May 2019

A Review on Mixed Criticality Methods

Alex Jenkel
University of Central Florrida, jenks@knights.ucf.edu

A Review of Mixed-Criticality System Schemes

Alex Jenkel

I. Inspiration

Mixed-criticality systems are a largely-growing area of interest in the real time systems community, as the area of study can help solve many implementation issues. Mixed-criticality is essentially the use of multiple schemes within a scheduling algorithm, and can vary from altering the processor speed of a system, mapping the resources of the system more effectively,

II. Paper 1: RTSS 2013 [1]

Summary

The goal of the 2013 RTSS paper is to determine whether or not it is possible to model a system in which a processor can lower its rate when exceptions arise so that all critical jobs' deadlines will be met. The strategy here will be to create a second task schedule for the degraded mode that does not involve the lower priority jobs. This will create free space that will allow the higher priority schedules to execute with the lower rate, which in turn brings up the question as to what the lowest rate is that a processor can run at and still guarantee correctness.

Setting up the model, several constraints are defined that will outline the calculations used for the rest of the model. Firstly, jobs are indexed from 1 to n, with n_h lying in that range and separates higher priority jobs $1,\ldots,n_h$ from the lower priority jobs $n_{h+1},n_{h+2},\ldots,n$. Similarly, a time vector is set up that denotes the release times and deadlines of each job, labelled $t_1,t_2,\ldots,t_{k+1}$; there are 2n t values in the vector. Once these parameters are established, several constraints are put in place so that accurate analysis may be conducted. One such constraint is that there has to be sufficient time allocated to each job so that it doesn't miss its deadline. Another constraint is that the time allocated to a job's execution must fall between two successive time intervals—i.e. t_j and t_{j+1}. The final constraint is that since at any point the processor may enter degraded mode, the sum of execution times of a High-Criticality task whose deadline doesn't exceed the range being observed (called t_m) must be less than s(the degraded execution speed which is less than one)$\bullet(t_m-t_L)$ where t_L is the point of time where the processor enters degraded speed. In other words, since the processor operates at less than one unit of execution per unit of time, it must have enough time under degradation to execute the higher criticality jobs.

Another aspect of the model is the scheduling policy that will be used. Two cases are considered: preemptive scheduling and nonpreemptive scheduling. A custom approach will be used while the processor runs in normal mode where a portion of each high-criticality job will run (but not the entire job) and then will alternate with a low-criticality job. This will assure that both A) jobs of both criticalities will execute under normal conditions and meet all deadlines, and also B) if the processor goes into degradation mode and the lower-criticality jobs are dropped, there will be gaps in which the higher-criticality jobs can execute with the degraded speed on the processor. While the processor is degraded the high-criticality jobs will execute according to preemptive EDF.

Nonpreemptive schemes are considered but are ultimately discarded as it is NP-hard to determine if a consistent model exists for what is desired. To this end only preemptive schemes will be considered.

Interpretation

After conducting numerous simulations, it was discovered that the degradation speed closely resembles the load of the higher-criticality jobs. Assuming that the jobs are preemptable (otherwise lower priority jobs could block degraded higher criticality jobs when the processor jumps to degradation mode) simulations provided analysis that showed the bounds put in place were only exceeded by at most 0.8, which gives evidence to the model's credibility. While this model is very generalized and fairly simplified, it still provides an excellent foundation in the study of mixed criticality jobs.

Further Exploration

A few options that any future works can expound upon include the use of multiple levels of criticality, resource sharing, multiple execution speeds, as well as considering that it is possible for the processor to return to normal mode after degrading.

In regard to multiple criticality levels, possibilities exist where—especially in safety-critical situations—a certain few jobs are given the highest priority, some jobs are given slightly lower levels of criticality and then most jobs are given the lowest level of criticality. By doing this, more exact schemes can be plotted, and better priority definitions can be described. This could also be combined with the possibility of having multiple processor speeds, which could be allocated to different criticality levels so that the correctness of higher-criticality jobs will be guaranteed.

III. Paper 2: ETH 2013 [2]

Summary

The idea behind the ETH paper is to create a model for a series of tasks that could contend for resources (here defined as either processors or memory) such that proper resource usage is allocated in both normal and degraded modes. The paper describes an acceptable outcome as one in which all tasks can be scheduled for all criticality levels. Proving this lies more in tune with using many simulations to determine whether or not a pattern exists.

As opposed to the RTSS 2013 paper, this paper puts many more variables into the analysis of each job's execution. Jobs in this model are referred to by their criticality level, two functions of criticality level that detail the execution time allotted to the respective job in both normal and degraded mode, and period. An example of a task set was included in the paper, shown below:

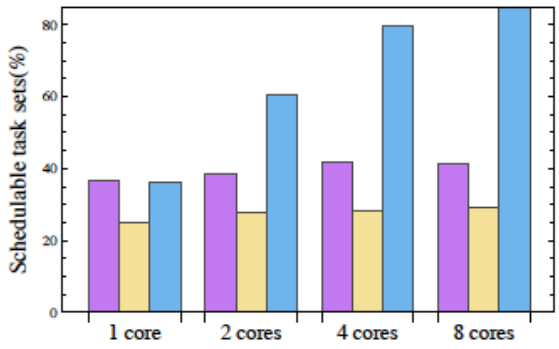
Table 1: Task set definition

τ_i	χ_i	W_i	$\mathbf{C}_i$	$C_{i,deg}$
τ_1	2	100	$C_1(1) = \{\{10, 14, 0, 0\}, \{0, 0, 20, 25\}, \{6, 8, 0, 0\}\}$ $C_1(2) = \{\{8, 30, 0, 0\}, \{0, 0, 15, 44\}, \{6, 12, 0, 0\}\}$	N/A
τ_2	2	50	$C_2(1) = \{\{8, 10, 0, 0\}, \{0, 0, 15, 18\}, \{1, 2, 0, 0\}\}$ $C_2(2) = \{\{5, 12, 0, 0\}, \{0, 0, 15, 20\}, \{1, 4, 0, 0\}\}$	N/A
τ_3	1	50	$C_3(1) = \{\{4, 5, 0, 0\}, \{0, 0, 6, 8\}, \{2, 4, 0, 0\}\}$ $C_3(2) = C_{3,deg}$	$C_{3,deg} = \{\{2, 2, 0, 0\}, \{0, 0, 2, 3\}, \{1, 2, 0, 0\}\}$
τ_4	1	200	$C_4(1) = \{\{10, 10, 0, 0\}, \{0, 0, 20, 20\}, \{10, 10, 0, 0\}\}$ $C_4(2) = C_{4,deg}$	$C_{4,deg} = \{\{0, 0, 0, 0\}, \{0, 0, 0, 0\}, \{0, 0, 0, 0\}\}$

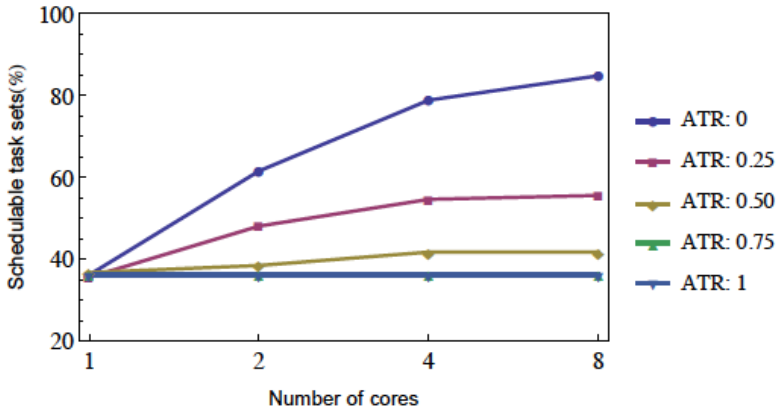
The model takes after cyclic executive in the sense that scheduling tables are created that separate jobs into frames, and then further into subframes. Barriers are created using special functions that identify where the worst-case execution time would be in the frame. These barriers are calculated using special interference analysis and are also used in the calculation of whether or not a system can change states, which vary based off of cost. A simulation was set up that would check on three criteria after setting up all of the constraints. A total of H/W_i jobs are scheduled for every job_i, all iterations of the same task are scheduled on the same processor, and all jobs honor their predecessor constraints. The simulator is designed so that if any of those three conditions, then the simulator will stop searching with the given tasks.

One detail that is then considered is that preemption should not be allowed as it would be practically impossible to accurately predict and simulate—as well as to implement. The idea here is that if a job is preempted and then shifted towards the end of the execution frame, it would then block other lower criticality jobs from running. Instead of blocking preemption altogether, a happy medium is utilized so that jobs can preempt other jobs at certain points in the schedule. This will allow both accurate analysis and description of the task-set and will also allow flexible run-time adjustment in the schedules.

Once all of the different constraints are accounted for, many simulations are run. Firstly, three different schemes of the scheduling algorithm are considered and plotted on many different cores in the system. Three variations include: Time-Triggered and Synchronized (TTS) subframes that are dynamically initialized and with core interference (purple), TTS subframes that are dynamically initialized with core interference (tan), and finally TTS subframes are dynamically initialized and core interference is not accounted for (blue). The following chart shows the results of the simulation that results in schedulable tasks with each scheme on 1, 2, 4, and 8 cores:



It can be seen that not considering the core interference among tasks results in a much higher schedulability percentage but does not necessarily represent the most accurate model (or would either require more complex programming to achieve). After this, the next comparison is used with resource sharing, and how much time should be spent doing calculations versus accessing memory. A ratio is used to model if a task would only access memory (ATR = 1), would only do calculations (ATR = 0), or anywhere in between. The following graph shows the results from this comparison.



This comparison shows a general trend of decreasing schedulability with increasing memory access time. Finally, different combinations of EDF-VD, preemption, and the TTS scheduling were compared against system utilization to see which scheme would be more viable. While in some cases EDF was comparable to TTS, in the scheme with no preemption allowed, TTS was acutely more schedulable at higher utilizations. The paper concludes by implementing the scheme into a real-world industrial application and determining if it is a practical method for scheduling tasks outside of a computer simulation.

Interpretation

This paper dives deep into the concept of resource sharing and planning on how to share the processor and memory so that all jobs can execute fully, even if all of the resources are already allocated. Many different comparisons were made that showcased the efficiency and versatility of the scheduling algorithm, as well as showed that there are certain points in which the model could not be helpful with (i.e. after a certain quantity of cores on certain applications there is no benefit in adding more). The scheme provided is comparable, if not better than EDF in just about all of the calculations given, showing that it is an extremely useful tool.

Further Exploration

While this paper did a very thorough job at researching resource allocation, there were a few details that I believe were neglected. Firstly, only two levels of criticality were ever used to simulate the different tests, and so it could be that for two levels of criticality the TTS scheduling is paramount but could fail many tests or needs more constraints once more criticality levels are added. Secondly, this paper doesn't consider multiple processor speeds, as mentioned in the RTSS 2013 paper, and assumes that the processor speed will be practically constant. Finally, it was not explicitly stated as to how aperiodic tasks would be handled. While the a priori tables do allow time for the aperiodic jobs to execute, it is unclear about how they would be handled, and whether or not they could contend for resources and slow down the other jobs running.

IV. Conclusion

Throughout the course of my investigation, I explored multiple ideas on how to tackle mixed criticality scheduling and the barriers associated with that. In the first paper I encountered, the object of the research was to inquire about scheduling based off of variable processor speeds. The matter in question revolves around how the processor may run into interrupts or exceptions that would cause it to slow down, and so certain, higher-priority jobs would need to have the guarantee of finishing at both speeds. This was accomplished by dropping lower criticality jobs so that the slower execution speed would have more time to execute the higher-level jobs. While this paper does an excellent job of finding a correlation between the slowest speed that can guarantee correctness, it does not take into account resource sharing and multicore scheduling. The next paper picked up where the other one left off, in a manner of speaking.

The second paper addresses the complexities of resource sharing and multicore scheduling. Many models, calculations, estimations and checks were done in order to calculate the schedulability of jobs that were randomly generated. Its findings were that the time-triggered and synchronized scheduling that followed a cyclic executive frame-oriented approach was actually quite useful and accurate with scheduling various tasks. Compared to EDF and plotted against a plethora of variables the paper tracked its productivity and found that the method had few shortfalls. The issue with this idea is that it assumes a variable processor speed, and also takes advantage of not having aperiodic tasks, which will not always be the case.

I am curious if there is a way to combine these two schemes. The first method employs a unique scheduling plan for the normal speed and then goes to EDF when it is in degraded mode. I speculate that if the TTS frame scheduling was adapted to mirror, or even pseudo-mirror the normal speed schedules, and then go into the full TTS schedule when the processor goes into degraded mode, the combinatorial scheme could accommodate both processor speed constraints as well as multicore scheduling and resource distribution problems.

V. References

1. Baruah, S., & Guo, Z. (2013, December). Mixed-criticality scheduling upon varying-speed processors. In *2013 IEEE 34th Real-Time Systems Symposium* (pp. 68-77). IEEE.
2. Giannopoulou, G., Stoimenov, N., Huang, P., & Thiele, L. (2013, September). Scheduling of mixed-criticality applications on resource-sharing multicore systems. In *Proceedings of the Eleventh ACM International Conference on Embedded Software* (p. 17). IEEE Press.