Mixed Criticality Systems - A Review*

Alan Burns and Robert I. Davis
Department of Computer Science,
University of York, York, UK.
email: {alan.burns, rob.davis}@york.ac.uk

Abstract

This review covers research on the topic of mixed criticality systems that has been published since Vestal's 2007 paper. It covers the period up to and including December 2016. The review is organised into the following topics: introduction and motivation, models, single processor analysis (including job-based, hard and soft tasks, fixed priority and EDF scheduling, shared resources and static and synchronous scheduling), multiprocessor analysis, related topics, realistic models, formal treatments, and systems issues. An appendix lists funded projects in the area of mixed criticality.

*Ninth edition, Jan 2017.

Contents

1 Introduction **3**

2 Mixed Criticality Models **5**

3 Single Processor Analysis **7**
- 3.1 Job scheduling . . . 7
- 3.2 Fixed Priority Scheduling . . . 7
  - 3.2.1 RTA-Based approaches . . . 7
  - 3.2.2 Slack scheduling . . . 11
  - 3.2.3 Period transformation . . . 12
- 3.3 EDF Scheduling . . . 14
- 3.4 Shared Resources . . . 15
- 3.5 Static and Synchronous Scheduling . . . 16
- 3.6 Varying Speed Processors . . . 17

4 Multiprocessor Analysis **17**
- 4.1 Task Allocation . . . 18
- 4.2 Schedulability Analysis . . . 19
- 4.3 Communication and other Resources . . . 20

5 Links to other Research Topics **24**
- 5.1 Hard and Soft Tasks . . . 24
- 5.2 Fault Tolerant Systems . . . 25
- 5.3 Hierarchical Scheduling . . . 26
- 5.4 Cyber Physical Systems and Internet of Things . . . 27
- 5.5 Probabilistic real-time systems . . . 27

6 More Realistic MCS Models **29**

7 More Formal Treatments **32**
- 7.1 Utilisation Bounds . . . 32
- 7.2 Speedup Factors . . . 32
- 7.3 Formal Language and Modelling Issues . . . 33

8 Systems Issues **34**

9 Conclusion **38**

10 Appendix - Projects **40**

1 Introduction

An increasingly important trend in the design of real-time and embedded systems is the integration of components with different levels of criticality onto a common hardware platform. At the same time, these platforms are migrating from single cores to multi-cores and, in the future, many-core architectures. Criticality is a designation of the level of assurance against failure needed for a system component. A mixed criticality system (MCS) is one that has two or more distinct levels (for example safety critical, mission critical and low-critical). Perhaps up to five levels may be identified (see, for example, the IEC 61508, DO-178B and DO-178C, DO-254 and ISO 26262 standards). Typical names for the levels are ASILs (Automotive Safety and Integrity Levels), DALs (Design Assurance Levels or Development Assurance Levels) and SILs (Safety Integrity Levels). It should be noted that not all standards and papers on MCS assign to 'criticality' the same meaning, an issue explored by Graydon and Bate [161], Esper et al. [138], Paulitsch et al. [292] and Ernst and Di Natale [137].

Most of the complex embedded systems found in, for example, the automotive and avionics industries are evolving into mixed criticality systems in order to meet stringent non-functional requirements relating to cost, space, weight, heat generation and power consumption (the latter being of particular relevance to mobile systems). Indeed the software standards in the European automotive industry (AUTOSAR[1]) and in the avionics domain (ARINC[2]) address mixed criticality issues; in the sense that they recognise that MCS must be supported on their platforms.

The fundamental research question underlying these initiatives and standards is: how, in a disciplined way, to reconcile the conflicting requirements of *partitioning* for (safety) assurance and *sharing* for efficient resource usage. This question gives rise to theoretical problems in modeling and verification, and systems problems relating to the design and implementation of the necessary hardware and software run-time controls.

A key aspect of MCS is that system parameters, such as tasks' worst-case execution times (WCETs), become dependent on the criticality level of the tasks. So the same code will have a higher WCET if it is defined to be safety-critical (as a higher level of assurance is required) than it would if it is just considered to be mission critical or indeed non-critical. This property of MCS significantly modifies/undermines many of the standard scheduling results. This report aims to review the research that has been published on MCS.

The first paper on the verification of a Mixed Criticality System used an exten-

[1] http://www.autosar.org/

[2] http://www.arinc.com/

sion of standard fixed priority (FP) real-time scheduling theory, and was published by Vestal (of Honeywell Aerospace) in 2007 [356][3]. It employed a somewhat restrictive work-flow model, focused on a single processor and made use of Response Time Analysis [19]. It showed that neither rate monotonic [254] nor deadline monotonic [241] priority assignment was optimal for MCS; however Audsley's optimal priority assignment algorithm [17] was found to be applicable.

This paper was followed by two publications in 2008 by Baruah and Vestal [59], and Huber et al. [206]. The first of these papers generalises Vestal's model by using a sporadic task model and by assessing fixed job-priority scheduling and dynamic priority scheduling. It contains the important result that EDF (Earliest Deadline First) does not dominate FP when criticality levels are introduced, and that there are feasible systems that cannot be scheduled by EDF. The latter paper addresses multi-processor issues and virtualisation (though it did not use that term). It focused on AUTOSAR and resource management (encapsulation and monitoring) with time-triggered applications and a trusted network layer.

Further impetus to defining MCS as a distinct research topic came from the white paper produced by Barhorst et al. [26], the Keynote talk that Baruah gave at the 2010 ECRTS conference[4] and a workshop report from the European Commission [350]. These have been followed up by tutorials on MCS at ESWEEK in 2012 and 2013[5], a workshop at HiPEAC in January 2013[6], a workshop (WICERT) at DATE 2013[7], a workshop (ReTiMiCS) at RTCSA 2013[8], workshops (WMC) at RTSS 2013[9], RTSS 2014[10], RTSS 2015[11] and RTSS 2016[12]; a workshop at the 19th International Conference on Reliable Software Technologies (Ada-Europe) in June 2014, and a full Dagstuhl School on Mixed Criticality and Many Core Platforms in March 2015[13].

This review [84] is organised as follows. In Section 2 we first consider mixed

[3]The term Mixed Criticality had been used before 2007 to address issues of non-interference in non-federated architectures such as IMA [190]; Vestal changed the focus of research by concentrating on real-time performance. Systems with more than one criticality level but aim to only give complete isolation are called *multiple-criticality systems*; the use of *mixed-criticality* implies some tradeoff between isolation and integration that involves resource sharing.

[4]Available from the conference web site: http://ecrts.eit.uni-kl.de/index.php?id=53.

[5]Embedded Systems Week: http://www.esweek.org/

[6]http://www.hipeac.net/conference/berlin/workshop/integration-mixed-criticality-subsystems-multi-core-processors

[7]http://atcproyectos.ugr.es/wicert/index.php/conference-proceedings

[8]http://igm.univ-mlv.fr/rtalgo/Events/RETIMICS/

[9]http://www.cs.york.ac.uk/ robdavis/wmc2013/

[10]http://www.cs.york.ac.uk/ robdavis/wmc2014/

[11]http://www.cs.york.ac.uk/ robdavis/wmc/

[12]https://gsathish.github.io/wmc2016/

[13]http://www.dagstuhl.de/15121

criticality models. Then in Section 3 single processor systems are covered (including fixed priority and EDF scheduling). Section 4 covers multiprocessor issues and Section 5 links this research to other topics such as hard and soft real-time scheduling and hierarchical scheduling. More realistic models are covered in Section 6, more formal work is covered in Section 7 and systems work is covered in Section 8. Conclusions are presented in Section 9. A short review of existing projects related to MCS is contained in an appendix.

2 Mixed Criticality Models

Inevitably not all papers on mixed criticality have used the same system or task model. Here we define a model that is generally applicable and is capable of describing the main results considered in this review.

A system is defined as a finite set of components $\mathcal{K}$. Each component has a level of criticality (designated by the systems engineer responsible for the entire system), L, and contains a finite set of *sporadic tasks*. Each task, τ_i, is defined by its period (minimum arrival interval), deadline, computation time and criticality level: (T_i, D_i, C_i, L_i). Tasks give rise to a potentially unbounded sequence of *jobs*.

The primary concern with the implementation of MCS is one of separation. Tasks from different components must not be allowed to interfere with each other. In particular, mechanisms must be in place to prevent a job from executing for more than the computation time C defined for its task, and to ensure that a task does not generate jobs that are closer together than T[14].

The requirement to protect the operation of one component from the faults of another is present in all systems that host multiple applications. It is however of particular significance if components have different criticality levels. Since without such protection, all components would need to be engineered to the strict standards of the highest criticality level, potentially massively increasing development costs.

After concerns of partitioning comes the need to use resources efficiently. This is facilitated by noting that the task parameters are not independent, in particular the worst-case computation/execution time estimate, C_i, will be derived by a process dictated by the criticality level. The higher the criticality level, the more conservative the verification process and hence the greater will be the value of C_i. This was the observation at the heart of the paper by Vestal [356].

For systems executing on hardware platforms with deterministic behaviour, any particular task will have a single real WCET (worst-case execution time); however, this value typically cannot be known with complete certainty. This uncertainty is primarily epistemic (uncertainty in what we know, or do not know, about the

[14] Or (period minus release jitter) if that is part of the task model.

system) rather than aleatory (uncertainty in the system itself). Although it is reasonable to assume confidence increases (i.e. uncertainty decreases) with larger estimates of worst-case execution time, this may not be universally true [161]. It would certainly be hard to estimate what increase in confidence would result from, say, a 10% increase in all Cs.

For systems executing on hardware platforms with time-randomised hardware components [93], then a probabilistic WCET (pWCET) [12, 109, 117, 129] can be obtained. The exceedance function for this probability distribution defines for any specific probability, derived from a required maximum failure rate associated with a criticality level, an execution time budget which has no greater probability of being exceeded on any given run [108]. The pWCET distribution therefore effectively defines different estimates of the WCET budget for the same task, for different criticality levels due to their different requirements on the maximum tolerable failure rate.

The focus on different computation times was extended to task periods in subsequent papers [30, 34, 36, 43, 48, 79, 82, 373]. Here tasks are event handlers. The higher the criticality level the more events must be handled, and hence the task must execute more frequently even if it does not execute for longer.

In MCS a task is now defined by: ($\vec{T}$, D, $\vec{C}$, L), where $\vec{C}$ and $\vec{T}$ are vectors of values – one per criticality level, with the constraints:

$$L1 > L2 \Rightarrow C(L1) \geq C(L2)$$

$$L1 > L2 \Rightarrow T(L1) \leq T(L2)$$

for any two criticality levels $L1$ and $L2$.

Note the completion of the model, by making D criticality dependent [43] has not as yet been addressed in detail. But it would have the constraint:

$$L1 > L2 \Rightarrow D(L1) \geq D(L2)$$

So a task may have a 'safety critical' deadline and an early QoS deadline.

All three constraints have the property that a task can progress from satisfying both its $L1$ and $L2$ parameters to satisfying only its $L1$ constraints (for $L1 > L2$).

Another feature of many of the papers considered in this review is that the system is defined to execute in a number of criticality *modes*. A system starts in the lowest criticality mode. If all jobs behave according to this mode then the system stays in that mode. But if any job attempts to execute for a longer time, or more frequently, than is acceptable in that mode then a criticality mode change occurs. Ultimately the system may change to the highest criticality mode.

Some papers allow the criticality mode to move down as well as up, but others (indeed the majority) restrict the model to increases in criticality only. We return to this issue in Section 6.

Finally, many papers restrict themselves to just two criticality levels; high (HI) and low (LO) with $HI > LO$. These are referred to as *dual-criticality* systems. Where modes are used, the system is either in a LO-criticality mode or a HI-criticality mode. And the set of task parameters is typically: (T_i, D_i, $C_i(HI)$, $C_i(LO)$, L_i). At the other extreme from just two criticality modes are the models presented by Ekberg et al. [130, 132, 134] in which any number of modes are allowed and the movement between modes is represented by a directed acyclic graph.

3 Single Processor Analysis

Since Vestal's 2007 paper [356] there has been a series of publications. Most of these papers address single processor platforms and independent components.

3.1 Job scheduling

Initially a number of papers considered the restricted problem of scheduling, on a single processor, a finite set of mixed criticality jobs with criticality dependent execution times [29, 37, 39, 51, 53, 58, 167, 243, 244, 289, 328, 329, 333]. This work has, however, largely been superseded by work on the more widely applicable task model.

3.2 Fixed Priority Scheduling

In this section we look at MCS schemes that are based on applying Response-Time Analysis (RTA), then those that consider slack scheduling and finally approaches that are derived from period transformations.

3.2.1 RTA-Based approaches

Vestal's approach was formalised (i.e. proof that the use of Audsley's priority assignment algorithm [17] was optimal) by Dorin et al. [123] in 2010. They also extended the model to include release jitter, and showed how sensitivity analysis could be applied.

Vestal's approach allowed the priorities of high and low criticality tasks to be interleaved, but all tasks had to be evaluated as if they were of the highest criticality. By introducing monitoring of task execution time, and the prevention of execution

time over-runs, higher resource usage can be delivered [43]. This is a crucial issue in mixed criticality scheduling; by the introduction of more trusted components a high utilisation of the available resources is facilitated.

In 2011 this approach was further extended [46, 79] to give a scheduling model and associated analysis framework for a single processor system that dominates all previous published analysis for MCS (using fixed priority scheduling) in that it made better use of the processor and could schedule all systems that could be guaranteed by other approaches, plus many that could not. These papers were however restricted to just two criticality levels (or modes). The system's run-time behaviour is either *low-criticality* (which relies on all execution times being bound by the low-criticality values and guarantees that all deadlines are met) or *high-criticality* (where only high criticality work is guaranteed but the rely condition[15] is weakened – the bound on high-criticality execution times is increased). The system's criticality change (from Low to High, i.e. LO to HI) is triggered by the observation, at run-time, that the stronger rely condition has been violated.

This change in criticality level has a number of similarities to systems that move between different operational modes (although there are also some significant differences [77, 161]). In the HI-criticality mode there are fewer tasks, but they have longer execution times or shorter periods. The literature on mode change protocols [24, 87, 135, 293, 305, 324, 351, 352], however, highlights one important problem: a system can be schedulable in every mode, but not schedulable during a mode change [352]. This is also true for systems that change criticality levels.

An optimal priority ordering is defined in the paper from Baruah et al. [46] in that it maximises the priority of high criticality tasks, subject to the system being schedulable. Both the high and low criticality tasks are ordered via deadline (deadline monotonic) and a simplified version of Audsley's algorithm is used to assign priorities from the lowest to the highest level. At each priority level the lowest priority task from the low criticality task set is tried first, if it is schedulable then the algorithm moves? up to the next priority level; if it is not schedulable then the lowest priority task from the high criticality set is tested. If it is schedulable then again the algorithm moves on to the next level. But if neither of these two tasks are schedulable then the search can be abandoned as the task set is unschedulable. In total a maximum of 2N tests are needed (where N is the number of tasks in the system)[16]. Note that this result follows from work on robust priority assignment [115]. As each set of LO/HI criticality tasks can be viewed as additional interference on the other subset, an optimal priority ordering can be obtained with each subset in

[15] A rely condition formalises the assumptions required for the guarantees to be valid [216].

[16] Strictly, only 2N-1 tests are needed as the highest priority task must be schedulable as its computation time is less than its deadline.

Deadline Monotonic priority order and a merge operation between them.

The protocol (dropping all LO-criticality work if any task executes for more than its $C(LO)$ value[17]), the derived analysis and the use of optimal priority ordering is shown [46] to out-perform other schemes (in terms of success ratio for randomly generated task sets). The analysis is based on standard RTA (Response-Time Analysis). For any task, τ_i, first its LO-criticality response-time ($R(LO)$) is computed using LO-criticality parameters for all the tasks. A criticality switch must occur before this value if the task is to be impacted by the change, otherwise it will have completed execution. The worst-case response-time in the HI-criticality mode ($R(HI)$) is computed by noting that all LO-criticality tasks must be abandoned by time $R(LO)$. The paper contains two methods for computing $R(HI)$, one involves a single upper bound, the other looks at all the possible critically change points before $R(LO)$ and computes the worst-case. The latter is more accurate, though still not exact; however, the gain is not significant and the simple upper bound test is probably sufficient in most cases.

To illustrate the above approaches one of the graphs from [46] is reproduced in Figure 1. This figure plots the percentage of task sets generated that were deemed schedulable for a system of 20 tasks, with on average 50% of those tasks having high criticality and each task having a high criticality execution time that is twice its low criticality execution time. The compared approaches are (from least effective to most effective): CrMPO which assigned priorities in criticality order, SMC-NO (static mixed criticality with no run-time monitoring) which is Vestal's original approach, SMC which is an adaptation of Vestal's approach in which LO-criticality tasks are monitored at run-time and are prevented from executing for more than $C(LO)$, and AMC-rtb and AMC-max which are the two methods introduced in the previous paragraph (AMC for adaptive mixed criticality). In the graph the UB-H&L line bounds the maximum possible number of schedulable task sets. It serves to illustrate the quality of the AMC-max approach.

The AMC-rtb approach was extended by Zhao et al [376, 377] in 2013 to incorporate preemption thresholds [312] into the model. They demonstrated a reduction in stack usage and improved performance for some parameter ranges. Another approach to combining AMC-rtb and existing scheduling theory is taken by Burns and Davis [83]. They consider the use of deferred preemption [75, 112] and demonstrate a significant improvement over fully preemptive AMC-rtb. The gain in schedulability they demonstrate is obtained by having a final non-preemptive region (FNPR) at the end of each criticality (e.g. at the end of $C(LO)$ and $C(HI)$), and by combining the assignment of priority and the determination of the size of this FNPR.

[17] First proposed by Baruah [29, 37].

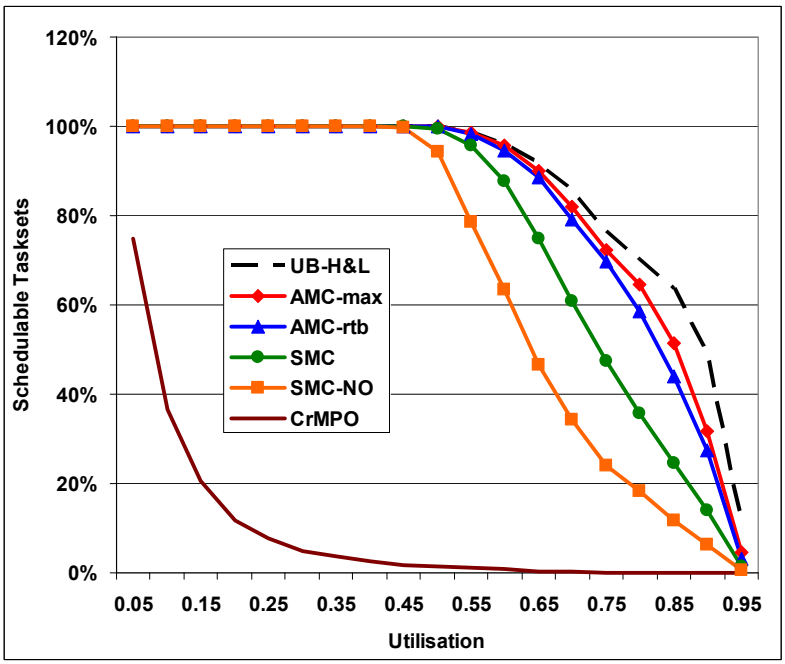

Figure 1: Percentage of Schedulable Task Sets

In keeping with a number of papers on MCS, the work of Baruah et al. [46] (and most of the subsequent modifications) restricted itself to dual criticality systems. Fleming and Burns [142] extended these models to an arbitrary number of criticality levels, focusing particularly on five levels as this is the maximum found in automotive and avionics standards. They observed that AMC-rtb remains a good approximation to AMC-max, and that AMC-max became computational expensive for increased levels. They concluded that AMC-rtb represented an adequate and effective form of analysis. A relatively minor improvement to AMC-max was published by Huang et al. [197] (they termed it AMC-IA); however there may be cases where their analysis is optimistic[18].

One characteristic of all the schemes defined above is that tasks do not change their priority after a criticality mode change. If priorities can change then a simple form of sufficient analysis is possible [47]. This work defines a new approach, PMC (priority may change). Evaluations show that PMC performs similarly to AMC-rtb, though neither dominates the other. An improved scheme, GFP (Generalised Fixed Priority) is proposed by Chen et al. [96]. They assign (using an

[18]This is the topic of on-going discussions.

heuristic) three priorities to each task. One for each of the two criticality levels, and one for the transition between the criticality modes. They demonstrate an improvement over AMC-rtg.

It was noted in the section on Mixed Criticality Models that the period parameter (T) can be criticality dependent as well as the worst-case execution time estimate (C). An application may consist of event handlers, and have different levels of constraint over the arrival patterns of the events. The higher the criticality, the closer together the events are assumed to arrive; and hence the smaller the T parameter. Baruah and Chattopadhyay [48] have reformulated the SMC and AMC analysis (introduced above) to apply to this model, in which the Ts rather than the Cs vary with criticality. Their evaluation results show similar behaviour to that depicted in Figure 1. Criticality specific periods are also address by Burns and Davis [82], Baruah [36], and by Zhang et al. [373] (who derived an improved analysis that they termed SAMC – Sufficient AMC).

3.2.2 Slack scheduling

An alternative approach to scheduling mixed criticality fixed priority systems is, for two level systems, to use a *slack scheduling* scheme in which low criticality jobs are run in the slack generated by high criticality jobs only using their low criticality execution budgets. This was first explored by Niz et al. [279]. One difficulty with this approach is to incorporate sporadic tasks. At what point can the 'slack' of a non-appearing sporadic task be allocated to low criticality jobs? Even for periodic tasks, ensuring schedulability of high criticality tasks in all circumstances is not straightforward. Niz et al. [279] compute the time at which a high criticality task must be released to ensure that it meets its deadline (a scheme similar to the dual-priority approach outlined in Section 5.1). However, Huang et al. [196] demonstrated that if a low criticality (high priority) task executes beyond its deadline, a high criticality (lower priority) task could miss its deadline. They show that either the low criticality task must be aborted at its deadline or (more practically) its priority must be reduced to a background level. They then derive safe analysis. Niz et al. have subsequently clarified their model (to remove the problem) and improved its performance [280, 281].

While slack is usually generated by tasks not executing for their full budget, it is also produced by the arrival of jobs being less frequent than anticipated in the worst-case. Neukirchner et al. [274, 275] adapt and extend a number of schemes for monitoring activation patterns. Their multi-mode approach is proved to be safe (no false negatives) and efficient (few false positives). Hu et al. [195] also consider budget management, and produce an effective scheme for minimising the overheads associated with slack management.

For a dual-criticality system $C(LO)$ values must, of course, be known. Once schedulability has been established however, it is possible to derive [317], using sensitivity analysis [67, 301], a scaling factor F ($F > 1$) such that the system remains schedulable with all $C(LO)$ values replaced by $F \cdot C(LO)$. Using these scaled values at run-time will increase the robustness of the system, as the LO-criticality tasks will be able to execute for a greater time before a criticality change is triggered. Scaling can also be applied to the $C(HI)$ values. Volp et al. [359] look at an alternative means of obtaining $C(LO)$ and $C(HI)$ values; they do not consider them to be estimates of worst-case execution time, but budgets set by some design optimisation process.

As scaling involves changing a task's computation time, and computation time influences priority assignment, it is possible to extend this approach by also allowing priorities to change as the system is made more robust [80]. A more dynamic budget management scheme is used by Gu and Easwaran [169] to postpone criticality level mode changes.

Sciandra et al. [320] are extending and applying scaling factors to intelligent transport systems. Issues of robustness are also addressed by Herman et al. [186].

3.2.3 Period transformation

As Vestal noted [356], an older protocol *period transformation* [322, 323] (PT), is also applicable to the mixed critically scheduling problem. Period transformation splits a task with period T and computation time C into two (or more) parts so that the task now has the parameters $T/2$ and $C/2$. Assuming all tasks have deadlines equal to their periods, the application of the optimal rate monotonic priority assignment scheme [254] will increase the relative priority of all transformed tasks. If all high criticality tasks are transformed so that their transformed periods are shorter than all low criticality tasks then the rate monotonic algorithm will deliver partitioned (i.e. criticality monotonic) priorities. All high criticality tasks will have priorities greater than all lower criticality tasks. The scheme can easily be extended to task sets with constrained deadlines ($D < T$). However, the scheme does introduce extra overheads from the increased number of context switches, and these could be excessive if there are low criticality tasks with short deadlines. A simple example of a period transformed task would be one with $T = D = 16$, $C(HI) = 8$ and $C(LO) = 4$; this task could be transformed to one with $T = D = 4$ and $C = 2$ Note, this is $C(HI)/4$, not $C(LO)/4$). The computation time is such that if the task executes according to its HI-criticality parameter it will take four invocations of the transformed task to complete, but if the LO-criticality assumption is valid it will only take two.

If overheads are ignored then Period Transformation performs well. Baruah

and Burns postulate [44] (and prove for two tasks) that this is primarily due to the inherent property of PT to deliver tasks sets with harmonic periods (that are then more likely to be schedulable). It does not seem that PT is of specific benefit to MCS.

To split the code of a task, either a static code transformation process must be used or the run-time must employ an execution-time server. With code transformation, the programmer must identify where in the code the split should be made. This does not lead to good code modulisation and is similar to the problems encountered when functions must to be split into short sections so that they can be 'packed' into the minor cycles of a cyclic executive [88]. There is also the problem of OS locks being retained between slices of the code; making the protected resource unavailable to other tasks.

With a dual-criticality task such as the one in the example above the point at which the task can be assumed to have executed for two units of time is itself criticality dependent. This to all intents and purposes makes code transformation impractical. Therefore, if the code is not to be changed then a run-time server must be used to restrict the amount of computation allowed per release of the (transformed) task. In practice this means that:

- Without PT, LO-criticality tasks may have high priorities and hence their execution times must be monitored (and enforced); HI-criticality tasks must also be monitored as they may need to trigger a criticality change if they execute for more than $C(LO)$ thereby triggering the abandonment of LO-criticality tasks.

- With PT, LO-criticality tasks have the lower priorities and hence they do not need to be monitored, HI-criticality tasks must be monitored to enforce the per release budget.

In general, there is less run-time intervention with PT. But recall there is considerably more task switching overhead if the periods of all HI-criticality tasks are reduced to less than all LO-criticality task periods.

For multiple criticality levels a number of transformations may be required to generate a criticality monotonic ordering [142]. For example if there are three tasks (H, M, and L) with criticality levels implied by their names, and periods 5, 33 and 9. Then first M must be divided by 11 to get a period of 3 (so less than 9), but then H must be divided by 5 to move it below the new value for M. As a result the transformed periods become 1, 3 and 9. It also seems that the theoretical benefit of PT diminishes with an increased number of criticality levels [142].

3.3 EDF Scheduling

The first paper to consider MCS with EDF scheduling was Baruah and Vestal [59] in 2008. Park and Kim [289] later introduced a slack-based mixed criticality scheme for EDF scheduled jobs which they called CBEDF (Criticality Based EDF). In essence they use a combination of off- and on-line analysis to run HI-criticality jobs as late as possible, and LO-criticality jobs in the generated slack. In effect they are utilising an older protocol developed by Chetto and Chetto [97] for running soft real-time tasks in the 'gaps' produced by running hard real-time tasks so as to just meet their deadlines.

A more complete analysis for EDF scheduled systems was presented by Ekberg and Yi [131, 172]. They mimicked the FP scheme by assigning two relative deadlines to each high criticality task. One deadline is the defining 'real' deadline of the task, the other is an artificial earlier deadline that is used to increase the likelihood of high criticality tasks executing before low criticality ones. At the point that the criticality of the system changes from low to high (due to a task exceeding its low criticality budget), all low criticality tasks are abandoned and the high criticality tasks revert to their defining deadlines. They demonstrate a clear improvement over previous schemes [173]. Later work [132] generalises the model to include changes to all task parameters and to incorporate more than two criticality levels. Tighter analysis is provided by Easwaran [126], although it is not clear that the method will scale to more than two criticality levels. Further improvements are presented by Yao et al. [366]. They use an improved schedulability test for EDF (a scheme called QPA [372]), and a genetic algorithm (GA) to find better artificial deadlines.

A similar scheme was presented by Baruah et al. [40,42], called EDF-VD (EDF - with virtual deadlines). Again for a dual-criticality system, HI-criticality tasks have their deadlines reduced (if necessary) during LO-criticality mode execution. All deadlines are reduced by the same factor. They demonstrate both theoretically and via evaluations that this is an effective scheme. Note, however, that this scheme is not as general as those reported above [126, 131, 172]. In these approaches a different reduction factor is used for each task. Nevertheless the use of a single value does allow schedulability bounds to be derived (see Section 7). An intermediate approach that uses just two scaling factors is provided by Masrur et al. [263]; there motivation being to develop an efficient scheme that could be used at run-time. In later work [60] Baruah has generalised the underlying MCS model to include criticality-specific values for period and deadline as well as WCET.

EDF scheduling of MCS is also addressed by Lipari and Buttazzo [253] using a reservation-based approach. Here sufficient budget is reserved for the high criticality tasks, but if they only make use of what is assumed by their low critical-

ity requirements then a set of low criticality tasks can be guaranteed. Again only two criticality levels are assumed. In effect low criticality tasks run in capacity reclaimed from high criticality tasks. Deadlines for the high criticality tasks are chosen to maximise the amount of capacity reclaiming.

A different approach to using spare capacity was derived by Su at al. [339,340] by exploiting *elastic task model* [90] in which the period of a task can change. They propose a minimum level of service for each LO-criticality task τ_i that is defined by a maximum period, T_i^{max}. The complete system must be schedulable when all HI-criticality tasks use their $C(HI)$ values and all LO-criticality tasks use their $C(LO)$ and T^{max} values. At run-time if HI-criticality tasks use less than their full HI-criticality entitlement then the LO-criticality tasks can run more frequently. They demonstrate that for certain parameter sets their approach performs better than EDF-VD.

3.4 Shared Resources

With mixed criticality systems it is not clear to what extent data should flow between criticality levels. There are strong objections to data flowing from low to high criticality applications unless the high criticality component is able to deal with potentially unreliable data [321] – this happens with some security protocols [66]. Even with data flowing in the other direction there remains the scheduling problem of not allowing a high criticality task to be delayed by a low criticality task that has either locked a shared resource for longer than expected or is executing at a raised priority ceiling level for too long.

Sharing resources within a criticality level is however a necessary part of any usable tasking model. In single criticality systems a number of priority ceiling protocols have been developed [25, 325]. These are beginning to be assessed in terms of their effectiveness for mixed criticality systems. Burns [76] extends the analysis for fixed priority systems by adding criticality specific blocking terms into the response-time analysis. He notes that the original form of the priority ceiling protocol (OPCP) [325] has some useful properties when applied to MCS. Resources can be easily partitioned between criticality levels and starvation of LO-criticality tasks while holding a lock on a resource can be prevented. With AMC-OPCP, a task can only suffer direct blocking if a resource is locked by a lower priority task of the same criticality.

Rather than use a software protocol, Engel [136] employs Hardware Transactional Memory to roll back any shared object to a previous state if a LO-criticality task overruns its budget while accessing the object.

For EDF-based scheduling Zhao et al. [376,379] attempt to integrate the Stack Resource Protocol (SRP) [25] and Preemption Threshold Scheduling [362] with

approaches to EDF scheduling that involve tasks having more than one deadline. This is not straightforward as these schemes assume that relative deadlines are fixed.

Alternative approaches are proposed by Lakshmanan et al. [231] by extending their single processor zero slack scheduling approach [279] to accommodate task synchronisation across criticality levels for fixed priority systems. They define two protocols: PCIP (Priority and Criticality Inheritance Protocol) and PCCP (Priority and Criticality Ceiling Protocol). Both of these contain the notion of criticality inheritance. This notion is also used by Zhao et al. [378] in their HLC-PCP (Highest-Locker Criticality Priority Ceiling Protocol) which they apply to the AMC scheduling scheme (see Section 3.2.1). For a dual criticality system they define three modes of execution, the usual two plus an intermediate mode which covers the time during which LO-criticality tasks are allowed to continue to execute if they are holding a lock on a resource that is shared with a HI-criticality task.

A more systematic scheme is proposed by Brandenburg [72]. Here all shared resources are placed in *resource servers* and all access to these servers is via a MC-IPC protocol. As a result only these servers and the support for the MC-IPC protocol have to be developed to the highest criticality level. Resource users can be of any criticality level, including non-critical. Data sharing within the context of the MC^2 architecture (see Section 4) is address by Chisholm et al. [98].

3.5 Static and Synchronous Scheduling

The move between criticality levels can be captured in a static schedule by switching between previously computed schedules; one per criticality level. This is explored by Baruah and Fohler [52]. Socci et al. [331, 333] show how these Time-Triggered (TT) tables can be produced via first simulating the behaviour one would obtain from the equivalent fixed priority task execution. Construction of the tables via tree search is addressed by Theis et al [346], and via the use of linear programming (LP) by Jan et al. [212]. For legacy systems Theis and Fohler [345] show how an existing single table may be used to support MCS. A particularly simple table driven approach is to use a cyclic executive, this is investigated by Burns et al. [45, 81, 85, 141, 144, 145] for multiprocessor systems in which the change from minor cycle to minor cycle is synchronised as is the change from executing code of one criticality to that of another. Both global and partitioned approaches are investigated, as are systems that use less processors for the HI-criticality work than they do for the LO-criticality work [145].

This use of tables is extended to synchronous reactive programs by Baruah [31, 32]. Here a DAG (Directed Acyclic Graph), of basic blocks that execute accord-

ing to the synchrony assumption, is produced that implements a dual-criticality program. The synchronous approach is also considered by Yip et al. [367] and by Cohen et al. [104]. The latter proving an application of mixed criticality from the railway industry, and an example of why data needs to flow between criticality levels.

3.6 Varying Speed Processors

Most analysis for MCS assumes a constant speed processor, but there are situations in which the speed of the processor is not known precisely (for example with asynchronous circuitry). Baruah and Guo [54] consider power issues that could lead to a processor having variable speed. As a processor slows down the execution time of the tasks increase. They simplify the model by assuming two basic speeds, normal and degraded. At the normal speed a scheduling table is used; at the degraded speed only HI-criticality jobs are executed and they use EDF. They have extended this work [55, 174, 175] to include a more expressive model, issues of processor self-monitoring (or not), and a probabilistic approach to performance variation. They have also considered system which have both uncertainty in execution times and processor speed [176].

Voltage scheduling, and thereby variable speed computation, is used by Huang et al. [201, 202] to respond to an temporal overload – if a $C(HI)$ value is exceeded and could lead to a LO-criticality task missing its deadline then energy is utilised to enable the processor to reduce computation times. Overall, their approach aims to reduce the system's expected energy consumption. DVFS management is also addressed by Haririan and Garcia-Ortiz [182] in their provision of a simulation framework for power management.

4 Multiprocessor Analysis

The first paper to discuss mixed criticality within the context of multiprocessor or multi-core platforms was by Anderson et al. [14] in 2009 and then extended in 2010 [269]. Five levels of criticality were identified; going from level-A (the highest) to level-E (the lowest). They envisaged an implementation scheme, which they call MC^2, that used a cyclic executive (static schedule) for level-A, partitioned preemptive EDF for level-B, global preemptive EDF for levels C and D and finally global best-effort for level-E. They considered only harmonic workloads but allowed slack to move between containers (servers). Each processor had a container for each criticality level, and a two-level hierarchical scheduler (see Section 5.3). Later work from this group [99, 186] evaluates the OS-induced overheads

associated with multiprocessor platforms. They also experimented with isolation techniques for LLC (last level cache) and DRAM. And have demonstrated, using MC^2, the benefits of having different isolation techniques for each criticality level [222].

This MC^2 framework is also used by Bommert [71] to support segmented mixed criticality parallel tasks.

In the remainder of this section we first look at task allocation (with global or partitioned scheduling), then consider analysis and finally communications and other systems resources. We note that there has also been work on implementing mixed-criticality synchronous systems on multiprocessor platforms [33].

4.1 Task Allocation

The issue of allocation was addressed by Lakshmanan et al. [232] by extending their single processor slack scheduling approach [279] to partitioned multiprocessor systems employing a Compress-on-Overload packing scheme. Allocation in a distributed architecture was addressed by Tamas-Selicean and Pop [341–344] in the context of static schedules (cyclic executives) and temporal partitioning. They observed that scheduling can sometimes be improved by increasing the criticality of some tasks so that single-criticality partitions become better balanced. This increase comes at a cost and so they employ search/optimisation routines (Simulated Annealing [148, 342] and Tabu [341, 343]) to obtain schedulability with minimum resource usage. Search routines, this time GAs (Genetic Algorithms), are also used by Zhang et al. [374] to undertake task placement in security-sensitive MCS. Their objective is to minimise energy consumption "while satisfying strict security and timing constraints". A toolset to aid partitioning is provided by Alonso et al. [11].

A more straightforward investigation of task allocation was undertaken by Kelly et al [220]. They considered partitioned homogeneous multiprocessors and compared first-fit and best-fit approaches with pre-ordering of the tasks based on either decreasing utilization or decreasing criticality. They used the original analysis of Vestal to test for schedulability on each processor, and concluded that in general first-fit decreasing criticality was best.

A comprehensive evaluation of many possible schemes is reported by Rodriguez at al. [308]. They consider EDF scheduling and used the analysis framework of EDF-VD (see Section 3.3). One of their conclusions is the effectiveness of a combined criticality-aware scheme in which HI-criticality tasks are allocated Worst-Fit and LO-criticality tasks are allocated using First-Fit; both with Decreasing Density. The same result is reported by Gu et al. [166]. They additionally note that if there are some very 'heavy' LO-criticality tasks (i.e. high utilisation or density) then space must be reserved for them before the HI-criticality tasks

are allocated. Partitioning with EDF-VD is also addressed by the work of Han et al. [179].

A global allocation scheme for MCS is proposed by Gratia et al. [158, 160]. They adapt the RUN scheduler [306], which uses a hierarchy of servers, to accommodate HI and LO criticality tasks. The latest version of their schedular (GMC-RUN) [159] has been extended to deal with more criticality levels.

Between fully partitioned and fully global scheduling is the class of schemes termed *semi-partitioned*. This is being addressed by Bletsas at al. [21, 68, 69] and Al-Bayati et al. [5]. The latter work uses two allocations for their two criticality modes. HI-criticality tasks do not migrate. During a mode change, carry-over LO-criticality jobs are dropped and new LO-criticality jobs executing on a different processor are given extended deadlines/periods (i.e. they utilise the elastic task model). A different approach is taken by Xu and Burns [365]; here a mode change in one processor results in LO-criticality jobs migrating to a different processor that has not suffered a criticality mode change. No deadlines are missed. If all processors suffer such a mode change then at least the timing needs of all HI-criticality tasks are protected.

With dual-criticality fault tolerant systems, a scheme in which high criticality tasks are replicated (duplicated) while low criticality tasks are not is investigated by Axer et al. [23] for independent periodic tasks running on a MPSoC (multi-processor system-on-chip). They provide reliability analysis that is used to inform task allocation.

A more theoretical approach (i.e. it is not directly implementable) is proposed by Lee et al. [236] with their MC-Fluid model. A fluid task model [192] executes each task at a rate proportional to its utilisation. If one ignores the cost of slicing up tasks in this way then the scheme delivers an optimal means of scheduling multiprocessor platforms. To produce a mixed criticality version of the fluid task model the fact that tasks do not have a single utilisation needs to be addressed. Lee et al. [236] do this and they also produce an implementable version of the model that performs well in simulation studies (when compared with other approaches). Baruah et al. [27, 50] derived a simplified fluid algorithm which they call MCF. Two further algorithms, MC-Sort and MC-slope, are proposed by Ramanathan and Easwaran [302, 303].

All the above work is focussed on standard single threaded tasks. In addition there has been some studies on parallel tasks and MCS – see Liu et al. [246, 256].

4.2 Schedulability Analysis

For globally scheduled systems Li and Baruah [245] take a 'standard' multiprocessor scheme, fpEDF [28] and combine it with their EDF-VD approach (see Section

3.3). Evaluations indicate that this is an effective combination. Extensions of this work [49] compare the use of partitioning or global scheduling for MCS. Their interim conclusion is that partitioning is by far the most effective approach to adopt.

Notwithstanding this result, Pathan derives [290] analysis for globally scheduled fixed priority systems. They adopt the single processor approach [46] (see Section 3.2) and integrate this with a form of analysis for multiprocessor scheduling that is amenable to optimal priority ordering (via Audsley's algorithm [17]). They demonstrate the effectiveness of their approach (by comparing success ratios).

A different and novel approach to multi-core scheduling of MCS is provided by Kritikakou et al. [226, 228]. They identify that a HI-criticality task will suffer interference from a LO-criticality task running on a different core due to the hardware platform's use of shared buses and memory controllers etc.. They monitor the execution time of the HI-criticality task and can identify when no further interference can be tolerated. At this point they abort the LO-criticality task even though it is not directly interfering. An implementation on a multi-core platform demonstrated effective performance of their scheme [228].

Extensions to deal with precedence constraints were given by Socci et al. [332] but only for jobs (not tasks).

4.3 Communication and other Resources

With a more complete platform such as a multiprocessor or System on Chip (SoC), perhaps with a NoC (Network-on-Chip), more resources have to be shared between criticality levels. The first design issue is therefore one of partitioning (as addressed above), how to ensure the behaviour of low criticality components does not adversely impact on the behaviour of higher criticality components. Pellizzoni et al. [294] in 2009 was the first to consider the deployment of mixed criticality systems (MCS) on multi-core and many-core platforms. They defined an Architectural Analysis and Design Language (AADL), a form of ADL (Architectural Description Language), for mixed criticality applications that facilitates system monitoring and budget enforcement of all computation and communication. Later Obermaisser et al. [285, 286] introduce a system model with gateways and end-to-end channels over hierarchical, heterogeneous and mixed criticality networks.

For a bus-based architecture it is necessary to control access to the bus so that applications on one core do not impact unreasonably on applications on other cores (whether of different or indeed the same criticality level). Pellizzoni et al. [295] show that a task can suffer a 300% increase in its worst-case execution time due to memory access interference even when it only spends 10% of its time on fetching from external memory on a 8-core system. To counter this, Yun et al. [368]

propose a memory throttling scheme for MCS. Kotaba et al [225] also propose a monitoring and control protocol to prevent processes flooding any shared communication media be it a bus or network. Kritikakou et al. [227] consider a scenario in which there are a few critical tasks that can suffer indirect interference from many lower critical tasks. They attempt to allow as much parallelism as possible - commensurate with the critical tasks retaining their temporal validity. Hassan and Patel [183] claim an improved bus arbitrator, called Carb, that is more criticality aware. Bounding the interference that a safety-critical task can suffer from lower criticality tasks using the same shared communication resources on a multi-core platform is also addressed by Nowotsch et al. [284].

Within the time-triggered model of distributed computation and communication a mixed criticality system is often viewed as one that has both time-triggered and event-triggered activities (also referred to as synchronous and asynchronous) [300, 335]. The time-triggered traffic is deemed to have the highest criticality, the event-triggered traffic can be either just best-effort or can have some level of assurance if its impact on the system is bounded; what Steiner [335] calls *rate-constrained*. Protocols that support this distinction can be supported on networks such as TTEthernet. Another TDMA-based approach, though this time built into the Real-Time Ethernet protocol, is proposed by Carvajal and Fischmeister in their open-source framework, *Atacama* [92]. Cilku at al. [102] describe a TDMA-based bus arbitration scheme. Novalk et al. [282] propose a scheduling algorithm for time-triggered traffic that minimises jitter while allowing HI-criticaltiy messages to be re-transmitted (following failure) at the expense of LO-critcality messages (which are abandoned). They also [283] consider how to produce an effective static schedule when there are unforeseen re-transmissions (for two and three levels of criticality).

A reconfigurable SDRAM controller is proposed by Goossen et al. [156] to schedule concurrent memory requests to the same physical memory. They also use a TDMA approach to share the controller's bandwidth. A key aspect of this controller is that it can adapt to changes in the run-time characteristics of the application(s). For example, a criticality mode change which should result in more bandwidth being assigned to the higher criticality tasks can be accommodated by what the authors call a *use-case switch*. Criticality aware DRAMs are also addressed by Jalle et al [211] in the context of a Space case study in which there are two criticality levels 'control' and 'payload'.

Virtual DRAMs are adapted by Ecco et al. [128] to isolate critical tasks (which are guaranteed) from non-critical tasks that, although not guaranteed, do perform adequately. Each virtual device supports one critical task and any number of non-critical tasks. All critical tasks run on dedicated cores, and hence the only potential source of inter-criticality interference is from the interconnection fabric (bus). By

use of virtual devices, the critical tasks benefit from interference-free memory access. DRAMs are also the focus of the work by Hassen et al [184] and Awan et al [20].

Kim at al. [221] propose a priority-based DRAM controller for MCS that separates critical and non-critical memory accesses. They demonstrate improved performance for the non-critical traffic. Note this work is focussed on supporting critical and non-critical traffic on the same memory banks (rather than mixed-criticality). A similar approach and result is provided by Goossens et al. [155] with their open-page policy.

Giannopoulou et al. [148, 149] use a different time-triggered approach. They partition access to the multiprocessor bus so that at any time, t, only memory accesses from tasks of the same criticality can occur. This may introduce some inefficiencies, but it reduces the temporal modelling of a mixed criticality shared bus to that of a single criticality shared bus. The latter problem is not, however, straightforward (but is beyond the scope of this review). In later work they generalise their approach by introducing the notion of isolation scheduling [198].

The problems involved in using a shared bus has lead Giannopoulou et al. to also include a Network-on-Chip (NoC) in their later work [151]. Burns et al. [45, 81, 85, 144] apply a 'one criticality at a time' approach to MCS scheduled by the use of a Cyclic Executive; they considered both partitioned and global allocation of jobs to frames.

Tobuschat et al. [353] have developed a NoC explicitly to support MCS. Their IDAMC protocol uses a *back suction* technique [121] to maximise the bandwidth given to low (or non) critical messages while ensuring that high-criticality messages arrive by their deadlines. The more familiar wormhole routing scheme [278] for a NoC has been expanded by Burns, Harbin and Indrusiak [86, 207] to provide support for mixed criticality traffic. Response-time analysis, already available for such protocols [326], is augmented to allow the size and frequency of traffic to be criticality aware. Wormhole routing is also used by Hollstein et al [191] to provide complete separation of mixed-criticality code; they also support run-time adaptability following any fault identified by a Built-In Self Test. On-chip networks require reliable/trusted interfaces to prevent babbling behaviour [74]; Ahmadian and Obermaisser [3] describe how to provide this via a time-triggered extension layer for a mixed-criticality NoC. Dynamic control of a mixed-criticality NoC is considered by Kostrzewa et al. [224]. Control over I/O contention via an Ethernet-based criticality-aware NoC is advocated by Abdallah et al. [1]. A focus on NoC secuity, in which HI-criticality messages need more protection than LO-criticality is taken by Papastefanakis et al. [288].

An alternative to having a NoC be used for all traffic (task to task and task to off chip memory) is proposed by Audsley [18, 153]. They advocate the use of

a separate memory hierarchy to link each core to off chip memory. A criticality aware protocol is used to pass requests and data through a number of efficient multiplexers. If the volume of requests and data is criticality dependent then analysis similar to that used for processor scheduling can be used on this memory traffic. The separation of execution-time from memory-access time is explored by Li and Wang [248]. They demonstrate that this distinction improves schedulability.

Controller Area Network (CAN) [152] is a widely used network for real-time applications, particularly in the automotive domain. It has been the subject of considerable attention with Response-Time Analysis derived [116] for what is, in essence, a fixed priority non-preemptive protocol. The use of CAN in mixed criticality applications has been addressed by Burns and Davis [82]. In this work it is the period of the traffic flows and the fault model that changes between criticality levels. A MixedCAN protocol was developed that makes use of a Trusted Network Component that polices the traffic that nodes are allowed to send over the network. Evaluations are used to show the advantages of using MixedCAN rather than a criticality agnostic approach. However the paper, in keeping with many other publications, only considered dual-criticality systems.

Herber at al. [185] also addressed the CAN protocol. They replaced the physical network controller with a set of virtual controllers that facilitate spacial separation. A weighted round robin scheduler in then used to give temporal isolation. Their motivation is to support virtualisation in an automotive platform. They do not however use criticality specific parameters for the different applications hosted on the same device.

Other protocols that have been considered in terms of their support for mixed criticality systems include FlexRay [157] and switched Ethernet [106, 107]. In the latter work, a change in criticality mode is broadcast to the entire system by adding a new field to the IEEE 1588 PTP (Precision Time Protocol).

A further communication protocol is addressed by Addisu et al. [2]. They consider JPEG2000 Video streaming over a wireless sensor network. With such a network the available bandwidth varies in an unpredictable way. They propose a bandwidth allocation scheme that is criticality aware. A wireless protocol (WirelessHART) is also used by Jin at al. [214, 215] to support delay analysis with fixed priority scheduling for sensor networks.

A novel scheduling approach (triangle scheduling) for mixed-criticality messages is proposed by Dürr et al. [125].

5 Links to other Research Topics

5.1 Hard and Soft Tasks

Although the label 'Mixed Criticality Systems' is relatively new, many older results and approaches can be reused and reinterpreted under this umbrella term. In particular dual-criticality systems in which there are hard and soft tasks combined has been studied since at least 1987 [239]. Hard tasks must be guaranteed. Soft tasks are then given the best possible service. But soft tasks are usually unbounded in some sense (either in terms of their execution time or their arrival frequency) and hence they must be constrained to execute only from within *servers* (execution-time servers). Servers have bounded impact on the hard tasks. Since 1987 a number of servers have been proposed. The major ones for fixed priority systems being the Periodic Server, the Deferrable Server, the Priority Exchange Server (all described by Lehoczky et al. [239]), and the Sporadic Server [334]. The ability to run soft tasks in the slack provided by the hard tasks is also supported by the Slack Stealing schemes [111, 118, 238, 304] which have similar properties to servers. These all have equivalent protocols for dynamic priority (EDF) systems; and some EDF specific ones exist such as the Constant Bandwith server [251].

Since their initial specification, analysis has improved and means of allocating and sharing capacity between servers have been investigated (see for example [63, 64, 369]). However, these results on servers (and examples of how they can be implemented in Operating Systems and programming languages) are generally known (see standard textbooks [88, 91, 257]). And hence they are not reviewed in more detail here. Note however, that standard servers only deal with the isolation/ partitioning aspect of MCS. To support sharing (of resources) there must be some means of moving capacity from the under utilised servers of high criticality tasks to the under provisioned servers of lower criticality tasks. The Extended Priority Exchange server [334] as well as work on making use of gain time, show how this can be achieved.

Another way of maximising the slack available for soft tasks is the dual-priority scheme [89, 119]. Here there are three bands of priority. The soft tasks run in the middle band while the hard tasks start in the lower band but are promoted to the higher band at the latest possible time commensurate with meeting their deadlines. So hard task execute when they have to, or when there are no soft tasks, soft tasks run otherwise.

Run-time adaptability for MCS has been addressed by Hu et al. [193, 194]. They present an approach to adaptively shape at runtime the inflow workload of LO-criticality tasks based on the actual demand of HI-criticality. This improves the QoS of LO-criticality tasks; but it not clear what level of guarantee is pro-

vided for these tasks. An alternative scheme, with the same aim, is given by Hikmet [189].

5.2 Fault Tolerant Systems

Fault tolerant systems (FTS) typically have means of identifying a fault and then recovering before there is a system failure. Various recovery techniques have been proposed including exception handling, recovery blocks, check-points, task re-execution and task replication. If, following a fault, extra work has to be undertaken then it follows that some existing work will need to be abandoned, or at least postponed. And this work must be less important than the tasks that are being re-executed. It follows that many fault tolerant systems are, in effect, mixed criticality.

To identify a fault, timeouts are often used. A job not completing before a deadline is evidence of some internal problem. Earlier warning can come from noting that a job is executing for more than its assumed worst-case execution time. Execution-time monitoring is therefore common in safety critical systems that are required to have at least some level of fault tolerance. Again this points to common techniques being required in FTS and MCS.

In was noted earlier, in the discussion on CAN (Section 4.3), that a fault model can be criticality dependent [82] – a task may, for example, be required to survive one fault if it is mission critical, but two faults if it is safety critical. The difference between assumed computation times at different criticality levels, may be a result of the inclusion or not of recovery techniques in the assumed worst-case execution time of tasks.

Although there is this clear link between FTS and MCS there has not yet been much work published that directly addresses fault-tolerant mixed criticality system. Exceptions being work by Huang et al. [204,205], a paper by Pathan [291] that both focus on service adaptation and the scheduling of fault-tolerant MCS, and a four-mode model developed by Al-Bayati et al. [4]. Work by Thekkilakattil uses Zonal Hazard Analysis and Fault Hazard Analysis [348] and Error-Burst models [347] to deliver both flexibility and real-time guarantees (for the most critical tasks). Thekkilakattil et al. [349] also considers the link between MCS and the tolerance of permanent faults. Lin et al. [249] attempt to integrate mixed criticality with the use of primary and backup executions (in both of the two criticality modes they consider). Islam at al. [208], in a paper that preceded that of Vestal, looked at combining different levels of replication for different levels of criticality.

As highlighted already in this review, many models and protocols for mixed criticality behaviour allow the system to move through a sequence of mode (criticality modes). With a two mode system (HI and LO) the system starts in the LO

mode in which all deadlines of all tasks are guaranteed, but can then transition to the HI mode in which only the HI-criticality tasks are guaranteed (and the LO-criticality tasks may actually be abandoned). It may, or may not, later return to the LO mode when it is safe to do so. Burns [77] attempts to compare these criticality mode changes with the more familiar system mode change. He concludes that the LO mode behaviour should be considered to be the 'normal' expected behaviour. A move away from this mode is best classified as a fault; with all other modes been considered forms of *graceful degradation*. A move back to the fully functional LO mode is closest in nature to an *operational* (sometime known as exceptional) mode change. Such a mode change is *planned but may never occur*.

5.3 Hierarchical Scheduling

One means of implementing a MCS where strong partitioning is needed between applications is to use a hierarchical (typically two-level) scheduler. A trusted base scheduler assigns budgets to each application. Within each application a secondary scheduler manages the threads of the application. There are a number of relevant results for such resource containment schemes (e.g. [95, 113, 114, 252, 310, 371]). Both single processor and multiprocessor platforms can support hierarchical scheduling.

Unfortunately when hierarchical scheduling is applied to MCS there is a loss of performance [230]. A simple interface providing a single budget and replenishment period (which is often associated with virtualisation or the use of a hypervisor [10]) is too inflexible to cater for a system that needs to switch between criticality levels. To provide a more efficient scheme, Lackorznshi et al. [230] propose 'flattening' the hierarchy by exposing some of the interval structure of the scheduled applications. They develop the notion of a scheduling context which they apply to MCS [358]. In effect they assign more than one budget to each 'guest' OS. As a result, applications that would otherwise not be schedulable are shown to utilise criticality to meet all deadlines. An alternative, but still flexible approach, is provided by Groesbrink et al. [163, 164]. They allow budgets to move between virtual machines executing on a hypervisor that is itself executing on a multi-core platform. The hypervisor controls access to the processor, the memory and shared I/O devices. Yet another scheme is described by Marinescu et al [261]; they are more concerned with partitioning as opposed to resource usage, but they do address distributed heterogeneous architectures. Hypervisors are also used by Cilku and Puschner [103], to give temporal and spacial separation on a multiprocessor platform, and Perez et al. [297] use a hierarchical scheduler to statically partition a wind power mixed criticality embedded system requiring certification under the IEC-61508 standard. A hypervisor for a mixed criticality on-board satellite soft-

ware systems is discussed by Salazar et al. [9, 313] and one for general control systems is addressed by Crespo et al [105]. The issue of minimising the overheads of a hypervisor is addressed by Blin et al. [70].

5.4 Cyber Physical Systems and Internet of Things

In parallel with the development of a distinct branch of research covering mixed criticality systems has been the identification of Cyber Physical Systems (CPSs) as a useful focus for system development. Not surprisingly it has been noted that many CPSs are also mixed criticality. For example Schneider et al. [319] note that many CPSs are a combination of deadline-critical and QoS-critical tasks. They propose a layered scheme in which QoS is maximised while hard deadline tasks are guaranteed. Izosimov and Levholt [210] use a safety-critical CPS to explore how metrics can be used to map potential hazards and risk from top level design down to mixed criticality components on a multi-core architecture. Issues of composability within an open CPSs is introduced in the short paper by Lee et al. [235].

Maurer and Kirner [264] consider the specification of cross-criticality interfaces (CCI) in CPSs that define the level of communication allowed between 'open' subsystems/components. Lee at al. [234] also look at interfaces and composition for mixed criticality CPSs.

The link between the Internet of Things, IoT, and MCS is made by Kamienski et al. [219] in the context of development methods for energy management in public buildings. Smart buildings are also the focus of the work of Dimopoulos et al. [122] on a context-aware management architecture.

5.5 Probabilistic real-time systems

In mixed criticality systems, the worst-case execution time of a task is expressed as a function of the criticality level (e.g. $C(LO)$ and $C(HI)$) with larger values for the WCET obtained for higher criticality levels. Research into probabilistic hard real-time systems can be viewed as extending this model to a continuum (or at least a large number of discrete values). Instead of a number of single values for the WCET with different levels of confidence, the worst-case execution time is expressed as a probability distribution, referred to as a pWCET [65].

The exceedance function (or 1 - CDF [19]) for the pWCET gives the probability that the task will exceed the specified execution time budget on any given run. Conversely, the exceedance function may be used to determine the execution time budget required such that the probability of overrunning that budget does not exceed a specified probability. This is illustrated in Figure 2. Here, an execution time

[19] Cumulative Distribution Function.

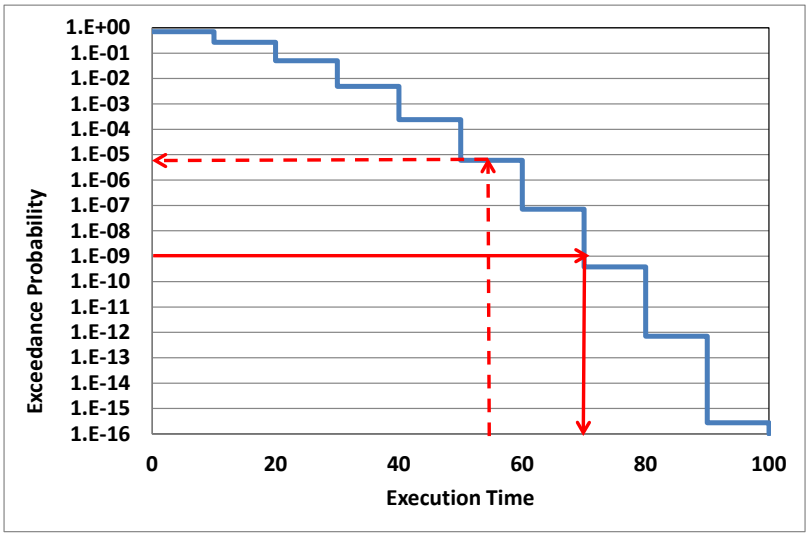

Figure 2: pWCET distribution as an Exceedance function

budget of 55 has a probability of being exceeded on any single run of 10^{-5}, whereas the execution time budget required to ensure that the probability of exceedance on any single run is at most 10^{-9} is 70. We note that exceedance probabilities and failure rates (e.g. 10^{-9} failure per hour) are not the same, but that such probabilities can be transformed into failure rates by accounting for the number of jobs in a given time period, or via probabilistic schedulability analysis techniques.

Probabilistic analysis provides an alternative treatment for mixed criticality systems, where high criticality tasks are specified as having an extremely low acceptable failure rate (e.g. 10^{-9} per hour), whereas a higher failure rate (e.g. 10^{-6} or 10^{-7} per hour) is permitted for lower criticality tasks. Probabilistic worst-case execution times [12, 93] and the probabilistic worst-case response times [120, 258] derived from them provide a match to requirements specified in this way. These techniques can potentially be used to show that pathological cases with very high execution times / high response times have a provably vanishingly low probability of occurring, thus avoiding the need to over-provision compute resources to handle these cases.

Just as MCS has expanded from a focus on worst-case execution times to one that includes arrival rates (for sporadic work), probabilistic analysis has been developed [7] for the case where the arrival rate of tasks is described by a probability distribution. This work could form a further link between MCS and probabilis-

tic analysis. Indeed Masrur [262] uses random jitter on the arrival time of LO-criticality tasks to improve schedulability.

Guo et al. [177] demonstrate the usefulness of a probabilistic framework in their analysis of an EDF scheduled system in which there is a permitted (but low) probability of timing faults. The chances of a HI-criticality task executing for more than its LO-criticality value is also expressed as a probability. Their current work assumes that task execution times are independent; this is an unrealistic assumption, but one that could be weakened in future work. Santinelli and George [314] also explore the probability space of worst-case execution times for MCS. Probabilistic analysis for the SMC and AMC schemes is derived by Maxim et al. [265]. And a Markov decision process is used by Alahmad and Gopalakrishnan [6] to model job releases in MCS. Probabilistic analysis is also used to investigate the safety of each criticality level [124].

6 More Realistic MCS Models

The abstract behavioural model described in Section 2 has been very useful in allowing key properties of mixed criticality systems to be derived, but it is open to criticism from systems engineers that it does not match their expectations. In particular:

- In the HI-criticality mode, LO-criticality tasks should not be abandoned. Some level of service should be maintained if at all possible, as LO-criticality tasks are still critical.
- For systems which operate for long periods of time it should be possible for the system to return to the LO-criticality mode when the conditions are appropriate. In this mode all functionality should be provided.

It can be argued that these criticisms are, at least partly, misplaced as any high integrity system should remain in the LO-criticality mode for its entire execution: the transition to HI-criticality mode is only a theoretical possibility that the scheduling analysis can exploit [46]. Nevertheless, in less critical applications (such as those envisaged in the automotive industry) actual criticality mode changes may be experienced during operation and the above criticisms should be addressed. Of course for some applications it is acceptable to provide only limited timing guarantees during these rare events, and hence no online controls are required [360].

However, where online action is required, to prevent the abandonment of all LO-criticality tasks following a criticality mode change a number of reconfigurations are possible:

1. Let any LO-criticality job that has started, run to completion (this is in effect what is assumed in many forms of analysis [46]).

2. Reduce priorities of the LO-criticality tasks [43], or similar with EDF scheduling [199, 200].

3. Increase the periods and deadlines of LO-criticality jobs [213,336,338–340], called *task stretching*, the *elastic task model* or *multi-rate*.

4. Impose only a weakly-hard constraint on the LO-criticality jobs [146].

5. Decrease the computation times of some or all of the LO-criticality tasks [80][20] [255].

6. Move some LO-criticality tasks to a different processor that has not experienced a criticality mode change [365].

7. Abandon LO-criticality work in a disciplined sequence [143, 171, 203, 307].

The fifth action leads to a modification to the system model; whereas for HI-criticality tasks we have $C(HI) \geq C(LO)$, for LO-criticality tasks we now have $C(HI) \leq C(LO)$. For some tasks $C(HI) = 0$, that is they are abandoned. For others a lower level of service can be guaranteed. For some they may be able to continue with their full computation time budgets.

The final approach is addressed by Fleming and Burns [143]; they introduce a further notion into the standard model; tasks are allocated to *applications* and each application is assigned (by the system designers) an *importance* level. LO-criticality tasks are abandoned in inverse order of importance. Huang et al. [203] also introduce an extension to the standard model by the use of an ICG (Interference Constraint Graph) to capture more specifically which tasks need to be dropped when particular higher criticality tasks exceed their allocated criticality-aware execution times. Controlled abandonment by the use of partitioning is advocated by Mahdiani and Masrur [260] in the context of the EDF-VD scheduling.

Obviously all seven schemes can be used together: complete or move all started jobs, allow some new jobs to have an extended deadline or reduced computation time or a weakly-hard constraint, reduce the priorities of some others jobs and abandon those of lowest importance in particular partitions. A particular approach is advocated by Su et al. [337]; here LO-criticality tasks have two periods (short and long) and two priorities. At the criticality mode change these tasks switch to their longer periods and new priorities. Analysis is provided to show that all modes are schedulable.

[20] Note equation (6) in this paper has a typo, both $R_i(LO)$ terms should be starred ($R_i^*(LO)$).

A flexible scheme utilising hierarchical scheduling is proposed by Gu et al. [127, 170]. They differentiate between minor violations of LO-criticality execution time which can be dealt with within a component (an internal mode change) and more extensive violations that requires a system-wide external mode change. In doing so they introduce a new mixed-criticality resource interface model for component-based system which supports isolation, virtualisation and conditionality.

In keeping with other mode change situations ([87,135,293,305,324,351,352]) a simple protocol for controlling the time of the change of mode back to LO-criticality is to wait until the system is idle (has no application tasks to run) and then the change can safely be made [351]. Santy et al. [318] extend this approach and produce a somewhat more efficient scheme that can be applied to globally scheduling multiprocessor systems (in which the system may never get to an idle tick). With a dual criticality system that has just transitioned into the HI-criticality mode (and hence no low-criticality jobs are executing); their protocol first waits until the highest priority HI-criticality job completes, then its waits until the next highest priority job is similarly inactive. This continues until the lowest priority job is inactive; it is then safe to reintroduce all LO-criticality tasks. Obviously if there is a further violation of the $C(LO)$ bound then the protocol is abandoned and subsequently restarted. The authors call this a SCR (Safe Criticality Reduction); their paper also has a second protocol, but this is less intuitive and considerably more expensive at run-time.

A more aggressive scheme for returning a system back to its LO-criticality mode is proposed by Bate et al. [61, 62]. In this approach a bailout protocol is proposed. HI-criticality tasks take out a loan if they execute for more than their $C(LO)$ estimate. Other tasks repay the loan by either not executing at all or by executing for less than expected. When the loan is repaid (and a further condition is met) the system returns to it normal mode. They demonstrate, using a scenario-based assessment, that the bailout protocol returned the system to the normal mode much quicker than the 'wait for idle tick' scheme.

As well as experiencing a criticality mode change a system can, of course, be structured to behave in a number of operational or behavioural modes. As indicated earlier, Burns [77] compares and contrasts these two forms of mode change. De Niz and Phan [280] note that the criticality of a task can depend on the behavioural mode of the system. They develop scheduling analysis for this dependency and consider the static allocation of such tasks to multiprocessor platforms.

Another aspect of the 'standard model' for MCS that can be argued to be unrealistic is the idea that a system with, say, five criticality levels would also have five different estimates of worst-case execution time for its most critical tasks. An augmented model has been proposed [78,279] that restricts each task to having just two estimates of WCET. So, in the general case where there are V criticality levels,

L_1 to L_V (with L_1 being the highest criticality), each task just has two C values. One represents its estimated execution time at its own criticality level ($C_i(L_i)$) and the other an estimate at the base (i.e., lowest) criticality level ($C_i(L_V)$). It follows that if a job is of the lowest criticality level (i.e., $L_i = L_V$) then it only has one WCET parameter. For all other jobs, $C(L_i) \geq C(L_V)$. The two parameters of this augmented model have been referred to as $C(self)$ (or $C(SF)$) and $C(normal)$ ($C(NL)$); the model seems to be sufficiently expressive to capture most of the key properties of mixed criticality systems. However, Baruah and Guo [56] has shown that: *"The Burns model is strictly less expressive than the Vestal model. Determining whether a given instance can be scheduled correctly remains NP-hard in the strong sense. Lower bounds on schedulability, as quantified using the speedup factor metric, are no better for the Burns model than for the Vestal model."* This quote introduces terms described in the next section.

7 More Formal Treatments

In this section we consider utilisation bounds, speedup factors and (formal) language issues.

7.1 Utilisation Bounds

For normal single criticality systems there are well known bounds on task set utilisation that will deliver a schedulable system with either fixed priority or EDF scheduling. Although the definition of utilisation is not straightforward when a task has more than one worst-case computation time, it is possible to give an effective definition and to derive least upper bounds (LUBs) for MCS. Santos-Jr et al. [316] derive a number of useful results for LUB. They construct a task set that is unschedulable (during a criticality mode change) with LUB arbitrarily close to 0. But where tasks have harmonic periods LUB can reach 1 (for a uniprocessor system). Between these two extremes they show that if higher criticality tasks do not have periods longer than lower criticality tasks then LUB lies between $ln2$ and $2(\sqrt{2}-1)$.

7.2 Speedup Factors

It has been shown [29,31,37,39,180] that the mixed criticality schedulability problem (preemptive or non-preemptive) is strongly NP-hard even if there are only two criticality levels. Hence only sufficient rather than exact analysis is possible. A list of open problems with regard to the schedulability of MCS is provided by Ekberg and Yi [133].

For approaches and tests that are only sufficient, an assessment of their quality is possible if a *speedup factor* can be computed. A speedup factor [218] of $X (X > 1)$ for schedulability test S implies that a task set that is schedulable on a processor of speed 1 will be deemed schedulable by S if the processor's speed is increased to X. Of course, in general, it is not possible to know if the task set is schedulable on the original speed 1 processor (this would require an exact and possibly even a clairvoyant test), but a real scheduling scheme and test with a speedup factor of say 2 is clearly better than one with a speed up factor of 10.

For job-based fixed priority scheduling, a priority assignment scheme and test has been found [38, 39, 57, 244] with a speed up factor of S_L (for L criticality levels), where S_L is the root of the equation $x^L = (1 + x)^{L-1}$. For $L = 2$ the result is $S_2 = (1 + \sqrt{5})/2$ which is equal to the golden ratio, $\phi = 1.618$. This can be compared with a partitioned approach (all HI-criticality jobs have priorities higher than all LO-criticality jobs) which has an unbounded speedup factor. This latter result is easily illustrated by considering a two job system. The LO-criticality job has a small computation time, 1, and deadline of 2. The HI-criticality job has a huge computation time of G and a deadline of $G + 1$. These two jobs will both meet their deadlines if the LO-criticality task is given the highest priority. But the reverse priority assignment (which executes the HI-criticality job first) will only be schedulable if $G + 1$ will fit into the deadline 2. To obtain this a speedup of $(G + 1)/2$ is required. As G can be arbitrary large the speedup factor is effectively unbounded.

For EDF scheduled systems Baruah et al. [42, 57] prove that a variant of EDF (EDF-VD, described in Section 3.3) in which HI-criticality sporadic tasks (in a dual-criticality system) have their deadlines reduced (in the LO-criticality mode) is also schedulable on a single processor that is speeded up by a factor ϕ. They also show that a finite set of independent jobs? scheduled on m identical multiprocessors is schedulable with a speed-up factor of $\phi + 1 + 1/m$. And on a partitioned system a speed-up factor of $\phi + \epsilon$ is derivable for any value of $\epsilon > 0$. In later work [41] they improve this bound to 4/3 (1.333) rather than ϕ (1.618). Further formal analysis of EDF-VD is provided by Li [242], Muller and Masrur [271] and Gu and Easwaran [168]. The MC-Fluid approach also has a speed-up factor of 4/3 [50], as does EDF-VD when applied to systems with degraded/imprecise guarantees [255].

7.3 Formal Language and Modelling Issues

The application of formal design languages, such as real-time BIT (Behavior Interactions Priorities), are being used to model MCS. And verification approaches such as model checking or simulation are being applied to both application software and

multi-core platforms. See, for example, the work of Socci et al. [330]. State-space explosion is of course always an issue with these approaches.

A 2005 paper by Amey at al. [13], which predates Vestal's work, looked at the (smart) certification of mixed criticality systems. They report real industrial application of formal code analysis to prove isolation between tasks of difference criticality levels. In one application, concerning safety-critical landing guidance for ship-borne helicopters, SIL 4 code (the highest in UK Defence Standard 00-55 [267]) was executing in the same processor and memory space as SIL3 and SIL2 code. Another application (a civil jet engine monitoring unit) have Level-C and Level-E code co-located (DO-178B standard [309] has levels A down to E). In both of these examples formal analysis of information flow at the program level was able to demonstrate code segregation.

Compile time checking is also advocated by Lindgren at al. [250] with their experimental RTFM-language. Language constructs allow static assessment of the interfaces between critical and non-critical code. At run-time however separation is achieved by assigning higher priorities to the critical tasks. As indicated earlier this is not a very effective strategy in terms of efficient resource usage.

Model-based design using Synchronous Reactive models is used to design embedded control systems and is formalised within languages such as LUSTRE, SIGNAL and Simulink. Its application to MCS is considered in detail by Zhao et al [375]; where they adopt the elastic mixed-criticality task model for fixed-priority scheduling.

An alternative task model to that which underpins most scheduling research on MCS is the sporadic DAG (Directed Acyclic Graph) task model. An initial study, within the context of multiprocessor federated systems is provided by Baruah [35].

8 Systems Issues

A fundamental issue with MCS is separation. Many of the more theoretical papers reviewed here assume various levels of run-time monitoring and control. However, few papers consider or demonstrate how the required mechanisms can be implemented, Neukirchner et al. [276] presented one such paper. They consider memory protection, timing fault containment, admission control and (re-)configuration middleware for MCS. Their framework [140] is aimed at supporting AUTOSAR conforming applications within the automotive domain. An early paper looking at non-interference at the memory level for IMA platforms within the avionics industry is that of Hill and Lake [190].

Another detailed study of the overheads for two common implementations schemes for MCS is presented by Sigrist et al. [327]. They conclude that overheads

of up to 97% can be encountered and they recommend that all scheduling models be extended to include parameters to capture the impact of run-time overheads.

The issue of monitoring is also addressed by Motruk et al. [270] in the context of their IDAMC (Integrated Dependable Architecture for Many Cores). This work builds on the the more general (i.e. not MCS specific) work on separation, isolation and monitoring for SoC/NoC architectures. The MultiPARTES project (see Section 10) is addressing virtualisation in terms of Model Driven Engineering for MCS [10, 101,102,354,355]. Goossens et al. [154] are also looking at virtualisation "to allow independent design, verification and execution" with their CompSOC architecture. Paravirtualisation of legacy RTOSs to provide the necessary memory isolation is considered by Armbrust et al. [15].

Hypervisor technology is also being used to give the appropriate level of isolation in MCS. The DREAMS architecture uses it [233] to minimise interference via modelling patterns of execution. Evripidou and Burns [139] employ different execution-time servers (deferrable server for short deadline event-triggered work, and periodic server for periodic work) under the control of a hypervisor to bound the overheads associated with server technology. If there is a criticality induced mode change then the deferrable servers are transposed to the much more efficient (but less responsive) periodic servers. A general hypervisor architecture for multi-core MCS is presented by Pérez et al. [296].

The development of purpose built hardware (FPGA based) to support reliable MCS is being undertaken as part of the RECOMP project (see Section 10). They aim to reduce the cost of certification for MCS on multiprocessor architectures by use of open source hardware and software [266, 277, 299]. Santos et al. [315] are also looking at systems built on FPGA platforms. They have developed a criticality-aware scrubbing mechanism that improves system reliability by up to 79%. Scrubbing is a technique for recovering from SEU (single event upsets) that effect FPGA platforms in harsh environments such as space.

Many MCS papers have, either explicitly or implicitly, focused on issues of safety and reliability. Criticality can however also refer to security. Within this domain it is usual to have different security levels. And hence much of the extensive literature on security is relevant, but is out of scope of this review. Some work is nevertheless applicable to safety and security; for example the definition of a *separation kernel* for a system-on-chip built using a time-triggered architecture [363].

Such a separation kernel has been developed by West et al. [247, 268, 364]. They can host guest operating systems, such as Linux but have their own real-time operating system (RTOS), QUEST-V. They partition the available cores into Sandboxes that have different criticality levels. Their architecture is aimed at achieving efficient resource partitioning and performance isolation. One means of achieving this is for interrupts to go directly to the appropriate partition, they do not have to

be first handled by the hypervisor.

PikeOS [217] also employs a separation microkernel to provide 'a powerful and efficient paravirtualization real-time operating system' [311] for a partitioned multi-core platform. Another high-assurance (micro) kernel is sel_4. Lyons and Heiser [259] show how the sel_4 model can be extended to cater for mixed criticality. VMs (virtual machines) that are appropriate for real-time Java-based mixed-criticality systems have been designed by Ziarek and Blanton [380] and Hamza et al. [178].

To implement the criticality mode change the run-time support system must support execution time monitoring, the modes and mode changes. Baruah and Burns [43] show how this can be achieved within the facilities provided by the Ada programming language. Kim and Jin do the same for a standard RTOS [223]. They make use of bitmaps to provide a very efficient implementation. DMPL [94] is a language designed specifically for distributed real-time MSC.

A further operating system designed to support mixed criticality is Kron-OS [110]. This controls the execution of RSFs (Repetitive Sequence of Frames) that is partitioned between two criticality levels.

As an alternative to using an RTOS to give the right level of protection and (safe) resources sharing, Zimmer et al. [381] have designed a processor (FlexPRET) to directly support MCS. They use fine-grained multithreading and scratch-pad memory to give protection to hard real-time tasks whilst increasing the resource utilisation of soft tasks. In effect soft tasks (threads) can safely exploit the spare capacity generated from the hard tasks at the cycle level. They have a soft-core FPGA implementation that caters for up to 8 hardware threads (each of which can support a number of software threads). A more focused scheme aimed at partitioning the cache is described by Lesage et al. [240]. An LC (least critical) cache replacement policy is evaluated by Kumar et al. [229]. The effective use of cache, for a multicore platform, is also considered by Chrisholm et al. [100].

A hardware platform that supports applications of different criticality must manage its I/O functions is a partitioned and hence safe (and secure) way. If lower criticality work can cause interrupt to occur 'at any time' then unpredictable overheads may be suffered by high criticality applications. This is a topic addressed by Paulitsch et al. [292]. They rightly claim that this a topic "often overlooked".

Although research on MCS has generated many different approaches, there have been few empirical benchmarks or comparative studies. One useful study however was published in 2012 by Huang et al. [196]. They compared Vestal's scheme with its optimal priority assignment, their improved slack scheduling scheme and Period Transformation (PT) (see Section 5.1). They conclude that Vestal's approach and period transformation usually (though not always) outperform slack scheduling; and that there are additional overheads with period transformation and

slack scheduling. Nevertheless the overheads were not excessive (typically an extra 0.3%). Later Fleming and Burns [142] compared Vestal's approach, AMC (see Section 3.2) and PT for multiple criticality levels. As the number of criticality levels increased the relative advantage of PT, even when overheads are ignored, was observed to decrease. This observation was also supported, in 2014, by Huang et al. [197] who updated their study and concluded that AMC-based scheduling gave the best performance for fixed priority sporadic task systems. This study also looked at the overheads involved in user-space implementation of AMC on top of Linux, but without kernel modifications.

The need for useful benchmarks is noted in a number of papers. One industrially inspired case study is provided by Harbin et al. [181]. The use of realistic simulations to evaluate schemes is discussed by Bate et al. [61, 62], Griffin et al [162] and Ittersshagen et al. [209]. A brief comparison of approaches to multiprocessor scheduling of MCS is provided by Osmolovskiy et al. [287]. The evaluation of communications within a MCS is considered by the work of Napier et al. [272] and Petrakis et al. [298].

Another systems issue of crucial important in many mobile embedded systems is power consumption. The work of Broekaert et al. [73] allocates and monitors power budgets to different criticality levels. If a crucial VM (Virtual Machine) "*overpassed its power budget during its time partition, the extra power consumed will be removed from the initial power budget of the next low critical VM scheduled*". Energy consumption is also addressed by Legout et al. [237]. They trade energy usage with deadline misses (of low-criticality tasks), and claim a 17% reduction in energy with deadline misses kept below 4%. The objective of minimising energy usage is used by Zhang et al. [374] to drive task allocation (in a multiprocessor system). As discussed earlier, a slightly different approach is taken by Huang et al. [201]. They advocate the use of DVFS (Dynamic Voltage and Frequency Scaling) to increase the speed of the processor if HI-criticality tasks need more than their $C(LO)$ requirement. Hence LO-criticality work is not abandoned, but more energy is used. They integrate their approach with the EDF-VD scheduling scheme (see Section 3.3) and have, more recently, addressed multi-core platforms [273]. This approach is extended by Ali et al. [8] who propose a new dynamic power-aware scheduling scheme for hardware with discrete frequency levels. Awan et al. [22] consider how energy-aware task allocation can be utilised in the context of heterogeneous multicore systems.

Where energy is limited or indeed the system is energy neutral, then criticality-aware energy usage becomes crucially important [357]. ENOS [361] is an experimental OS that addresses mixed resources (time and energy) and mixed criticality. It transforms the system through a series of 'energy modes' including one that ensures all state is safely stored in persistent memory before system blackout. En-

ergy harvesting in the context of a battery-less real-time system is considered by Asyaban et al. [16]. They propose a scheduling scheme that satisfies both temporal and success-ratio constraints whilst addressing uncertainty in the platform's power management. Even where energy is not limited, isolation in terms of power usage and temperature control is important; an issue addressed by Grüttner in the context of heterogeneous MPSoCs [165].

Complex mixed criticality systems also present a number of significant challenges at the specification and design stage. Herrera et al. [187, 188] propose a modelling and design framework for MCS hosted on Systems-on-Chip and/or Systems-of-Systems. They present a core ontology but freely admit that there is considerable work to do before a sound engineering process is available for system builders/architects. Giannopoulou et al. [147] support the development of MCS on multi-core platforms by the development of an appropriate tool chain. This group has also considered [150, 370] the mapping and design of fault-tolerant MCS to multicore platforms.

9 Conclusion

As identified in the introduction, the fundamental issue with MCS is how to reconcile the differing needs of separation (for safety) and sharing (for efficient resource usage). These concerns have lead to somewhat of a bifurcation in the resulting research. Much of the implementation and systems work, in for example the funded projects (see the appendix), has concentrated on how to safely partition a system so that high integrity components can, in some way, share computational and communication resources. By comparison, the more theoretical and scheduling research has largely focused on how criticality-specific worst-case execution times can be utilised to deliver systems that are schedulable at each criticality level but have high processor utilisation.

Unfortunately these two areas of research are not easily integrated. Flexible scheduling requires, at least, dynamic partitioning. Certified systems require complete separation or at least static partitioning. Future work must address this mismatch.

A second topic for future work is a move away from a processor-centric view of MCS to one that incorporates other shared resources. For example communication – particularly on a multi-core or many-core platform. Can a shared bus provide the required separation, or is a Network-on-Chip protocol required? Work is only beginning to address these issues.

What becomes clear from reading the extensive literature that has been produced since Vestal's paper, is that MCS presents a collection of interesting issues

that are both theoretically intriguing and implementationally challenging. It is to be expected that MCS will continue to be a focus for practical and theoretical work for some time to come.

Acknowledgements

The authors would like to thank Sanjoy Baruah for a number of very useful discussions on the topic of this paper. The research that went into writing this paper is funded in part by the ESPRC grants, MCC (EP/K011626/1) and MCCps.

10 Appendix - Projects

This appendix outlines some of the publicly funded projects that have to some degree focused on MCS.

1. **MultiPARTES** is a research project aimed at developing tools and solutions for building trusted embedded systems with mixed criticality components on multicore platforms[21]. It is a collaborative EU FP7 ICT research project with 9 partners. A major outcome of the project is the*XtratuM*, an open source hypervisor designed as a generic virtualization layer for heterogeneous multicores. Project completed in 2013.

2. The **RECOMP** (Reduced certification cost for trusted multi-core platforms) research project aims to establish methods, tools and platforms for enabling cost-efficient certification and re-certification of safety-critical systems and mixed-criticality systems, i.e. systems containing safety-critical and non-safety-critical components[22]. It was a European funded project from the ARTEMIS Joint Undertaking. Project completed in 2013.

3. **ACROSS** is a research project that aims to develop and implement an ARTEMIS cross-domain reference architecture exploiting, for example, PikeOS, partitioning and time-triggered bus access protocols, for embedded systems based on the architecture blueprint developed in the EU FP7 project GENESYS[23]. Project completed in 2013.

4. **CERTAINTY** (Certification of Real Time Applications designed for mixed criticality) is an EU FP7 research project[24]. It aims to be a key enabler of the certification process for mixed-critical embedded systems featuring functions dependent on information of varying confidence levels. Such a certification process is particularly needed in the avionic, automotive and automation domains, where concurrent functions with real-time and safety-critical requirements are a reality. Project completed in 2014.

5. **IMPReSS** is a joint EU-Brazil project [25]. The aim of the project is to provide a Systems Development Platform which enables rapid and cost effective development of mixed criticality complex systems involving Internet of Things

[21] http://www.multipartes.eu/
[22] http://atc.ugr.es/recomp/
[23] http://www.across-project.eu/
[24] http://www.certainty-project.eu/
[25] http://impressproject.eu/news.php

and Services (IoTS) and at the same time facilitates the interplay with users and external systems. Project completed in 2016.

6. **MCC** (Mixed Criticality Embedded Systems on Many-Core Platforms) is a UK funded (EPSRC) project looking at the design, verification and implementation of Mixed Criticality Systems[26]. It has a specific focus on developing NoC criticality-aware protocols. Project completed in 2016. A follow on 3 year project, MCCps started in 2016 (also funded by EPSRC).

7. **PROXIMA** is an EU FP7 IP Project[27]. The PROXIMA thesis is that the temporal behaviour of mixed-criticality complex real-time embedded systems executing on multi-core and many-core platforms can be analysed effectively via innovative probabilistic techniques. PROXIMA defines new hardware and software architectural paradigms based on the concept of randomisation. Project due to complete in 2016.

8. **CONTREX** (Design of embedded mixed-criticality CONTRol systems under consideration of EXtra-functional properties) focusses on platforms and addresses properties such as real-time, power, temperature and reliability[28]. It aims to develop meta-models for design and analysis. Project due to complete in 2016.

9. **DREAMS** (Distributed REal-time Architecture for Mixed Criticality Systems) aims to produce a European reference architecture for mixed-criticality systems[29]. Project due to complete in Sept. 2017.

10. **EMC**^{2} (Embedded multi-core systems for mixed criticality applications in dynamic and changeable real-time environments) is an ARTEMIS project with 100 (approx) partners is looking at open and adaptive systems. Project due to complete in 2017.

11. **SAFURE** (Safety And Security By Design For Interconnected Mixed-Critical Cyber-Physical Systems) is an EU Horizon 2020 research project[30]. SAFURE targets the design of cyber-physical systems by implementing a methodology that ensures safety and security "by construction". This methodology is enabled by a framework developed to extend system capabilities so as to control the concurrent effects of security threats on the system behaviour. Project due to complete in 2018.

[26] http://www.cs.york.ac.uk/research/research-groups/rts/mcc/
[27] http://www.proxima-project.eu/
[28] https://contrex.offis.de/home/
[29] http://www.dreams-project.eu/
[30] https://safure.eu/

References

[1] L. Abdallah, M. Jan, J. Ermont, and C. Fraboul. I/O contention aware mapping of multi-criticalities real-time applications over many-core architectures. In *Proc. WiP, RTAS*, pages 25–28, 2016.

[2] A. Addisu, L. George, V. Sciandra, and M. Agueh. Mixed criticality scheduling applied to jpeg2000 video streaming over wireless multimedia sensor networks. In *Proc. WMC, RTSS*, pages 55–60, 2013.

[3] H. Ahmadian and R. Obermaisser. Time-triggered extension layer for on-chip network interfaces in mixed-criticality systems. In *Proc. Digital System Design (DSD)*, pages 693–699. IEEE, 2015.

[4] Z. Al-Bayati, J. Caplan, B.H. Meyer, and H. Zeng. A four-mode model for efficient fault-tolerant mixed-criticality systems. In *Proc. DATE*, pages 97–102. IEEE, 2016.

[5] Z. Al-Bayati, Q. Zhao, A. Youssef, H. Zeng, and Z. Gu. Enhanced partitioned scheduling of mixed-criticality systems on multicore platforms. In *20th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 630–635, 2015.

[6] B. Alahmad and S. Gopalakrishnan. A Risk-Constrained Markov Decision Process Approach to Scheduling Mixed-Criticality Job Sets. In *Proc 4th WMC (RTSS)*, 2016.

[7] B. Alahmad, S. Gopalakrishnan, L. Santinelli, and L. Cucu-Grosjean. Probabilities for mixed-criticality problems: Bridging the uncertainty gap. In *WiP, RTSS*, pages 1–4, 2011.

[8] I. Ali, J. Seo, and K.H. Kim. A dynamic power-aware scheduling of mixed-criticality real-time systems. In *Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM)*, pages 438–445, 2015.

[9] A. Alonso, J.A. de la Puente, J. Zamorano, M.A. de Miguel, E. Salazar, and J. Garrido. Safety concept for a mixed criticality on-board software system. *IFAC-PapersOnLine*, 48(10):240–245, 2015.

[10] A. Alonso, C. Jouvray, S. Trujillo, M.A. de Miguel, C. Grepet, and J. Simo. Towards model-driven engineering for mixed-criticality systems: MultiPARTES approach. In *Proceedings of the Conference on Design, Automation and Test in Europe, WICERT*, DATE, 2013.

[11] A. Alonso, E. Salazar, and M.A. de Miguel. A toolset for the development of mixed-criticality partitioned systems. In *HiPEAC Workshop*, 2014.

[12] S. Altmeyer, L. Cucu-Grosjean, and R.I. Davis. Static probabilistic timing analysis for real-time systems using random replacement caches. *Real-Time Systems*, 51(1):77–123, 2015.

[13] P. Amey, R. Chapman, and N. White. Smart certification of mixed criticality systems. In *Reliable Software Technologies, Proc. of the Ada Europe Conference*, pages 144–155. Springer Verlag, LNCS 3555, 2005.

[14] J.H. Anderson, S.K. Baruah, and B.B. Brandenburg. Multicore operating-system support for mixed criticality. In *Proc. of the Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification, San Francisco*, 2009.

[15] E. Armbrust, J. Song, G. Bloom, and G. Parmer. On spatial isolation for mixed-criticality, embedded systems. In L. Cucu-Grosjean and R. Davis, editors, *Proc. 2nd Workshop on Mixed Criticality Systems (WMC), RTSS*, pages 15–20, 2014.

[16] S. Asyaban, M. Kargahi, L. Thiele, and M. Mohaqeqi. Analysis and scheduling of a battery-less mixed-criticality system with energy uncertainty. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(1):23, 2016.

[17] N.C. Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, 2001.

[18] N.C. Audsley. Memory architectures for NoC-based real-time mixed criticality systems. In *Proc. WMC, RTSS*, pages 37–42, 2013.

[19] N.C. Audsley, A. Burns, M. Richardson, K. Tindell, and A.J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[20] M.A. Awan, K. Bletsas, P. Souto, B. Akesson, E. Tovar, and J. Ali. Mixed-criticality scheduling with memory regulation. In *Proc. WiP, ECRTS*, page 22, 2016.

[21] M.A. Awan, K. Bletsas, P. Souto, and E. Tovar. Semi-partitioned mixed-criticality scheduling. Technical report, TR. CISTER/ISEP. http://www. cister. isep. ipp. pt/-docs, 2016.

[22] M.A. Awan, D. Masson, and E. Tovar. Energy-aware task allocation onto unrelated heterogeneous multicore platform for mixed criticality systems. In *WiP, RTSS*, 2015.

[23] P. Axer, M. Sebastian, and R. Ernst. Reliability analysis for mpsocs with mixed-critical, hard real-time constraints. In *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '11, pages 149–158. ACM, 2011.

[24] A. Azim and S. Fischmeister. Efficient mode changes in multi-mode systems. In *Proc. Computer Design (ICCD)*, pages 592–599. IEEE, 2016.

[25] T.P. Baker. A stack-based resource allocation policy for realtime processes. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*, pages 191–200, 1990.

[26] J. Barhorst, T. Belote, P. Binns, J. Hoffman, J. Paunicka, P. Sarathy, J. Scoredos, P. Stanfill, D. Stuart, and R. Urzi. White paper: A research agenda for mixed-criticality systems, April 2009. Available at http://www.cse.wustl.edu/˜ cdgill/CPSWEEK09_MCAR.

[27] S. K. Baruah, A. Burns, and Z. Guo. Scheduling mixed-criticality systems to guarantee some service under all non-erroneous behaviours. In *Proc. ECRTS*, pages 131–140, 2016.

[28] S.K. Baruah. Optimal utilization bounds for fixed priority scheduling of periodic task systems on identical multiprocessors. *IEEE Transactions on Software Engineering*, 53(6), 2004.

[29] S.K. Baruah. Mixed criticality schedulability analysis is highly intractable. Technical report, University of North Carolina at Chapel Hill, 2009.

[30] S.K. Baruah. Certification-cognizant scheduling of tasks with pessimistic frequency specification. In *Proc. 7th IEEE International Symposium on Industrial Embedded Systems (SIES'12)*, pages 31–38, 2012.

[31] S.K. Baruah. Semantic-preserving implementation of multirate mixed criticality synchronous programs. In *Proc. RTNS*, 2012.

[32] S.K. Baruah. Implementing mixed-criticality synchronous reactive programs upon uniprocessor platforms. *Real-Time Systems Journal*, 49(6), 2013.

[33] S.K. Baruah. Implementing mixed criticality synchronous reactive systems upon multiprocessor platforms. Technical report, University of North Carolina at Chapel Hill, 2013.

[34] S.K. Baruah. Response-time analysis of mixed criticality systems with pessimistic frequency specification. Technical report, University of North Carolina at Chapel Hill, 2013.

[35] S.K Baruah. The federated scheduling of systems of mixed-criticality sporadic DAG tasks. In *Proc. Real-Time Systems Symposium (RTSS)*, pages 227–236. IEEE, 2016.

[36] S.K. Baruah. Schedulability analysis of mixed-criticality systems with multiple frequency specifications. In *Proc. International Conference on Embedded Software (EMSOFT)*, page 24. ACM, 2016.

[37] S.K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. Scheduling real-time mixed-criticality jobs. In P. Hliněný and A.ín Kucera, editors, *Proc. of the 35th International Symposium on the Mathematical Foundations of Computer Science*, volume 6281 of *Lecture Notes in Computer Science*, pages 90–101. Springer, 2010.

[38] S.K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. Mixed-criticality scheduling. In *10th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP), Nymburk, Czech Republic*, pages 1–3, 2011.

[39] S.K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. Scheduling real-time mixed-criticality jobs. *IEEE Transactions on Computers*, 61(8):1140–1152, 2012.

[40] S.K. Baruah, V. Bonifaci, G. D'angelo, H. Li, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie. Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems. *Journal of the ACM (JACM)*, 62(2):14, 2015.

[41] S.K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *Proc. of ECRTS, Pisa*, pages 145–154, 2012.

[42] S.K. Baruah, V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. Mixed-criticality scheduling of sporadic task systems. In *Proc. of the 19th Annual European Symposium on Algorithms (ESA 2011) LNCS 6942, Saarbruecken, Germany*, pages 555–566, 2011.

[43] S.K. Baruah and A. Burns. Implementing mixed criticality systems in Ada. In A. Romanovsky, editor, *Proc. of Reliable Software Technologies - Ada-Europe 2011*, pages 174–188. Springer, 2011.

[44] S.K. Baruah and A. Burns. Fixed-priority scheduling of dual-criticality systems. In *Proc. 21st RTNS*, pages 173–182. ACM, 2013.

[45] S.K. Baruah and A. Burns. Achieving temporal isolation in multiprocessor mixed-criticality systems. In L. Cucu-Grosjean and R. Davis, editors, *Proc. 2nd Workshop on Mixed Criticality Systems (WMC), RTSS*, pages 21–26, 2014.

[46] S.K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*, pages 34–43, 2011.

[47] S.K. Baruah, A. Burns, and R.I. Davis. An extended fixed priority scheme for mixed criticality systems. In L. George and G. Lipari, editors, *Proc. ReTiMiCS, RTCSA*, pages 18–24, 2013.

[48] S.K. Baruah and B. Chattopadhyay. Response-time analysis of mixed criticality systems with pessimistic frequency specification. In *Proc. RTCSA*, 2013.

[49] S.K. Baruah, B. Chattopadhyay, H. Li, and I. Shin. Mixed-criticality scheduling on multiprocessors. *Real-Time Systems Journal*, 50:142–177, 2014.

[50] S.K. Baruah, A. Easwaran, and Z. Guo. MC-Fluid: Simplified and optimally quantified. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*, pages 327–337, 2015.

[51] S.K. Baruah, A. Easwaran, and Z. Guo. Mixed-criticality scheduling to minimize makespan. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 65, 2016.

[52] S.K. Baruah and G. Fohler. Certification-cognizant time-triggered scheduling of mixed-criticality systems. In *Proc. of IEEE Real-time Systems Symposium 2011*, December 2011.

[53] S.K. Baruah and Z. Guo. Mixed criticality scheduling upon unreliable processors. Technical report, University of North Carolina at Chapel Hill, 2013.

[54] S.K. Baruah and Z. Guo. Mixed-criticality scheduling upon varying-speed processors. In *Proc. IEEE 34th Real-Time Systems Symposium*, pages 68–77, 2013.

[55] S.K. Baruah and Z. Guo. Scheduling mixed-criticality implicit-deadline sporadic task systems upon a varying-speed processor. In *Proc. IEEE Real-Time Systems Symposium*, pages 31–400. IEEE, 2014.

[56] S.K. Baruah and Z. Guo. Mixed-criticality job models: a comparison. In L. Cucu-Grosjean and R. Davis, editors, *Proc. 3rd Workshop on Mixed Criticality Systems (WMC), RTSS*, pages 1–5, 2015.

[57] S.K. Baruah, H. Li, and L. Stougie. Mixed-criticality scheduling: Improving resource-augmented results. In *Computers and Their Applications, ISCA*, pages 217–223, 2010.

[58] S.K. Baruah, H. Li, and L. Stougie. Towards the design of certifiable mixed-criticality systems. In *Proc. of the IEEE Real-Time Technology and Applications Symposium (RTAS)*, pages 13–22. IEEE, 2010.

[59] S.K. Baruah and S. Vestal. Schedulability analysis of sporadic tasks with multiple criticality specifications. In *ECRTS*, pages 147–155, 2008.

[60] S.K. Barugh. Scheduling analysis for a general model of mixed-criticality recurrent real-time tasks. In *Proc. IEEE RTSS*, pages 25–34, 2016.

[61] I. Bate, A. Burns, and R. I. Davis. An enhanced bailout protocol for mixed criticality embedded software. *IEEE Transactions on Software Engineering*, PP(99), 2016.

[62] I. Bate, A. Burns, and R.I. Davis. A bailout protocol for mixed criticality systems. In *Proc. 27th ECRTS*, pages 259–268, 2015.

[63] G. Bernat and A. Burns. New results on fixed priority aperiodic servers. In *Proc. 20th IEEE Real-Time Systems Symposium*, pages 68–78, 1999.

[64] G. Bernat and A. Burns. Multiple servers and capacity sharing for implementing flexible scheduling. *Real-Time Systems Journal*, 22:49–75, 2002.

[65] G. Bernat, A. Colin, and S.M. Petters. Wcet analysis of probabilistic hard real-time systems. In *23rd IEEE Real-Time Systems Symposium*, pages 279–288. IEEE, 2002.

[66] K.J. Biba. Integrity considerations for secure computer systems. Mtr-3153, Mitre Corporation, 1977.

[67] E. Bini, M. Di Natale, and G.C. Buttazzo. Sensitivity analysis for fixed-priority real-time systems. In *Proc. ECRTS*, pages 13–22, 2006.

[68] K. Bletsas and S.M. Petters. Using NPS-F for mixed criticality systems. In *Proc. WiP, RTSS*, page 25, 2012.

[69] K. Bletsas and S.M. Petters. Using NPS-F for mixed criticality multicore ystems. Cister-tr-130303, CISTER, 2013.

[70] A. Blin, C. Courtaud, J. Sopena, and G. Muller. Maximizing parallelism without exploding deadlines in a mixed-criticality embedded system. In *Proc. ECRTS*, pages 109–119, 2016.

[71] M. Bommert. Schedule-aware distributed of parallel load in a mixed criticality environment. In *Proc. JRWRTC, RTNS*, pages 21–24, 2013.

[72] B.B. Brandenburg. A synchronous IPC protocol for predicatable access to shared resources in mixed-criticality systems. In *Proc. IEEE Real-Time Systems Symposium*, pages 196–206. IEEE, 2014.

[73] F. Broekaert, A. Fritsch, L. Sa, and S. Tverdyshev. Towards power-efficient mixed-critical systems. In *Proc. of OSPERT 2013*, pages 30–35, 2013.

[74] I. Broster and A. Burns. An analysable bus-guardian for event-triggered communication. In *Proc. of the 24th Real-time Systems Symposium*, pages 410–419, Cancun, Mexico, Dec 2003. Computer Society, IEEE.

[75] A. Burns. Preemptive priority based scheduling: An appropriate engineering approach. In S.H. Son, editor, *Advances in Real-Time Systems*, pages 225–248. Prentice-Hall, 1994.

[76] A. Burns. The application of the original priority ceiling protocol to mixed criticality systems. In L. George and G. Lipari, editors, *Proc. ReTiMiCS, RTCSA*, pages 7–11, 2013.

[77] A. Burns. System mode changes - general and criticality-based. In L. Cucu-Grosjean and R. Davis, editors, *Proc. 2nd Workshop on Mixed Criticality Systems (WMC), RTSS*, pages 3–8, 2014.

[78] A. Burns. An augmented model for mixed criticality. In Davis Baruah, Cucu-Grosjean and Maiza, editors, *Mixed Criticality on Multicore/Manycore Platforms (Dagstuhl Seminar 15121)*, volume 5(3), pages 92–93. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2015.

[79] A. Burns and S.K. Baruah. Timing faults and mixed criticality systems. In Jones and Lloyd, editors, *Dependable and Historic Computing*, volume LNCS 6875, pages 147–166. Springer, 2011.

[80] A. Burns and S.K. Baruah. Towards a more practical model for mixed criticality systems. In *Proc. 1st Workshop on Mixed Criticality Systems (WMC), RTSS*, pages 1–6, 2013.

[81] A. Burns and S.K. Baruah. Semi-partitioned cyclic executives for mixed criticality systems. In L. Cucu-Grosjean and R. Davis, editors, *Proc. 3rd Workshop on Mixed Criticality Systems (WMC), RTSS*, pages 36–41, 2015.

[82] A. Burns and R.I. Davis. Mixed criticality on controller area network. In *Proc. Euromicro Conference on Real-Time Systems (ECRTS)*, pages 125–134, 2013.

[83] A. Burns and R.I. Davis. Adaptive mixed criticality scheduling with deferred preemption. In *Proc. IEEE Real-Time Systems Symposium*, pages 21–30. IEEE, 2014.

[84] A. Burns and R.I. Davis. Mixed criticality systems: A review. Technical Report MCC-1(H), available at http://www-users.cs.york.ac.uk/burns/review.pdf, Department of Computer Science, University of York, 2016.

[85] A. Burns, T. Fleming, and S.K. Baruah. Cyclic executives, multi-core platforms and mixed-criticality applications. In *Proc. 27th ECRTS*, pages 3–12, 2015.

[86] A. Burns, J. Harbin, and L.S. Indrusiak. A Wormhole NoC protocol for mixed criticality systems. In *Proc. IEEE Real-Time Systems Symposium*, pages 184–195. IEEE, 2014.

[87] A. Burns and T.J. Quiggle. Effective use of abort in programming mode changes. *Ada Letters*, 1990.

[88] A. Burns and A. J. Wellings. *Real-Time Systems and Programming Languages*. Addison Wesley Longman, 4th edition, 2009.

[89] A. Burns and A.J. Wellings. Dual priority scheduling in Ada 95 and real-time POSIX. In *Proc. of 21th IFAC/IFIP Workshop on Real-Time Programming (WRTP96),*, 1996.

[90] G. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *IEEE Real-Time Systems Symposium*, pages 286–295, 1998.

[91] G.C. Buttazzo. *Hard Real-Time Computing Systems*. Springer, 2005.

[92] G. Carvajal and S. Fischmeister. An open platform for mixed-criticality real-time ethernet. In *Proceedings of the Conference on Design, Automation and Test in Europe*, Proc. DATE, pages 153–156, 2013.

[93] F.J. Cazorla, E. Quiones, T. Vardanega, L. Cucu-Grosjean, B. Triquet, G. Bernat, E.D. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim. PROARTIS: Probabilistically analyzable real-time systems. *ACM Trans. Embedded Comput. Syst.*, 12(2):94, 2013.

[94] S. Chaki and D. Kyle. Dmpl: Programming and verifying distributed mixed-synchrony and mixed-critical software. *Technical Report CMU/SEI-2016-TR-005*, 2016.

[95] F. Checconi, T. Cucinotta, D. Faggioli, and G. Lipari. Hierarchical multiprocessor cpu reservations for the linux kernel. In *Proc. of 5th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2009)*, 2009.

[96] Y. Chen, K.G. Shin, and H. Xiong. Generalizing fixed-priority scheduling for better schedulability in mixed-criticality systems. *Information Processing Letters*, 116(8):508–512, 2016.

[97] H. Chetto and M. Chetto. Some results of the earliest deadline scheduling algorithm. *IEEE Transactions on Software Engineering*, 15(10):1261–1269, 1989.

[98] M. Chisholm, N. Kim, B.C. Ward, N. Otterness, J.H. Anderson, and F.D. Smith. Reconciling the tension between hardware isolation and data sharing in mixed-criticality, multicore systems. In *Proc. Real-Time Systems Symposium (RTSS)*, pages 57–68. IEEE, 2016.

[99] M. Chisholm, B. Ward, N. Kim, and J. Anderson. Cache sharing and isolation tradeoffs in multicore mixed-criticality systems. Technical report, University of North Carolina, 2015.

[100] M. Chrisholm, B. Ward, N. Kim, and J. Anderson. Cache-sharing and isolation tradeoffs in multicore mixed-criticality systems. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*, pages 305–316, 2015.

[101] B. Cilku, A. Crespo, P. Puschner, J. Coronel, and S. Peiro. A memory arbitration scheme for mixed-criticality multocore platforms. In L. Cucu-Grosjean and R. Davis, editors, *Proc. 2nd Workshop on Mixed Criticality Systems (WMC), RTSS*, pages 27–32, 2014.

[102] B. Cilku, A. Crespo, P. Puschner, J. Coronel, and S. Peiro. A TDMA-based arbitration scheme for mixed-criticality multicore platforms. In *Proc EBCCSP*, pages 1–6. IEEE, 2015.

[103] B. Cilku and P. Puschner. Towards temporal and spatial isolation in memory hierarchies for mixed-criticality systems with hypervisors. In L. George and G. Lipari, editors, *Proc. ReTiMiCS, RTCSA*, pages 25–28, 2013.

[104] A. Cohen, V. Perrelle, D. Potop-Butucaru, E. Soubiran, and Z. Zhang. Mixed-criticality in railway systems: A case study on signaling application. *Ada User Journal, Proc of Workshop on Mixed Criticality for Industrial Systems (WMCIS'2014)*, 35(2):138–143, 2014.

[105] A. Crespo, A. Alonso, M. Marcos, J.A. Puente, and P. Balbastre. Mixed criticality in control systems. In *Proc. 19th World Congress The Federation of Automatic Control*, pages 12261–12271, 2014.

[106] O. Cros, F. Fauberteau, L. George, and X. Li. Mixed-criticality over switched ethernet networks. *Ada User Journal, Proc of Workshop on Mixed Criticality for Industrial Systems (WMCIS'2014)*, 35(2):138–143, 2014.

[107] O. Cros, L. George, and X.Li. A protocol for mixed-criticality management in switched ethernet networks. In L. Cucu-Grosjean and R. Davis, editors, *Proc. 3rd Workshop on Mixed Criticality Systems (WMC), RTSS*, pages 12–17, 2015.

[108] L. Cucu-Grosjean. Independence - a misunderstood property of and for probabilistic real-time systems. In N. Audsley and S.K. Baruah, editors, *In Real-Time Systems: the past, the present and the future*, pages 29–37, 2013.

[109] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiones, and F.J. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *Proc. 24th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 91–101, 2012.

[110] V. David, A. Barbot, and D. Chabrol. Dependable real-time system and mixed criticality: Seeking safety, flexibility and efficiency with Kron-OS. *Ada User Journal*, 35(4):259–265, 2014.

[111] R.I. Davis. *On exploiting spare capacity in hard real-time systems*. PhD thesis, University of York, UK, 1995.

[112] R.I. Davis and M. Bertogna. Optimal fixed priority scheduling with deferred preemption. In *Proc. IEEE Real-Time Systems Symposium*, pages 39–50, 2012.

[113] R.I. Davis and A. Burns. Hierarchical fixed priority preemptive scheduling. In *Proc. of IEEE Real-Time Systems Symposium (RTSS)*, pages 389–398, 2005.

[114] R.I. Davis and A. Burns. Resource sharing in hierarchical fixed priority preemptive systems. In *Proceeding IEEE Real-Time Systems Symposium (RTSS)*, 2006.

[115] R.I. Davis and A. Burns. Robust priority assignment for fixed priority real-time systems. In *Proc. of IEEE Real-Time Systems Symposium (RTSS)*, 2007.

[116] R.I. Davis, A. Burns, R.J. Bril, and J.J. Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Journal of Real-Time Systems*, 35(3):239–272, 2007.

[117] R.I. Davis, L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean. Analysis of probabilistic cache related pre-emption delays. In *ECRTS*, pages 129–138, 2013.

[118] R.I. Davis, K. Tindell, and A. Burns. Scheduling slack time in fixed priority preemptive systems. In *Proc. 14th IEEE Real-Time Systems Symposium*, 1993.

[119] R.I. Davis and A. J. Wellings. Dual priority scheduling. In *Proc. 16th IEEE Real-Time Systems Symposium*, pages 100–109, 1995.

[120] J.L. Díaz, D.F. Garcia, K. Kim, C.G. Lee, L.L. Bello, J.M. López, and O. Mirabella. Stochastic analysis of periodic real-time systems. In *IEEE Real-Time Systems Symposium (RTSS)*, 2002.

[121] J. Diemer and R. Ernst. Back suction: Service guarantees for latency-sensitive on-chip networks. In *Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, Proc. NOCS '10, pages 155–162. IEEE Computer Society, 2010.

[122] A.C. Dimopoulos, G. Bravos, G. Dimitrakopoulos, M. Nikolaidou, V. Nikolopoulos, and D. Anagnostopoulos. A multi-core context-aware management architecture for mixed-criticality smart building applications. In *Proc. System of Systems Engineering Conference (SoSE)*, pages 1–6. IEEE, 2016.

[123] F. Dorin, P. Richard, M. Richard, and J. Goossens. Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities. *Real-Time Systems Journal*, 46(3):305–331, 2010.

[124] S. Draskovic, P. Huang, and L. Thiele. On the safety of mixed-criticality scheduling. In *Proc. 4th WMC (RTSS)*, page 6, 2016.

[125] C. Dürr, Z. Hanzálek, C. Konrad, R. Sitters, O.C. Vásquez, and G. Woeginger. The triangle scheduling problem. *arXiv preprint arXiv:1602.04365*, 2016.

[126] A. Easwaran. Demand-based scheduling of mixed-criticality sporadic tasks on one processor. In *Proc. IEEE 34th Real-Time Systems Symposium*, pages 78–87, 2013.

[127] A. Easwaran and I. Shin. Compositional mixed-criticality scheduling. *CRTS 2014*, 2014.

[128] L. Ecco, S. Tobuschat, S. Saidi, and R. Ernst. A mixed critical memory controller using bank privatization and fixed priority scheduling. In *Proc. Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–10. IEEE, 2014.

[129] S. Edgar and A. Burns. Statistical analysis of WCET for scheduling. In *Proc. 22nd IEEE Real-Time Systems Symposium*, 2001.

[130] P. Ekberg, M. Stigge, N. Guan, and W. Yi. State-based mode switching with applications to mixed criticality systems. In *Proc. WMC, RTSS*, pages 61–66, 2013.

[131] P. Ekberg and W. Yi. Bounding and shaping the demand of mixed-criticality sporadic task systems. In *ECRTS*, pages 135–144, 2012.

[132] P. Ekberg and W. Yi. Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. *Journal of Real-Time Systems*, 50:48–86, 2014.

[133] P. Ekberg and W. Yi. A note on some open problems in mixed-criticality scheduling. In *Proc. RTOPS, 27th ECRTS*, pages 1–2, 2015.

[134] P. Ekberg and W. Yi. Schedulability analysis of a graph-based task model for mixed-criticality systems. *Real-Time Systems*, 52:1–37, 2016.

[135] P. Emberson and I. Bate. Minimising task migrations and priority changes in mode transitions. In *Proc. of the 13th IEEE Real-Time And Embedded Technology And Applications Symposium (RTAS 07)*, pages 158–167, 2007.

[136] B. Engel. Tightening critical section bounds in mixed-criticality systems through preemptible hardware transactional memory. In *Proc. OSPERT*, pages 17–22, 2016.

[137] R. Ernst and M. Di Natale. Mixed criticality systems?a history of misconceptions? *IEEE Design & Test*, 33(5):65–74, 2016.

[138] A. Esper, G. Neilissen, V. Neils, and E. Tovar. How realistic is the mixed-criticality real-time system model. In *23rd International Conference on Real-Time Networks and Systems (RTNS 2015)*, pages 139–148, 2015.

[139] C. Evripidou and A. Burns. Scheduling for mixed-criticality hypervisor systems in the automotive domain. In *Proc. 4th WMC (RTSS)*, page 6, 2016.

[140] G. Farrall, C. Stellwag, J. Diemer, and R. Ernst. Hardware and software support for mixed-criticality multicore systems. In *Proceedings of the Conference on Design, Automation and Test in Europe, WICERT*, DATE, 2013.

[141] T. Fleming, S.K. Baruah, and A. Burns. Improving the schedulability of mixed criticality cyclic executives via limited task splitting. In *Proc. of the 24th International Conference RTNS*, pages 277–286. ACM, 2016.

[142] T. Fleming and A. Burns. Extending mixed criticality scheduling. In *Proc. WMC, RTSS*, pages 7–12, 2013.

[143] T. Fleming and A. Burns. Incorporating the notion of importance into mixed criticality systems. In L. Cucu-Grosjean and R. Davis, editors, *Proc. 2nd Workshop on Mixed Criticality Systems (WMC), RTSS*, pages 33–38, 2014.

[144] T. Fleming and A. Burns. Investigating mixed criticality cyclic executive schedule generation. In L. Cucu-Grosjean and R. Davis, editors, *Proc. 3rd Workshop on Mixed Criticality Systems (WMC), RTSS*, pages 42–47, 2015.

[145] T. Fleming and A. Burns. Utilising asymmetric parallelism in multi-core mcs implemented via cyclic executives. In *Proc. 4th WMC (RTSS)*, page 6, 2016.

[146] O. Gettings, S. Quinton, and R.I. Davis. Mixed criticality systems with weakly-hard constraints. In *23rd International Conference on Real-Time Networks and Systems (RTNS 2015)*, pages 237–246, 2015.

[147] G. Giannopoulou and et al. DOL-BIP-Critical: A tool chain for rigorous design and implementation of mixed-criticality multi-core systems. Technical Report TR-2016-363, Verimag, 2016.

[148] G. Giannopoulou, P. Huang, A. Gomez, and L. Thiele. Mixed-criticality runtime mechanisms and evaluation on multicore. In *Proc. RTAS*, 2015.

[149] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele. Scheduling of mixed-criticality applications on resource-sharing multicore systems. In *Proc. Int. Conference on Embedded Software (EMSOFT)*, Montreal, 2013.

[150] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele. Mapping mixed-criticality applications on multi-core architectures. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1–6. IEEE, 2014.

[151] G. Giannopoulou, N. Stoimenov, P. Huang, L. Thiele, and B. D. de Dinechin. Mixed-criticality scheduling on cluster-based manycores with shared communication and storage resources. *Real-Time Systems*, pages 1–51, 2015.

[152] Robert Bosch GmbH. CAN specification version 2.0. Technical report, Postfach 30 02 40, D-70442 Stuttgart, 1991.

[153] M. Gomony, J. Garside, B. Akesson, N. Audsley, and K. Goossens. A globally arbitrated memory tree for mixed-time-criticality systems. *IEEE Transactions on Computers*, 2016.

[154] K. Goossens, A. Azevedo, K. Chandrasekar, M.D. Gomony, S. Goossens, M. Koedam, Y. Li, D. Mirzoyan, A. Molnos, A.B. Nejad, A. Nelson, and S. Sinha. Virtual execution platforms for mixed-time-criticality systems: The compsoc architecture and design flow. *SIGBED Rev.*, 10(3):23–34, 2013.

[155] S. Goossens, B. Akesson, and K. Goossens. Conservative open-page policy for mixed time-criticality memory controllers. In *Proc. DATE*, pages 525–530, 2013.

[156] S. Goossens, J. Kuijsten, B. Akesson, and K. Goossens. A reconfigurable real-time SDRAM controller for mixed time-criticality systems. In *Int'l Conf. on Hardware/-Software Codesign and System Synthesis (CODES+ISSS)*, 2013.

[157] D. Goswami, M. Lukasiewycz, R. Schneider, and S. Chakraborty. Time-triggered implementations of mixed-criticality automotive software. In *Proceedings of the Conference on Design, Automation and Test in Europe*, Proc. DATE, pages 1227–1232, 2012.

[158] R. Gratia, T. Robert, and L. Pautet. Adaptation of RUN to mixed-criticality systems. In *Proc. 8th Junior Researcher Workshop on Real-Time Computing, RTNS*, 2014.

[159] R. Gratia, T. Robert, and L. Pautet. Generalized mixed-criticality scheduling based on RUN. In *23rd International Conference on Real-Time Networks and Systems (RTNS 2015)*, pages 267–276, 2015.

[160] R. Gratia, T. Robert, and L. Pautet. Scheduling of mixed-criticality systems with run. In *Proc. ETFA*, pages 1–8. IEEE, 2015.

[161] P. Graydon and I. Bate. Safety assurance driven problem formulation for mixed-criticality scheduling. In *Proc. WMC, RTSS*, pages 19–24, 2013.

[162] D. Griffin, I. Bate, B. Lesage, and F. Soboczenski. Evaluating mixed criticality scheduling algorithms with realistic workloads. In L. Cucu-Grosjean and R. Davis, editors, *Proc. 3rd Workshop on Mixed Criticality Systems (WMC), RTSS*, pages 24–29, 2015.

[163] S. Groesbrink, L. Almeida, M. de Sousa, and S.M. Petters. Towards certifiable adaptive reservations for hypervisor-based virtualization. In *Proc. of the 20th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014.

[164] S. Groesbrink, S. Oberthr, and D. Baldin. Architecture for adaptive resource assignment to virtualized mixed-criticality real-time systems. In *Special Issue on the 4th Workshop on Adaptive and Reconfigurable Embedded Systems (APRES 2012)*, volume 10(1). ACM SIGBED Review, 2013.

[165] K. Grüttner. Empowering mixed-criticality system engineers in the dark silicon era: Towards power and temperature analysis of heterogeneous mpsocs at system level. In *Model-Implementation Fidelity in Cyber Physical System Design*, pages 57–90. Springer, 2017.

[166] C. Gu, N. Guan, Q. Deng, and W. Yi. Partitioned mixed-criticality scheduling on multiprocessor platforms. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6. IEEE, 2014.

[167] C. Gu, N. Guan, Q. Deng1, and W. Yi. Improving ocbp-based scheduling for mixed-criticality sporadic task systems. In *Proc. RTCSA*, 2013.

[168] X. Gu and A. Easwaran. Optimal speedup bound for 2-level mixed-criticality arbitrary deadline systems. In *Proceedings RTSOPS (ECRTS)*, pages 15–16, 2014.

[169] X. Gu and A. Easwaran. Dynamic budget management with service guarantees for mixed-criticality systems. In *Proc. Real-Time Systems Symposium (RTSS)*, pages 47–56. IEEE, 2016.

[170] X. Gu, A. Easwaran, K.M. Phan, and I. Shin. Compositional mixed-criticality scheduling. Technical report, Nanyang Technological University, Singapore, 2014.

[171] X. Gu, K.-M. Phan, A. Easwaran, and I. Shin. Resource efficient isolation mechanisms in mixed-criticality scheduling. In *Proc. 27th ECRTS*, pages 13–24. IEEE, 2015.

[172] N. Guan, P. Ekberg, M. Stigge, and W. Yi. Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems. In *IEEE RTSS*, pages 13–23, 2011.

[173] N. Guan and W. Yi. Improveing the scheduling of certifiable mixed criticality sopradic task systems. Technical report, University of Uppsala, 2012.

[174] Z. Guo. Mixed-criticality scheduling on varying-speed platforms with bounded performance drop rate. In *Proc SMARTCOMP*, pages 1–3. IEEE, 2016.

[175] Z. Guo and S.K. Baruah. Implementing mixed-criticality systems upon a preemptive varying-speed processor. *Leibniz Transactions on Embedded Systems*, 1(2):03–103:19, 2014.

[176] Z. Guo and S.K. Baruah. The concurrent consideration of uncertainty in WCETs and processor speeds in mixed criticality systems. In *23rd International Conference on Real-Time Networks and Systems (RTNS 2015)*, pages 247–256, 2015.

[177] Z. Guo, L. Santinelli, and K. Yang. EDF schedulability analysis on mixed-criticality systems with permitted failure probability. In *Proc. RTCSA*, 2015.

[178] H. Hamza, A. Hughes, and R. Kirner. On the design of a Java virtual machine for mixed-criticality systems. In *Proc. JTRES*. ACM, 2015.

[179] J.J. Han, X. Tao, D. Zhu, and H. Aydin. Criticality-aware partitioning for multicore mixed-criticality systems. In *Proc. Parallel Processing (ICPP)*, pages 227–235. IEEE, 2016.

[180] Z. Hanzálek, T. Tunys, and P. Sucha. An analysis of the non-preemptive mixed-criticality match-up scheduling problem. *Journal of Scheduling*, pages 1–7, 2016.

[181] J. Harbin, T. Fleming, L.S. Indrusiak, and A. Burns. GMCB: An industrial benchmark for use in real-time mixed-criticality networks-on-chip. In *Proc. WATERS, 27th ECRTS*, 2015.

[182] P. Haririan and A. Garcia-Ortiz. A framework for hardware-based DVFS management in multicore mixed-criticality systems. In *Proc. 10th Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–7. IEEE, 2015.

[183] M. Hassan and H. Patel. Criticality-and requirement-aware bus arbitration for multi-core mixed criticality systems. In *Proc RTAS*, pages 1–11. IEEE, 2016.

[184] M. Hassan, H. Patel, and R. Pellizzoni. A framework for scheduling dram memory accesses for multi-core mixed-time critical systems. In *Proc. RTAS*, pages 307–316. IEEE, 2015.

[185] C. Herber, A. Richter, H. Rauchfuss, and A. Herkersdorf. Spatial and temporal isolation of virtual can controllers. In *Proc. VtRES, RTCSA*, 2013.

[186] J. Herman, C. Kenna, M. Mollison, J. Anderson, and D. Johnson. RTOS support for multicore mixed-criticality systems. In *Proc. of the 18th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2012.

[187] F. Herrera, S.H.A. Niaki, and I. Sander. Towards a modelling and design framework for mixed-criticality socs and systems-of-systems. In *Proc. 16th Euromicro Conf. on Digital Systems Design*, pages 989–996, 2013.

[188] F. Herrera, P. Penil, and E. Villar. A model-based, single-source approach to design-space exploration and synthesis of mixed-criticality systems. In *Proc. SCOPES*, pages 88–91, 2015.

[189] M. Hikmet, M.M. Kuo, P.S. Roop, and P. Ranjitkar. Mixed-criticality systems as a service for non-critical tasks. In *Proc. ISORC*, pages 221–228, 2016.

[190] M.G. Hill and T.W. Lake. Non-interference analysis for mixed criticality code in avionics systems. In *Proceedings of the 15th IEEE international conference on Automated software engineering*, pages 257–260. IEEE Computer Society, 2000.

[191] T. Hollstein, S.P Azad, T. Kogge, and B. Niazmand. Mixed-criticality NoC partitioning based on the NoCDepend dependability technique. In *Proc. 10th Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–8. IEEE, 2015.

[192] P. Holman and J.H. Anderson. Adapting Pfair scheduling for symmetric multiprocessors. *Journal of Embedded Computing*, 1(4):543–564, 2005.

[193] B. Hu, K. Huang, G. Chen, L. Cheng, and A. Knoll. Adaptive runtime shaping for mixed-criticality systems. In *Proc. 12th International Conference on Embedded Software, EMSOFT*, pages 11–20. IEEE Press, 2015.

[194] B. Hu, K. Huang, G. Chen, L. Cheng, and A. Knoll. Adaptive workload management in mixed-criticality systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(1):14, 2016.

[195] B. Hu, K. Huang, P. Huang, L. Thiele, and A. Knoll. On-the-fly fast overrun budgeting for mixed-criticality systems. In *Proc. International Conference on Embedded Software (EMSOFT)*, pages 1–10. IEEE, 2016.

[196] H-M. Huang, C. Gill, and C. Lu. Implementation and evaluation of mixed criticality scheduling approaches for periodic tasks. In *Proc. of the IEEE Real-Time Technology and Applications Symposium (RTAS)*, pages 23–32, 2012.

[197] H-M. Huang, C. Gill, and C. Lu. Implementation and evaluation of mixed criticality scheduling approaches for sporadic tasks. *ACM Trans. Embedded Systems*, 13:126:1– 126:25, 2014.

[198] P. Huang, G. Giannopoulou, R. Ahmed, D.B. Bartolini, and L. Thiele. An isolation scheduling model for multicores. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*, pages 141–152, 2015.

[199] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele. Service adaptions for mixed-criticality systems. Technical Report 350, ETH Zurich, Laboratory TIK, 2013.

[200] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele. Service adaptions for mixed-criticality systems. In *19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, Singapore, 2014.

[201] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele. Energy efficient DVFS scheduling for mixed-criticality systems. In *Proc. Embedded Software (EMSOFT)*, pages 1–10. IEEE, 2014.

[202] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele. Run and be safe: mixed-criticality scheduling with temporal processor speedup. In *Proc. DATE*, 2015.

[203] P. Huang, P. Kumar, N. Stoimenov, and L. Thiele. Interference constraint grapha new specification for mixed-criticality systems. In *Proc. 18th Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, 2013.

[204] P. Huang, H. Yang, and L. Thiele. On the scheduling of fault-tolerant mixed-criticality systems. Technical report, Technical Report 351, ETH Zurich, Laboratory TIK, 2013.

[205] P. Huang, H. Yang, and L. Thiele. On the scheduling of fault-tolerant mixed-criticality systems. In *Proc. Design Automation Conference (DAC)*, pages 1–6. IEEE, 2014.

[206] B. Huber, C. El Salloum, and R. Obermaisser. A resource management framework for mixed-criticality embedded systems. In *34th IEEE IECON*, pages 2425–2431, 2008.

[207] L.S. Indrusiak, J. Harbin, and A. Burns. Average and worst-case latency improvements in mixed-criticality wormhole networks-on-chip. In *Proc. European/Euromicro Conference on Real-Time Systems (ECRTS)*, pages 47–56. IEEE, 2015.

[208] S. Islam, R. Lindstrom, and N.Suri. Dependability driven integration of mixed criticality SW components. In *9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, ISORC 2006*, page 11, 2006.

[209] P. Ittershagen, K. Gruttner, and W. Nebel. Mixed-criticality system modelling with dynamic execution mode switching. Technical report, Technical Report OFFIS, Oldenburg, Germany, 2015.

[210] V. Izosimov and E. Levholt. Mixed criticality metric for safety-critical cyber-physical systems on multicore archiectures. *MEDIAN: Methods*, 2(8), 2015.

[211] J. Jalle, E. Quinones, J. Abella, L. Fossati, M. Zulianello, and F.J. Cazorla. A dual-criticality memory controler (DCmc): Proposal and evaluation of a space case study. In *Proc. IEEE Real-Time Systems Symposium*, pages 207–217. IEEE, 2014.

[212] M. Jan, L. Zaourar, V. Legout, and L. Pautet. Handling criticality mode change in time-triggered systems through linear programming. *Ada User Journal, Proc of Workshop on Mixed Criticality for Industrial Systems (WMCIS'2014)*, 35(2):138–143, 2014.

[213] M. Jan, L. Zaourar, and M. Pitel. Maximizing the execution rate of low criticality tasks in mixed criticality system. In *Proc. 1st WMC, RTSS*, pages 43–48, 2013.

[214] X. Jin, J. Wang, and P. Zeng. End-to-end delay analysis for mixed-criticality wirelesshart networks. *Automatica Sinica*, 2(3):282–289, 2015.

[215] X. Jin, C. Xia, H. Xu, J. Wang, and P. Zeng. Mixed criticality scheduling for industrial wireless sensor networks. *Sensors*, 16(9):1376, 2016.

[216] C.B. Jones. Tentative steps toward a development method for interfering programs. *Transactions on Programming Languages and System*, 5(4):596–619, 1983.

[217] R. Kaiser. The PikeOS concept history and design,. Technical Report white paper, SYSGO, 2007.

[218] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM (JACM)*, 47(4):617–643, 2000.

[219] C. Kamienski, M. Jentsch, M. Eisenhauer, J. Kiljander, E. Ferrera, P. Rosengren, J. Thestrup, E. Souto, W. S. Andrade, and D. Sadok. Application development for the internet of things: A context-aware mixed criticality systems development platform. *Computer Communications*, 2016.

[220] O.R. Kelly, H. Aydin, and B. Zhao. On partitioned scheduling of fixed-priority mixed-criticality task sets. In *IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1051–1059, 2011.

[221] H. Kim, D. Broman, E. Lee, M. Zimmer, A. Shrivastava, and J. Oh. A predictable and command-level priority-based DRAM controller for mixed-criticality systems. In *Proc. Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 317–326. IEEE, 2015.

[222] N. Kim, B.C. Ward, M. Chisholm, C-Y. Fu, J.H. Anderson, and F.D. Smith. Attacking the one-out-of-m multicore problem by combining hardware management with mixed-criticality provisioning. In *Proc RTAS)*, pages 1–12. IEEE, 2016.

[223] Y.-S. Kim and H.-W. Jin. Towards a practical implementation of criticality mode change in RTOS. Technical report, Konkuk University, Korea, 2014.

[224] A. Kostrzewa, S. Saidi, and R. Ernst. Dynamic control for mixed-criticality networks-on-chip. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*, pages 317–326, 2015.

[225] O. Kotaba, J. Nowotschy, M. Paulitschy, S.M. Petters, and H. Theiling. Multicore in real-time systems temporal isolation challenges due to shared resources. In *Proceedings of the Conference on Design, Automation and Test in Europe, WICERT*, DATE, 2013.

[226] A. Kritikakou, O. Baldellon, C. Pagetti, C. Rochange, M. Roy, and F. Vargas. Monitoring on-line timing information to support mixed-critical workloads. In *WiP, RTSS*, pages 9–10, 2013.

[227] A. Kritikakou, C. Pagetti, O. Baldellon, M. Roy, and C. Rochange. Run-time control to increase task parallelism in mixed-critical systems. In *ECRTS*, pages 119–128, 2014.

[228] A. Kritikakou, C. Pagetti, C. Rochange, M. Roy, M. Faugre, S. Girbal, and D.G. Prez. Distributed run-time WCET controller for concurrent critical tasks in mixed-critical systems. In *Proc. RTNS*, 2014.

[229] N.G. Kumar, S. Vyas, R.K. Cytron, C.D. Gill, J. Zambreno, and P.H. Jones. Cache design for mixed criticality real-time systems. In *Proc. ICCD*, pages 513–516. IEEE, 2014.

[230] A. Lackorzynski, A. Warg, M. Voelp, and H. Haertig. Flattening hierarchical scheduling. In *Proc. ACM EMSOFT*, pages 93–102, 2012.

[231] K. Lakshmanan, D. de Niz, and R. Rajkumar. Mixed-criticality task synchronization in zero-slack scheduling. In *IEEE RTAS*, pages 47–56, 2011.

[232] K. Lakshmanan, D. de Niz, R. Rajkumar, and G. Moreno. Resource allocation in distributed mixed-criticality cyber-physical systems. In *ICDCS*, pages 169–178, 2010.

[233] A. Larrucea, I. Martinez, V. Brocal, S. Peirò, H. Ahmadian, J. Perez, and R. Obermaisser. DREAMS: Cross-domain mixed-criticality patterns. In *Proc. 4th WMC (RTSS)*, page 6, 2016.

[234] J. Lee, H.S. Chwa, A. Easwaran, I. Shin, and I. Lee. Towards compositional mixed-criticality real-time scheduling in open systems. In L. Almeida and D. de Niz, editors, *Proc. 8th Workshop on Compositional Real-Time Systems (CRTS), RTSS*, 2015.

[235] J. Lee, H.S. Chwa, A. Easwaran, I. Shin, and I. Lee. Towards compositional mixed-criticality real-time scheduling in open systems: invited paper. *ACM SIGBED Review*, 13(3):49–51, 2016.

[236] J. Lee, K-M. P, Z Gu, J. Lee, A. Easwaran, I. Shin, and I. Lee. MC-Fluid: Fluid model-based mixed-criticality scheduling on multiprocessors. In *Proc. IEEE Real-Time Systems Symposium*, pages 41–52. IEEE, 2014.

[237] V. Legout, M. Jan, and L. Pautet. Mixed-criticality multiprocessor real-time systems: Energy consumption vs deadline misses. In L. George and G. Lipari, editors, *Proc. ReTiMiCS, RTCSA*, pages 1–6, 2013.

[238] J.P. Lehoczky and S. Ramos-Thuel. An optimal algorithm for scheduling soft-aperiodic tasks fixed-priority preemptive systems. In *Proc. 13th IEEE Real-Time Systems Symposium*, pages 110–123, 1992.

[239] J.P. Lehoczky, L. Sha, and J.K. Strosnider. Enhanced aperiodic responsiveness in a hard real-time environment. In *Proc. 8th IEEE Real-Time Systems Symposium*, pages 261–270, 1987.

[240] B. Lesage, I. Puaut, and A. Seznec. PRETI: Partitioned real-time shared cache for mixed-criticality real-time systems. In *Proc. 20th RTNS*, pages 171–180, 2012.

[241] J. Y-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation (Netherlands)*, 2(4):237–250, Dec. 1982.

[242] H. Li. *Scheduling Mixed-Criticality Real-Time Systems*. PhD thesis, The University of North Carolina at Chapel Hill, 2013.

[243] H. Li and S.K. Baruah. An algorithm for scheduling certifiable mixed-criticality sporadic task systems. In *Proc. of the Real-Time Systems Symposium*, pages 183–192, San Diego, CA, 2010. IEEE Computer Society Press.

[244] H. Li and S.K. Baruah. Load-based schedulability analysis of certifiable mixed-criticality systems. In *Proc. EMSOFT*, pages 99–107. ACM Press, 2010.

[245] H. Li and S.K. Baruah. Global mixed-criticality scheduling on multiprocessors. In *Proc, ECRTS*, pages 99–107. IEEE Computer Society Press, 2012.

[246] J. Li, D. Ferry, S. Ahuja, K. Agrawal, C. Gill, and C. Lu. Mixed-criticality federated scheduling for parallel real-time tasks. In *Proc. RTAS*, pages 1–12. IEEE, 2016.

[247] Y. Li, R. West, and E. Missimer. The quest-v separation kernel for mixed criticality systems. In *Proc. 1st WMC, RTSS*, pages 31–36, 2013.

[248] Z. Li and L. Wang. Memory-aware scheduling for mixed-criticality systems. In *IProc ICCSA*, pages 140–156. Springer, LNCS 9787, 2016.

[249] J. Lin, A.M.K. Cheng, D. Steel, and M.Y.-C. Wu. Scheduling mixed-criticality real-time tasks with fault tolerance. In L. Cucu-Grosjean and R. Davis, editors, *Proc. 2nd Workshop on Mixed Criticality Systems (WMC), RTSS*, pages 39–44, 2014.

[250] P. Lindgren, D. Pereira, J. Eriksson, M. Lindner, and L. Miguel. Rtfm-lang static semantics for systems with mixed criticality. *Ada User Journal, Proc of Workshop on Mixed Criticality for Industrial Systems (WMCIS'2014)*, 35(2):128–132, 2014.

[251] G. Lipari and S.K. Baruah. Greedy reclaimation of unused bandwidth in constant bandwidth servers. In *IEEE Proc. of the 12th Euromicro Conference on Real-Time Systems*, Stokholm, Sweden, June 2000.

[252] G. Lipari and E. Bini. A methodology for designing hierarchical scheduling systems. *J. Embedded Comput.*, 1(2):257–269, 2005.

[253] G. Lipari and G. Buttazzo. Resource reservation for mixed criticality systems. In *Proceeding of Workshop on Real-Time Systems: the past, the present, and the future*, pages 60–74, York, UK, 2013.

[254] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *JACM*, 20(1):46–61, 1973.

[255] D. Liu, J. Spasic, N. Guan, G. Chen, S. Liu, T. Stefanov, and W. Yi. Edf-vd scheduling of mixed-criticality systems with degraded quality guarantees. In *IEEE RTSS*, pages 35–46, 2016.

[256] G. Liu, Y. Lu, S. Wang, and Z. Gu. Partitioned multiprocessor scheduling of mixed-criticality parallel jobs. In *Proc. Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2014.

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

[258] J. López, J. Díaz, J. Entrialgo, and D. García. Stochastic analysis of real-time systems under preemptive priority-driven scheduling. *Real-Time Systems*, pages 180–207, 2008.

[259] A. Lyons and G. Heiser. Mixed-criticality support in a high-assurance, general-purpose microkernel. In L. Cucu-Grosjean and R. Davis, editors, *Proc. 2nd Workshop on Mixed Criticality Systems (WMC), RTSS*, pages 9–14, 2014.

[260] M. Mahdiani and A. Masrur. Introducing utilization caps into mixed-criticality scheduling. In *Proc. Digital System Design (DSD)*, pages 388–395. IEEE, 2016.

[261] S.O. Marinescu, D. Tamas-Selicean, V. Acretoaie, and P. Pop. Timing analysis of mixed-criticality hard real-time applications implemented on distributed partitioned architectures. In *17th IEEE International Conference on Emerging Technologies and Factory Automation*, 2012.

[262] A. Masrur. A probabilistic scheduling framework for mixed-criticality systems. In *Proc. DAC*, page 132. ACM, 2016.

[263] A. Masrur, D. Muller, and M. Werner. Bi-level deadline scaling for admission control in mixed-criticality systems. In *Proc. 21st IEEE Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 100–109. IEEE, 2015.

[264] S. Maurer and R. Kirner. Cross-criticality interfaces for cyber-physical systems. In *Proc. 1st IEEE Int'l Conference on Event-based Control, Communication, and Signal Processing*, 2015.

[265] D. Maxim, R.I. Davis, L. Cucu-Grosjean, and A. Easwaran. Probabilistic analysis for mixed criticality scheduling with SMC and AMC. In *Proc. 4th WMC (RTSS)*, page 6, 2016.

[266] M. Mendez, J.L.G. Rivas, D.F. Garca-Valdecasas, and J. Diaz. Open platform for mixed-criticality applications. In *Proceedings of the Conference on Design, Automation and Test in Europe, WICERT*, DATE, 2013.

[267] Minstry of Defence. Requirements for safety related software in defence equipment. Defence standard, 00-55, Minstry of Defence, 1997.

[268] E. Missimer, K. Missimer, and R. West. Mixed-criticality scheduling with i/o. In *Proc. ECRTS*, pages 120–130, 2016.

[269] M. Mollison, J. Erickson, J. Anderson, S.K. Baruah, and J. Scoredos. Mixed criticality real-time scheduling for multicore systems. In *Proc. of the 7th IEEE International Conference on Embedded Software and Systems*, pages 1864–1871, 2010.

[270] B. Motruk, J. Diemer, R. Buchty, R. Ernst, and M. Berekovic. Idamc: A many-core platform with run-time monitoring for mixed-criticality. *Ninth IEEE International Symposium on High-Assurance Systems Engineering (HASE'05)*, pages 24–31, 2012.

[271] D. Muller and A. Masrur. The scheduling region of two-level mixed-criticality systems based on EDF-VD. In *Proceedings of the Conference on Design, Automation and Test in Europe*, Proc. DATE, pages 978–981, 2014.

[272] K. Napier, O. Horst, and C. Prehofer. Comparably evaluating communication performance within mixed-criticality systems. In *Proc. 4th WMC (RTSS)*, page 6, 2016.

[273] S. Narayana, P. Huang, G. Giannopoulou, L. Thiele, and R.V. Prasad. Exploring energy saving for mixed-criticality systems on multi-cores. In *Proc. RTAS*, pages 1–12. IEEE, 2016.

[274] M. Neukirchner, P. Axer, T. Michaels, and R. Ernst. Monitoring of workload arrival functions for mixed-criticality systems. In *Proc. IEEE 34th Real-Time Systems Symposium*, pages 88–96, 2013.

[275] M. Neukirchner, S. Quinton, and K. Lampka. Multi-mode monitoring for mixed-criticality real-time systems. In *Int'l Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2013.

[276] M. Neukirchner, S. Stein, H. Schrom, J. Schlatow, and R. Ernst. *Contract-based dynamic task management for mixed-criticality systems*, pages 18–27. IEEE, 2011.

[277] R. Nevalainen, U. Kremer, O. Slotosch, D. Truscan, and V. Wong. Impact of multicore platforms in hardware and software certification. In *Proceedings of the Conference on Design, Automation and Test in Europe, WICERT*, DATE, 2013.

[278] L.M. Ni and P.K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, Feb 1993.

[279] D.de Niz, K. Lakshmanan, and R. Rajkumar. On the scheduling of mixed-criticality real-time task sets. In *Real-Time Systems Symposium*, pages 291–300. IEEE Computer Society, 2009.

[280] D.de Niz and L.T.X. Phan. Partitioned scheduling of multi-modal mixed-criticality real-time systems on multiprocessor platforms. In *Proc. Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 111–122, April 2014.

[281] D.de Niz, L. Wrage, A. Rowe, and R. Rajkumar. Utility-based resource overbooking for cyber-physical systems. In *Proc. RTCSA*, 2013.

[282] A. Novak, P. Sucha, and Z. Hanzalek. Efficient algorithm for jitter minimization in time-triggered periodic mixed-criticality message scheduling problem. In *Proc. RTNS*, pages 23–31. ACM, 2016.

[283] A. Novak, P. Sucha, and Z. Hanzalek. On solving non-preemptive mixed-criticality match-up scheduling problem with two and three criticality levels. *arXiv preprint arXiv:1610.07384*, 2016.

[284] J. Nowotsch, M. Paulitsch, D. Bhler, H. Theiling, S. Wegener, and M. Schmidt. Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement. In *ECRTS*, pages 109–118, 2014.

[285] R. Obermaisser, Z. Owda, M. Abuteir, H. Ahmadian, and D. Weber. End-to-end real-time communication in mixed-criticality systems based on networked multicore chips. In *Proc 17th Euromicro Conference on Digital Systems Design*, pages 293–302. IEEE, 2014.

[286] R. Obermaisser and D. Weber. Architectures for mixed-criticality systems based on networked multi-core chips. In *Proc. ETFA*, pages 1–10, 2014.

[287] S. Osmolovskiy, I. Fedorov, V. Vinogradov, E. Ivanova, and D. Shakurov. Mixed-criticality scheduling in real-time multiprocessor systems. In *Proc. Conference of Open Innovations Association and Seminar on Information Security and Protection of Information Technology (FRUCT-ISPIT)*, pages 257–265, 2016.

[288] E. Papastefanakis, X. Li, and L. George. A mixed criticality approach for the security of critical flows in a network-on-chip. *ACM SIGBED Review*, 13(4):67–72, 2016.

[289] T. Park and S Kim. Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems. In *Proc. ACM EMSOFT*, pages 253–262, 2011.

[290] R.M. Pathan. Schedulability analysis of mixed criticality systems on multiprocessors. In *Proc. of ECRTS, Pisa*, pages 309–320, 2012.

[291] R.M. Pathan. Fault-tolerant and real-time scheduling for mixed-criticality systems. *Journal of Real-Time Systems*, 50(4):509–547, 2014.

[292] M. Paulitsch, O.M. Duarte, H. Karray, K. Mueller, D. Muench, and J. Nowotsch. Mixed-criticality embedded systems–a balance ensuring partitioning and performance. In *Euromicro Conference on Digital System Design (DSD)*, pages 453–461. IEEE, 2015.

[293] P. Pedro and A. Burns. Schedulability analysis for mode changes in flexible real-time systems. In *10th Euromicro Workshop on Real-Time Systems*, pages 172–179. IEEE Computer Society, 1998.

[294] R. Pellizzoni, P. Meredith, M-Y. Nam, M. Sun, M. Caccamo, and L. Sha. Handling mixed-criticality in soc-based real-time embedded systems. In *Proc. of the 7th ACM international conference on Embedded software, EMSOFT*, pages 235–244. ACM Press, 2009.

[295] R. Pellizzoni, A. Schranzhofery, J. Cheny, M. Caccamo, and L. Thiele. Worst case delay analysis for memory interference in multicore systems. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 741–746, 2010.

[296] H. Pérez, J.J. Gutiérrez, S. Peiró, and A. Crespo. Distributed architecture for developing mixed-criticality systems in multi-core platforms. *Journal of Systems and Software*, 123:145–159, 2017.

[297] J. Perez, D. Gonzalez, S. Trujillo, T. Trapman, and J. M. Garate. A safety concept for a wind power mixed criticality embedded system based on multicore partitioning. In *Proc. 1st WMC, RTSS*, pages 25–30, 2013.

[298] P. Petrakis, M. Abuteir, M.D. Grammatikakis, K. Papadimitriou, R. Obermaisser, Z. Owda, A. Papagrigoriou, M. Soulie, and M. Coppola. On-chip networks for mixed-criticality systems. In *Proc. Application-specific Systems, Architectures and Processors (ASAP*, pages 164–169. IEEE, 2016.

[299] P. Pop, L. Tsiopoulos, S. Voss, O. Slotosch, C. Ficek, U. Nyman, and A. Ruiz. Methods and tools for reducing certification costs of mixed-criticality applications on multi-core platforms: the RECOMP approach. In *Proceedings of the Conference on Design, Automation and Test in Europe, WICERT*, DATE, 2013.

[300] T. Pop, P. Eles, and Z. Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *Proceedings of the tenth international symposium on Hardware/software codesign*, CODES '02, pages 187–192. ACM, 2002.

[301] S. Punnekkat, R.I Davis, and A. Burns. Sensitivity analysis of real-time task sets. In *Proc. of the Conference of Advances in Computing Science - ASIAN '97*, pages 72–82. Springer, 1997.

[302] S. Ramanathan and A. Easwaran. MC-Fluid: rate assignment strategies. In L. Cucu-Grosjean and R. Davis, editors, *Proc. 3rd Workshop on Mixed Criticality Systems (WMC), RTSS*, pages 6–11, 2015.

[303] S. Ramanathan, X. Gu, and A. Easwaran. The feasibility analysis of mixed-criticality systems. In *Proc. RTOPS, ECRTS*, 2016.

[304] S. Ramos-Thuel and J.P. Lehoczky. Algorithms for scheduling hard aperiodic tasks in fixed priority systems using slack stealing. In *Proc. 15th IEEE Real-Time Systems Symposium*, pages 22–35, 1994.

[305] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new protocol. *Journal of Real-Time Systems*, 26(2):161–197, 2004.

[306] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt. RUN: Optimal multiprocessor real-time scheduling via reduction to uniprocessor. In *Real-Time Systems Symposium (RTSS)*, pages 104–115. IEEE, 2011.

[307] J. Ren and L.T.X. Phan. Mixed-criticality scheduling on multiprocessors using task grouping. In *Proc. 27th ECRTS*, pages 25–36. IEEE, 2015.

[308] P. Rodriguez, L. George, Y. Abdeddaim, and J. Goossens. Multi-criteria evaluation of partitioned EDF-VD for mixed criticality systems upon identical processors. In *Proc. 1st WMC, RTSS*, pages 49–54, 2013.

[309] RTCA-EUROCAE. *Software Considerations in Airborne Systems and Equipment Certification DO-178B/ED-12B*. RTCA, Inc, December 1992.

[310] S. Saewong, R. Rajkumar, J.P. Lehoczky, and M.H. Klein. Analysis of hierarchical fixed- priority scheduling. In *Proc. of the 14th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 173–181, 2002.

[311] S. Saidi, R. Ernst, S. Uhrig, H. Theiling, and B.D. de Dinechin. The shift to multicores in real-time and safety-critical systems. In *Proc. 10th International Conference on Hardware/Software Codesign and System Synthesis*, pages 220–229. IEEE Press, 2015.

[312] M. Saksena and Y. Wang. Scaleable real-time systems design using preemption thresholds. In *Proceeding 21st IEEE Real-Time Systems Symposium.*, pages 25–34, 2000.

[313] E. Salazar, A. Alejandro, and J. Garrido. Mixed-criticality design of a satellite software system. In *Proc. 19th World Congress The Federation of Automatic Control*, pages 12278–12283, 2014.

[314] L. Santinelli and L. George. Probabilities and mixed-criticalities: the probabilistic C-Space. In L. Cucu-Grosjean and R. Davis, editors, *Proc. 3rd Workshop on Mixed Criticality Systems (WMC), RTSS*, pages 30–35, 2015.

[315] R. Santos, S. Venkataraman, A. Das, and A. Kumar. Criticality-aware scrubbing mechanism for SRAM-based FPGAs. Technical report, Nanyang Technological University, Singapore, 2014.

[316] J. A. Santos-Jr, G. Lima, and K. Bletsas. Considerations on the least upper bound for mixed-criticality real-time systems. In *5th Brazilian Symposium on Computing Systems Engineering (SBESC)*, 2015.

[317] F. Santy, L. George, P. Thierry, and J. Goossens. Relaxing mixed-criticality scheduling strictness for task sets scheduled with FP. In *Proc. of the Euromicro Conference on Real-Time Systems*, pages 155–165, 2012.

[318] 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.

[319] R. Schneider, D. Goswami, A. Masrur, M. Becker, and S. Chakraborty. Multi-layered scheduling of mixed-criticality cyber-physical systems. *Journal of Systems Architecture*, 59(10, Part D):1215 – 1230, 2013.

[320] V. Sciandra, P. Courbin, and L George. Application of mixed criticality scheduling model to intelligent transportation systems architecture. In *Proc. WiP, RTSS*, page 11, 2012.

[321] L. Sha. Resilient mixed criticality systems. *CrossTalk The Journal of Defense Software Engineering*, pages 9–14, October 2009.

[322] L. Sha, J.P. Lehoczky, and R. Rajkumar. Solutions for some practical problems in prioritizing preemptive scheduling. In *Proc. 7th IEEE Real-Time Sytems Symposium*, 1986.

[323] L. Sha, J.P. Lehoczky, and R. Rajkumar. Task scheduling in distributed real-time systems. In *Proc. of IEEE Industrial Electronics Conference*, 1987.

[324] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode change protocols for priority-driven premptive scheduling. *Journal of Real-Time Systems*, 1(3):244–264, 1989.

[325] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronisation. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

[326] Z. Shi and A. Burns. Real-time communication analysis for on-chip networks with wormhole switching. In *Proc. of the 2nd ACM/IEEE International Symposium on Networks-on-Chip(NoCS)*, pages 161–170, 2008.

[327] L. Sigrist, G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele. Mapping mixed-criticality applications on multi-core architectures. In *Proc. DATE*, pages 1–6, 2014.

[328] D. Socci, P. Poplavko, S. Bensalem, and M. Bozga. Mixed critical earliest deadline first. Technical Report TR-2012-22, Verimag Research Report, 2012.

[329] D. Socci, P. Poplavko, S. Bensalem, and M. Bozga. Mixed critical earliest deadline first. In *Proc. Euromicro Conference on Real-Time Systems (ECRTS)*, 2013.

[330] D. Socci, P. Poplavko, S. Bensalem, and M. Bozga. Modeling mixed-critical systems in real-time bip. In L. George and G. Lipari, editors, *Proc. ReTiMiCS, RTCSA*, pages 29–34, 2013.

[331] D. Socci, P. Poplavko, S. Bensalem, and M. Bozga. Time-triggered mixed critical scheduler. In *Proc. WMC, RTSS*, pages 67–72, 2013.

[332] D. Socci, P. Poplavko, S. Bensalem, and M. Bozga. Multiprocessor scheduling of precedence-constrained mixed-critical jobs. Technical Report TR-2014-11, Verimag, Research Report, 2014.

[333] D. Socci, P. Poplavko, S. Bensalem, and M. Bozga. Time-triggered mixed-critical scheduler on single- and multi-processor platforms. Technical Report TR-2015-8, Verimag, 2015.

[334] B. Sprunt, J. Lehoczky, and L. Sha. Exploiting unused periodic time for aperiodic service using the extended priority exchange algorithm. In *Proc. 9th IEEE Real-Time Systems Symposium*, pages 251–258, 1988.

[335] W. Steiner. Synthesis of static communication schedules for mixed-criticality systems. *2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, pages 11–18, 2011.

[336] H. Su, P. Deng, D. Zhu, and Q. Zhu. Fixed-priority dual-rate mixed-criticality systems: Schedulability analysis and performance optimization. In *Proc. Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 59–68. IEEE, 2016.

[337] H. Su, P. Deng, D. Zhu, and Q. Zhu. Fixed-priority dual-rate mixed-criticality systems: Schedulability analysis and performance optimization. In *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 59–68, 2016.

[338] H. Su, N. Guan, and D. Zhu. Service guarantee exploration for mixed-criticality systems. In *Proc. Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–10. IEEE, 2014.

[339] 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.

[340] H. Su, D. Zhu, and D. Mosse. Scheduling algorithms for elastic mixed-criticality tasks in multicore systems. In *Proc. RTCSA*, 2013.

[341] D. Tamas-Selicean and P. Pop. Design optimisation of mixed criticality real-time applications on cost-constrained partitioned architectures. In *Real-Time Systems Symposium (RTSS)*, pages 24–33, 2011.

[342] D. Tamas-Selicean and P. Pop. Optimization of time-partitions for mixed criticality real-time distributed embedded systems. In *14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, pages 2–10, 2011.

[343] D. Tamas-Selicean and P. Pop. Task mapping and partition allocation for mixed criticality real-time systems. In *IEEE Pacific Rim Int. Sym. on Dependable Computing*, pages 282–283, 2011.

[344] D. Tamas-Selicean and P. Pop. Design optimisation of mixed criticality real-time applications on cost-constrained partitioned architectures. *ACM Transactions on Embedded Systems*, 14(3):50:1–50:29, 2015.

[345] J. Theis and G. Fohler. Mixed criticality scheduling in time-triggered legacy systems. In *Proc. WMC, RTSS*, pages 73–78, 2013.

[346] J. Theis, G. Fohler, and S.K. Baruah. Schedule table generation of time-triggered mixed criticality systems. In *Proc. WMC, RTSS*, pages 79–84, 2013.

[347] A. Thekkilakattil, R. Dobrin, and S. Punnekkat. Fault tolerant scheduling of mixed criticality real-time tasks under error bursts. In *The International Conference on Information and Communication Technologies ICICT'14*. Elsevier Procedia Computer Science, 2014.

[348] A. Thekkilakattil, R. Dobrin, and S. Punnekkat. Mixed criticality scheduling in fault-tolerant distributed real-time systems. In *Embedded Systems (ICES), 2014 International Conference on*, pages 92–97. IEEE, 2014.

[349] A Thekkilakattl, A. Burns, R. Dobrin, and S. Punnekkat. Mixed criticality systems: Beyond transient faults. In L. Cucu-Grosjean and R. Davis, editors, *Proc. 3rd Workshop on Mixed Criticality Systems (WMC), RTSS*, pages 18–23, 2015.

[350] H. Thompson. Mixed criticality systems. http://cordis.europa.eu/fp7/ict/embedded-systems-engineering/documents/sra-mixed-criticality-systems.pdf, EU, ICT, February 2012.

[351] K. Tindell and A Alonso. A very simple protocol for mode changes in priority preemptive systems. Technical report, Universidad Politecnica de Madrid, 1996.

[352] K. Tindell, A. Burns, and A. J. Wellings. Mode changes in priority preemptive scheduled systems. In *Proc. Real Time Systems Symposium*, pages 100–109, Phoenix, Arizona, 1992.

[353] S. Tobuschat, P. Axer, R. Ernst, and J. Diemer. IDAMC: A NoC for mixed criticality systems. In *Proc. RTCSA*, 2013.

[354] S. Trujillo, A. Crespo, and A. Alonso. MultiPARTES: Multicore virtualization for mixed-criticality systems. In *Digital System Design (DSD), 2013 Euromicro Conference on*, pages 260–265, 2013.

[355] S. Trujillo, A. Crespo, A. Alonso, and J. Perez. MultiPARTES: Multi-core partitioning and virtualization for easing the certification of mixed-criticality systems. *Microprocessors and Microsystems (online version)*, 2014.

[356] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, pages 239–243, 2007.

[357] M. Völp, M. Hähnel, and A. Lackorzynski. Has energy surpassed timeliness? scheduling energy-constrained mixed-criticality systems. In *Proc. RTAS*, pages 275–284. IEEE, 2014.

[358] M. Völp, A. Lackorzynski, and H. Härtig. On the expressiveness of fixed priority scheduling contexts for mixed criticality scheduling. In *Proc. WMC, RTSS*, pages 13–18, 2013.

[359] M. Völp, M. Roitzsch, and H. Härtig. Towards an interpretation of mixed criticality for optimistic scheduling. In *21st IEEE RTAS, Work-in-Progress*, pages 15–16, 2015.

[360] G. von der Brüggen, K-H. Chen, W-H. Huang, and J-J. Chen. Systems with dynamic real-time guarantees in uncertain and faulty execution environments. In *Proc. Real-Time Systems Symposium (RTSS)*, pages 303–314. IEEE, 2016.

[361] P. Wagemann, T. Distler, H. Janker, P. Raffeck, and V. Sieh. A kernel for energy-neutral real-time systems with mixed criticalities. In *Proc. RTAS*, pages 1–12. IEEE, 2016.

[362] Y. Wang and M. Saksena. Scheduling fixed-priority tasks with preemption threshold. In *6th Real-Time Computing Systems and Applications (RTCSA)*, pages 328–335. IEEE, 1999.

[363] A. Wasicek, C. El-Salloum, and H. Kopetz. A system-on-a-chip platform for mixed-criticality applications. In *3th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, pages 210–216, 2010.

[364] R. West, Y. Li, E. Missimer, and M. Danish. A virtualized separation kernel for mixed-criticality systems. *ACM Transactions on Computer Systems (TOCS)*, 34(3):8, 2016.

[365] H. Xu and A. Burns. Semi-partitioned model for dual-core mixed criticality system. In *23rd International Conference on Real-Time Networks and Systems (RTNS 2015)*, pages 257–266, 2015.

[366] C. Yao, L. Qiao, L. Zheng, and X. Huagang. Efficient schedulability analysis for mixed-criticality systems under deadline-based scheduling. *Chinese Journal of Aeronautics*, 2014.

[367] E. Yip, M.M.Y Kuo, D. Broman, and P.S Roop. Relaxing the synchronous approach for mixed-criticality systems. In *Proc. Real-Time and Embedded Technology and Application Symposium (RTAS)*, pages 89–100. IEEE, 2014.

[368] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. Memory access control in multiproccessor for real-time mixed criticality. In *Proc. of ECRTS, Pisa*, pages 299–308, 2012.

[369] A. Zabos, R.I. Davis, A. Burns, and M. González Harbour. Spare capacity distribution using exact response-time analysis. In *17th International Conference on Real-Time and Network Systems*, pages 97–106, 2009.

[370] L. Zeng, P. Huang, and L. Thiele. Towards the design of fault-tolerant mixed-criticality systems on multicores. In *Proc. Compilers, Architectures and Synthesis for Embedded Systems*, page 6. ACM, 2016.

[371] F. Zhang and A. Burns. Analysis of hierarchical EDF preemptive scheduling. In *Proc. of IEEE Real-Time Systems Symposium (RTSS)*, pages 423–435, 2007.

[372] F. Zhang and A. Burns. Schedulability analysis for real-time systems with EDF scheduling. *IEEE Transaction on Computers*, 58(9):1250–1258, 2008.

[373] N. Zhang, C. Xu, J. Li, and M. Peng. A sufficient response-time analysis for mixed criticality systems with pessimistic period. *Journal of Computational Information Systems*, 11(6):1955–1964, 2015.

[374] X. Zhang, J. Zhan, W. Jiang, Y. Ma, and K. Jiang. Design optimization of security-sensitive mixed-criticality real-time embedded systems. In L. George and G. Lipari, editors, *Proc. ReTiMiCS, RTCSA*, pages 12–17, 2013.

[375] Q. Zhao, Z. Al-Bayati, Z. Gu, and H. Zeng. Optimized implementation of multirate mixed-criticality synchronous reactive models. *ACM Trans. Des. Autom. Electron. Syst.*, 22(2):23:1–23:25, 2016.

[376] Q. Zhao, Z. Gu, and H. Zeng. Integration of resource synchronization and preemption-thresholds into EDF-based mixed-criticality scheduling algorithm. In *Proc. RTCSA*, 2013.

[377] Q. Zhao, Z. Gu, and H. Zeng. PT-AMC: Integrating preemption thresholds into mixed-criticality scheduling. In *Proc. DATE*, pages 141–146, 2013.

[378] Q. Zhao, Z. Gu, and H. Zeng. HLC-PCP: A resource synchronization protocol for certifiable mixed criticality scheduling. *Embedded Systems Letters, IEEE*, 6(1), 2014.

[379] Q. Zhao, Z. Gu, and H. Zeng. Resource synchronization and preemption thresholds within mixed-criticality scheduling. *ACM Transactions on Embedded Computing Systems (TECS)*, 14(4):81, 2015.

[380] L. Ziarek and E. Blanton. The Fiji MultiVM archiecture. In *Proc. JTRES*. ACM, 2015.

[381] M. Zimmer, D.Broman, C. Shaver, and E.A. Lee. FlexPRET: A processor platform for mixed-criticality systems. In *Proc. RTAS*, pages 101–110, 2014.