops. 2) The jobs admitted may have a pattern based on the value of R(i) whether it is a rational or irrational value. 3) The first jobs of any LO criticality tasks are admitted upon mode switch. The consecutive jobs released after may not be admitted.

Now we move to the new algorithm which is EDF with graceful virtual deadline EDF-GVD. We refer to the virtual deadline formula to compute VD for a high criticality task as follow: $D_i^v = ((C_{LO} / C_{HI}) * D_i)$. However, this equation doesn't provide an optimal value for VD. The proposed equation to compute VD is as follow: $D_i^v = q * D_i$, for each task that belongs to high criticality. Variable q is a value between 0 and 1 and defined as the scaling factor. Before explaining the algorithm, we need to explain the demand bound function DBF and the conditions required by the algorithm. DBF determine the maximum demanded required by task T_i during time interval t. the demand will include all previous jobs that has been released up to time t.

The algorithm works by finding the value of q that satisfy the DBF conditions. First, it performs a binary search that satisfy the following conditions:

$$\textit{(A):}\ \sum_{\tau_i \in \tau_{\mathrm{LO}}} \mathrm{dbf}_{\mathrm{LO}}^{\mathrm{LO}}(\tau_i, \ell) + \sum_{\tau_i \in \tau_{\mathrm{HI}}} \mathrm{dbf}_{\mathrm{HI}}^{\mathrm{LO}}(\tau_i, \ell) \le \ell,\ \forall \ell \ge 0.$$

$$\textit{(B):}\ \sum_{\tau_i \in \tau_{\mathrm{LO}}} \mathrm{dbf}_{\mathrm{LO}}^{\mathrm{HI}}(\tau_i, \ell) + \sum_{\tau_i \in \tau_{\mathrm{HI}}} \mathrm{dbf}_{\mathrm{HI}}^{\mathrm{HI}}(\tau_i, \ell) \le \ell,\ \forall \ell \ge 0.$$

The first condition states that the total demand for low criticality task in low mode over a time t plus the total demand for high criticality task in low mode over the same time t must be less than or equal to t for each t larger than or equal to 0. Similarly, the second states the same but in high criticality mode.

Therefore, the value of q must satisfy the above conditions and the binary search will determine whether to increase or decrease the search domain based on them. If both conditions are satisfied, then we found q. Otherwise the algorithm continue to look. After finding the value of q, we substitute that with new suggested equation to calculate the virtual deadline. Next, the algorithm will set the D_i^v for each high criticality task after it is released, and the scheduling will be based on EDF. The VD value will be used as the second deadline for high criticality tasks in HI mode. If any high criticality job doesn't finish by its lower WCET, the system switch to high mode and all current low criticality jobs will be dropped. However, low criticality jobs will continue to release new jobs in high mode based on the admission rate procedure with their deadline. If job is no admitted, then it will be dropped. Thus, WCET for low criticality task are not considered as hard deadline. It is rather closer to soft deadline with some flexibility of being dropped in high mode only.

The system model will switch back to low mode when if it finds an idle tick (idle time instant) and when the last job of each high criticality task finishes within its C_{LO}. Otherwise, the system will remain in high mode. Although the system model is optimal in the sense it makes guarantee some service to low criticality tasks. Nevertheless, it is not designed for a system where mode switch occurs frequently. In such a case, the system designer may reconsider the C_{LO} and ensure it will not be exceeded frequently. In addition, the research verifies that an idle instant time must exist after some time in high mode which can trigger the system to switch back. The research provides two bounds for both conditions. 1) a bound for the last high criticality task that exceeds its C_{LO} and the first instant of idle time after that. 2) a bound for the latest instant in which the system transfers to high mode and the first time instant after that.

In conclusion, we have covered several papers that deals with mixed criticality system from different perspective. We investigated several approaches to accommodate low criticality task in high mode. All the research in this area are based on the Vestal model with some modifications and assumptions. Also, note that for each research there a certain model that is used. However, we explain it once and then make the distinction afterward. Moreover, we covered some important protocols such FJP that provides a more robust way to transfer from high mode to low mode. In addition to the admission control procedure that will determine the eligibility of low task to be executed in high mode. Similarly, we explained many algorithms, concepts, and ideas that enrich the field of MCS. All resources that have been used are included in the reference page which can be referred to for more detailed analysis about the research.

References:

[1] A.Burns and S. Baruah, “Towards a More Practical Model for Mixed Criticality Systems,” in the 1st Workshop on Mixed-Criticality Systems (colocated with RTSS), 2013.

[2] F. Santy, G. Raravi, G. Nelissen, V. Nelis, P. Kumar, J. Goossens, and E. Tovar. Two protocols to reduce the criticality level of multiprocessor mixed-criticality systems. In Proc. RTNS, pages 183–192. ACM, 2013.

[3] H. Su and D. Zhu. An elastic mixed-criticality task model and its scheduling algorithm. In Proceedings of the Conference on Design, Automation and Test in Europe, DATE, pages 147–152, 2013.

[4] Z Guo, K Yang, S Vaidhun, S Arefin, SK Das, H Xiong. Uniprocessor Mixed-Criticality Scheduling with Graceful Degradation by Completion Rate. IEEE Real-Time Systems Symposium (RTSS), Pages 373-383 2018

[5] A. Burns and R. Davis. Adaptive mixed criticality scheduling with deferred pre-emption IEEE Real Time Systems Symposium, Dec 2014