A Survey of Research into Mixed Criticality Systems

ALAN BURNS, University of York
ROBERT I. DAVIS, University of York

This survey covers research into mixed criticality systems that has been published since Vestal's seminal paper in 2007, up until the end of 2016. The survey is organised along the lines of the major research areas within this topic. These include single processor analysis (including fixed priority and EDF scheduling, shared resources and static and synchronous scheduling), multiprocessor analysis, realistic models, and systems issues. The survey also explores the relationship between research into mixed criticality systems and other topics such as hard and soft time constraints, fault tolerant scheduling, hierarchical scheduling, cyber physical systems, probabilistic real-time systems, and industrial safety standards.

CCS Concepts: •**Computer systems organization** → *Real-time system specification;* •**Software and its engineering** → *Real-time schedulability;* Real-time systems software;

Additional Key Words and Phrases: Real-Time Systems, Mixed Criticality Systems, Scheduling

ACM Reference Format:
Alan Burns and Robert I. Davis, 2017, A Survey of Research into Mixed Criticality Systems. *ACM Trans. Embedd. Comput. Syst.* 50, 6, Article 82 (December 2017), 35 pages.
DOI: 0000001.0000001

1. INTRODUCTION

An increasingly important trend in the design of real-time embedded systems is the integration of components with different levels of criticality onto a common computing platform. At the same time, these platforms are migrating from single-core to multi-core hardware, and in the future to many-core architectures. Criticality is a designation of the level of assurance needed against failure for a system component. Mixed Criticality Systems (MCS) are systems that have components of two or more distinct criticality levels, for example safety-critical, mission-critical and non-critical.

Most of the complex embedded systems found in the automotive and avionics sectors are evolving into MCS 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.

The fundamental research question underlying MCS 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 hardware and run-time software.

A key aspect of MCS is that parameters, such as the worst-case execution time (WCET) estimates for tasks, become dependent on the criticality level of the components they belong to. Thus the same code may have a higher WCET estimate if it is defined to be safety critical rather than mission critical, as a higher level of assurance is required. This property of MCS significantly undermines or modifies many of the standard scheduling results.

A commonly used exemplar of a MCS is an unmanned aerial vehicle. In order to operate in civilian airspace, the safety-critical flight control software must be certified by a civil aviation authority. By contrast, mission-critical software associated with capturing and processing images must be 'fit for purpose' and can be signed off by the system's designers. Other software, such as a

The research that went into writing this survey was funded, in part, by the ESPRC grants, MCC (EP/K011626/1) and MCCps (EP/P003664/1). EPSRC Research Data Management: No new primary data was created during this study.
Author's addresses: A. Burns and R.I. Davis, Department of Computer Science, University of York.

route planner, is desirable and improves the Quality of Service (QoS) of the system, but is less critical. The criticality of a component determines the level of rigour applied in the design and analysis used to determine its correct functionality and resource usage (e.g. processor execution time, communication bandwidth etc.). As a result, the same component can have more than one resource-usage profile. If it is used as part of the flight control subsystem then more conservative assumptions as to its potential resource usage need to be made; if it is part of a mission-critical activity then lower, though still realistic, resource requirements can be assumed. Put simply, all safety-critical software must meet its timing requirements when conservative assumptions are made; and all mission-critical components must meet their requirements when more realistic assumptions are made for *all* of the software. This crucial property, of being able to take a criticality specific view of resource usage, makes the verification task more complex, but opens up the possibility of much more efficient resource usage.

A simplistic contrived example illustrates this tradeoff. Consider a system with just two components, A and B. Component A is safety critical and needs a 2-core platform to guarantee its temporal behaviour, when analysed using the conservative techniques prescribed for this level of criticality. Component B is mission critical and needs a 1-core platform. It would therefore seem that a 3-core platform is required. However, if A is also analysed as if it were mission critical, i.e. using the same techniques applied to B, then it may only require a single core. Hence a 2-core system is sufficient for the mission; a saving of one whole core and its associated cost, heat and power consumption. In the unlikely event that A needs more than one core then B will be abandoned and A will have both cores, thus satisfying the safety case for A. By contrast in normal operation, both A and B will run adequately on the 2-core platform.

This very simple example hides most of the important details of the resource usage model and the necessary verification. Nevertheless, it highlights the advantages that can accrue from the verification techniques currently being developed for MCSs. It also points to the need for run-time monitoring and protection mechanisms that will protect A from B and that can abandon B if either A or B operates outside the assumptions encapsulated in their mission-critical profiles.

The first paper on the verification of MCS was published by Vestal [2007][1]. It employed a somewhat restrictive work-flow model, focused on a single processor and made use of Response Time Analysis [Joseph and Pandya 1986; Audsley et al. 1993] for fixed priority (FP) scheduling. Vestal [2007] showed that neither rate monotonic [Liu and Layland 1973] nor deadline monotonic [Leung and Whitehead 1982] priority assignment was optimal for MCS; however the optimal priority assignment algorithm of Audsley [2001] was found to be applicable. This paper was followed by publications by Baruah and Vestal [2008] and Huber et al. [2008]. The first of these papers generalises Vestal's model. It contains the important result that Earliest Deadline First (EDF) scheduling does not dominate fixed priority scheduling when there is more than one criticality level, and that there are feasible systems that cannot be scheduled by EDF. This is in direct contrast to the case with just one criticality level [Dertouzos 1974]. The second paper addresses multi-processor issues and virtualisation. It focuses on resource management via encapsulation and monitoring, assumes 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. [2009], the keynote talk that Baruah gave at the 2010 ECRTS conference[2], and a workshop report from the European Commission [Thompson 2012]. Since then, the research topic has led to a wealth of publications (reviewed in this survey), as well as the establishment of a peer-

[1]The term Mixed Criticality had been used before 2007 to address issues of non-interference in non-federated architectures such as IMA [Hill and Lake 2000]; Vestal changed the focus of research by concentrating on real-time performance. Systems with more than one criticality level that only aim to give complete isolation are called *multiple-criticality systems*; the use of *mixed-criticality* implies some tradeoff between isolation and integration that involves resource sharing.
[2]Available from the conference web site: http://ecrts.eit.uni-kl.de/index.php?id=53.

reviewed Workshop on Mixed Criticality Systems (WMC)[3] at the RTSS conference, and a series of Dagstuhl Seminars on MCS[4].

This survey covers the considerable body of research into MCS stemming from the model presented by Vestal [2007]. Industry practice and safety standards provide a somewhat different perspective on MCS; these differences are discussed in Section 7.6.

The survey is organised along the lines of the major research topics within the area of MCS. In Section 2 we consider mixed criticality models. Section 3 covers research into single processor systems, while Section 4 reviews research targeting multiprocessor systems. More realistic models are covered in Section 5, with systems issues covered in Section 6. Section 7 links research on MCS to other research topics and industry practice. The survey concludes with Section 8 which outlines a number of open problems and areas where further research is needed.

2. MIXED CRITICALITY MODELS

Inevitably not all papers on mixed criticality have used the same system or task model. Below we describe a model that is generally applicable and relates to the main results considered in this survey.

A system is defined as a finite set of components $\mathcal{K}$. Each component has a criticality level L designated by the system designer, and contains a finite set of *sporadic tasks*. Each task τ_i is defined by its period (or minimum inter-arrival time), deadline, worst-case execution time (WCET) and criticality level: (T_i, D_i, C_i, L_i). Each task gives 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, appropriate mechanisms must be used to prevent jobs of a task τ_i from executing for more than the WCET estimate C_i, and to ensure that the task does not generate jobs that are closer together than its minimum inter-arrival time T_i.

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

After concerns about partitioning comes the need to use resources efficiently. This is facilitated by noting that the task parameters are criticality dependent, in particular the WCET estimate, C_i, is derived by a process dictated by the criticality level. The higher the criticality level the more conservative the verification process, and hence the larger the value of C_i. This was the observation at the heart of the seminal work by Vestal [2007].

For systems executing on hardware platforms with deterministic behaviour, any particular task will have a single actual WCET; however, this value typically cannot be determined with complete certainty. This uncertainty is primarily epistemic, i.e. uncertainty in what we know or do not know about the system, rather than aleatory, i.e. uncertainty in the system itself. Although it is reasonable to assume confidence increases and uncertainty decreases with larger estimates of WCET, this may not be universally true [Graydon and Bate 2013]. It would certainly be hard to estimate what increase in confidence would result from say a 10% increase in all WCET estimates.

The focus on different WCET estimates was extended to task periods in a series of subsequent papers [Burns and Baruah 2011; Baruah 2012a; Baruah and Burns 2011; Baruah 2013b; Baruah and Chattopadhyay 2013; Burns and Davis 2013; Zhang et al. 2015; Baruah 2016b]. Here, tasks are viewed as 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 typically 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 that:

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

[3]https://gsathish.github.io/wmc2016/
[4]http://www.dagstuhl.de/17131

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

for any two criticality levels $L1$ and $L2$, where $L1$ is the higher level.

Note the completion of the model, making deadlines criticality dependent [Baruah and Burns 2011], has not yet been addressed in detail; however, it would have the constraint that:

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

Thus a task may have a safety-critical deadline and an earlier Quality of Service (QoS) one.

The three inequalities given above each have the property that a task can progress from satisfying its $L2$ constraints, and hence also its $L1$ constraints, to satisfying only its $L1$ constraints.

Another feature of many of the papers considered in this survey is that the system is considered to execute in a number of criticality *modes*. A system starts in the lowest criticality mode. If all jobs behave according their low-criticality constraints then the system stays in that mode; however, if any job attempts to execute for a longer time, or more frequently, than is acceptable in low-criticality 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, while others restrict the model to increases in criticality mode only. We return to this issue in Section 5.

Finally, many papers restrict themselves to just two criticality levels; high (HI) and low (LO) with $HI > LO$. MCS with just two criticality levels are referred to as *dual-criticality* systems. Where modes are used, the system is either in a low-criticality mode or a high-criticality mode. The set of task parameters is typically given by: (T_i, D_i, $C_i(HI)$, $C_i(LO)$, L_i). At the other extreme are the models in which any number of modes are allowed and the movement between modes is represented by a Directed Acyclic Graph (DAG) [Ekberg and Yi 2014; Ekberg et al. 2013; Ekberg and Yi 2015b][5].

3. SINGLE PROCESSOR ANALYSIS

In this section, we look at research into single processor MCS. This includes work on scheduling using fixed priority and EDF, shared resources, and finally static and synchronous approaches.

A number of papers have considered the restricted problem of scheduling a finite set of mixed criticality jobs with criticality-dependent execution times on a single processor [Baruah et al. 2010b; Li and Baruah 2010a; Baruah et al. 2012; Park and Kim 2011; Baruah et al. 2010; Li and Baruah 2010b; Socci et al. 2013a; Gu et al. 2013; Socci et al. 2015b; Baruah et al. 2016].

The mixed criticality schedulability problem (preemptive or non-preemptive) is strongly NP-hard [Baruah et al. 2010, 2012; Baruah 2012b; Hanzálek et al. 2016] even when there are only two criticality levels. Hence only sufficient rather than exact analysis is possible in practice. A list of open problems on the complexity of scheduling MCS is given by Ekberg and Yi [2015a].

For approaches and tests that are only sufficient, an assessment of their quality is possible if a *speedup factor* [Kalyanasundaram and Pruhs 2000] can be computed. A speedup factor of $X (X \geq 1)$ for schedulability test S implies that a task set that is feasible on a processor of speed 1 will be deemed schedulable by test S if the processor's speed is increased to X.

For fixed priority scheduling of mixed criticality jobs, a priority assignment scheme and test has been found by Baruah et al. [2010a, 2011, 2012] and Li and Baruah [2010b] 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, $\phi \approx 1.618$. This can be compared with a priority partitioning approach, where all high-criticality jobs have priorities higher than all low-criticality jobs, which has no finite speedup factor. For EDF scheduled systems, Baruah et al. [2010a] also showed that a finite set of independent jobs scheduled on m identical multiprocessors is schedulable with a speed-up factor of $\phi + 1 + 1/m$.

[5] There are many other works on scheduling DAGs; however, we restrict the scope here to those specifically addressing MCS.

Considering the sporadic task model, for single criticality systems there are well known bounds on task set utilisation under both fixed priority and EDF scheduling [Liu and Layland 1973]. In the mixed criticality case, although the definition of utilisation is not straightforward when tasks can have more than one worst-case execution time, it is possible to give an effective definition and to derive a least upper bound ($LUB(k)$) in terms of the utilisation at some criticality level k. Santos-Jr et al. [2015] derive a number of such useful results. They construct a task set that is unschedulable during a criticality mode change and has a *LUB* arbitrarily close to 0. They also show that when tasks have harmonic periods the *LUB* can reach 1 for a single processor system. Between these two extremes, if the higher criticality tasks do not have periods that are longer than those of lower criticality tasks then the *LUB* lies is in the range $ln(2)$ to $2(\sqrt{2}-1)$.

3.1. Fixed Priority Scheduling

In this section we review fixed priority scheduling for MCS, including research based on applying Response-Time Analysis (RTA), slack stealing, and period transformation.

3.1.1. RTA-Based approaches. The approach of Vestal [2007] was formalised by Dorin et al. [2010] who proved that the priority assignment algorithm of Audsley [2001] is optimal in that case. Dorin et al. [2010] also extended the model to include release jitter, and showed how sensitivity analysis could be applied.

Vestal's approach allows the priorities of high- and low-criticality tasks to be interleaved; however, all tasks still have to be evaluated as if they were of the highest criticality. Baruah and Burns [2011] introduced a scheme called Static Mixed Criticality (SMC) which monitors task execution times and prevents execution time budget over-runs, permitting higher resource usage. This is a crucial issue in mixed criticality scheduling; the introduction of more trusted components facilitates higher utilisation of the available resources.

Burns and Baruah [2011] and Baruah et al. [2011b] further extended this approach. The system's run-time behaviour is either *low-criticality* (which relies on all execution times being bounded by the low-criticality values and guarantees that all deadlines are met) or *high-criticality* (where only high-criticality tasks are guaranteed but the rely condition[6] is weakened, i.e. the bound on high-criticality execution times is increased). Change in the system's criticality mode, from low to high, is triggered by a run-time monitor detecting that the stronger rely condition has been violated. This change in criticality mode has a number of similarities to systems that move between different operational modes, although there are also some significant differences [Graydon and Bate 2013; Burns 2014]. In the high-criticality mode there are fewer tasks, but they have longer execution times or shorter periods. The literature on mode change protocols highlights one important problem: a system can be schedulable in every mode, but not schedulable during a mode change [Tindell et al. 1992]. This is also true of systems that change criticality mode.

Baruah et al. [2011b] showed that the Adaptive Mixed Criticality (AMC) protocol (dropping all low-criticality jobs if any task executes for more than its $C(LO)$ WCET estimate[7]), out-performs other fixed priority schemes. The analysis for AMC is based on Response-Time Analysis (RTA) [Joseph and Pandya 1986; Audsley et al. 1993]. For any task, τ_i, first its low-criticality response-time ($R(LO)$) is computed using low-criticality parameters for all the tasks. A criticality mode change must occur before this time if the task is to be impacted by the change, otherwise it will have completed execution. The worst-case response time in the high criticality mode ($R(HI)$) is computed by noting that all low-criticality tasks must have been abandoned by time $R(LO)$. The paper contains two methods for computing $R(HI)$, AMC-rtb involves a single upper bound, and AMC-max looks at all the possible criticality mode 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 AMC-rtb test is effective in most cases. An optimal priority assignment algorithm is

[6] A rely condition formalises the assumptions required for the guarantees to be valid [Jones 1983].
[7] First proposed by Baruah et al. [2010].

defined for AMC [Baruah et al. 2011b] which maximises the priority of high criticality tasks, subject to the system being schedulable. This assignment algorithm derives from work on robust priority assignment [Davis and Burns 2007].

The AMC approach was extended by Zhao et al. [2013a,b] to incorporate preemption thresholds [Saksena and Wang 2000] into the model. They demonstrated a reduction in stack usage and improved performance for some parameter ranges. Another approach to combining AMC and existing scheduling theory was taken by Burns and Davis [2014]. They considered the use of deferred preemption [Burns 1994] demonstrating a significant improvement over fully preemptive AMC. Here, the gain in schedulability is obtained by having a final non-preemptive region (FNPR) at the end of $C(LO)$ and $C(HI)$ execution, and by combining the assignment of priority and the determination of the size of this FNPR [Davis and Bertogna 2012].

In keeping with a number of papers on MCS, the above work on AMC was restricted to dual-criticality systems. Fleming and Burns [2013] extended these models to an arbitrary number of criticality levels, focusing particularly on five criticality levels as this is the maximum found in automotive and avionics standards. They observed that AMC-rtb remains a good approximation to AMC-max, while AMC-max became computational expensive with an increased number of criticality 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. [2014a] (referred to as AMC-IA); however there may be cases where their analysis is unsound[8].

One characteristic of all the AMC schemes discussed above is that tasks do not change their priority after a criticality mode change. Baruah et al. [2013] considered the case where the Priority May Change (PMC) and provided a simple form of sufficient analysis for this approach. Evaluations show that the analysis for PMC performs similarly to AMC-rtb, though neither dominates the other. An improved scheme, GFP (Generalised Fixed Priority) was proposed by Chen et al. [2016]. They assign three priorities to each task using a heuristic; one for each of the two criticality levels, and one for the transition between the criticality modes. They demonstrate an improvement over AMC-rtb.

Similar to the WCET estimate (C), the period parameter (T) can also vary with criticality level. An application may consist of event handlers and have different levels of constraint on the arrival patterns of the associated events. The higher the criticality level, the closer together the events are assumed to arrive, and hence the smaller the minimum inter-arrival time (T). Baruah and Chattopadhyay [2013] reformulated the SMC and AMC analysis for this model. Criticality specific periods are also addressed by Burns and Davis [2013] and Baruah [2016b], and also by Zhang et al. [2015] who derived an improved analysis referred to as SAMC (Sufficient AMC).

3.1.2. Slack scheduling. An alternative approach to scheduling dual-criticality systems using fixed priorities is to use a *slack scheduling* scheme. Niz et al. [2009] first explore this scheme where low-criticality jobs run in the slack generated by high-criticality jobs when the latter only consume their low criticality execution time budgets. 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. [2009] compute the time at which a high-criticality task must be released to ensure that it will meet its deadline (a concept similar to that used in dual-priority scheduling [Davis and Wellings 1995]). However, Huang et al. [2012] demonstrated that if a low-criticality high priority task executes beyond its deadline, then a high-criticality low 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 a sound analysis. Niz et al. [2013] and Niz and Phan [2014] subsequently modified the enforcement rule in their model to remove the problem and improved its performance.

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

While slack is usually generated by tasks not executing for their full execution time budget, it is also produced by jobs arriving less frequently than anticipated in the worst-case. [Neukirchner et al. 2013b,a] adapt and extend a number of schemes for monitoring such activation patterns. Hu et al. [2016b] 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 be known; however, once schedulability has been established, Santy et al. [2012] show that it is possible to derive, using sensitivity analysis [Punnekkat et al. 1997; Bini et al. 2006], 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 increases the robustness of the system, since the low-criticality tasks will be able to execute for a greater time before a criticality mode change is triggered. Scaling can also be applied to the $C(HI)$ values. Völp et al. [2015] look at an alternative means of obtaining $C(LO)$ and $C(HI)$ values; they do not consider them to be estimates of WCET, but rather budgets set by some design optimisation process. As scaling involves changing a task's execution time budget, and this influences priority assignment, it is possible to extend this approach by also allowing priorities to change as the system is made more robust [Burns and Baruah 2013]. A more dynamic budget management scheme is used by Gu and Easwaran [2016] to postpone criticality mode changes. Issues of robustness are also addressed by Herman et al. [2012].

3.1.3. Period transformation. As Vestal [2007] noted, an older protocol, called *period transformation*(PT) [Sha et al. 1986, 1987], is also applicable to the mixed criticality scheduling problem. Period transformation splits a task with period T and execution time C into N parts so that the revised task has parameters T/N and C/N. Assuming all tasks have deadlines equal to their periods, then if all high-criticality tasks are transformed so that their new periods are shorter than those of the low-criticality tasks and rate monotonic priority assignment [Liu and Layland 1973] is used, then the tasks will be partitioned by priority, with the high-criticality tasks having the higher priorities. This is referred to as criticality monotonic priority ordering. The scheme can easily be extended to task sets with constrained deadlines ($D < T$).

Period transformation introduces extra overheads from the increased number of context switches, and these could be excessive if there are low-criticality tasks with short deadlines. If overheads are ignored then period transformation performs well. Baruah and Burns [2013] show that this is primarily due to the inherent property of period transformation to deliver tasks sets with harmonic periods, which are more likely to be schedulable.

For multiple criticality levels a number of transformations may be required to generate a criticality monotonic priority ordering [Fleming and Burns 2013]. For example, assume there are three tasks (H, M, and L) with criticality levels implied by their names, and periods of 14, 36 and 16. First the period of M must be divided by 3 to get a period of 12 (so less than 16), but then the period of H must be divided by 2 to move it below the new period of M. As a result the transformed periods become 7, 12 and 16. It also seems that the theoretical benefit of period transformation diminishes with an increased number of criticality levels [Fleming and Burns 2013].

3.2. EDF Scheduling

Baruah and Vestal [2008] were the first to consider EDF scheduling for MCS. Park and Kim [2011] later introduced a slack-based mixed criticality scheme for EDF scheduled jobs which they called CBEDF (Criticality Based EDF). CBEDF makes use of a combination of off- and on-line analysis to run high-criticality jobs as late as possible, and low-criticality jobs in the generated slack. In effect utilising an older protocol developed by Chetto and Chetto [1989] for running soft real-time tasks in the 'gaps' produced by running hard real-time tasks so that they just meet their deadlines.

A more complete analysis for EDF scheduled systems was presented by Guan et al. [2011] and Ekberg and Yi [2012]. They assigned two relative deadlines to each high-criticality task. One deadline is the 'real' deadline of the task, the other is an artificial earlier deadline that is effectively used to increase the priority of high-criticality tasks enabling them to execute before low-criticality ones. When the criticality mode of the system changes from low to high, due to a high-criticality task exceeding its low-criticality budget $C(LO)$, all low-criticality tasks are abandoned and the high-criticality tasks revert to their real deadlines. Guan et al. [2011] and Ekberg and Yi [2012] demonstrate that this approach provides a clear improvement over previous schemes, including those for fixed priority scheduling of MCS. Later work by Ekberg and Yi [2014] generalises the model to include changes to all task parameters and to incorporate more than two criticality levels. Tighter analysis was provided by Easwaran [2013], although it is not clear that this method scales to more than two criticality levels. Further improvements were presented by Yao et al. [2014]. They use a more efficient schedulability test for EDF, called QPA [Zhang and Burns 2008], and a genetic algorithm to find more effective artificial deadlines.

A similar scheme, called EDF-VD (EDF - with virtual deadlines), was presented by Baruah et al. [2011a, 2015] for dual-criticality systems. With EDF-VD, high-criticality tasks have their deadlines reduced if necessary during low-criticality mode; however, unlike the previous approaches, all deadlines are reduced by the same factor. Both theoretical results and empirical evaluations demonstrate that EDF-VD is an effective scheme. EDF-VD was shown to have a speedup factor of $\phi \approx 1.618$ for single processor systems [Baruah et al. 2011a], with this bound later improved to 4/3 (1.333) [Baruah et al. 2012]. Further formal analysis of EDF-VD was provided by Li [2013], Muller and Masrur [2014] and Gu and Easwaran [2014].

An intermediate approach that uses just two scaling factors is provided by Masrur et al. [2015]; their motivation being to develop an efficient scheme that could be used at run-time. Later work by Baruah [2016c] has generalised the underlying model to include criticality-specific values for period and deadline as well as WCET.

EDF scheduling of MCS was also addressed by Lipari and Buttazzo [2013] using a reservation-based approach. Here sufficient budget is reserved for the high-criticality tasks; however, if they only make use of what is assumed by their low-criticality requirements, i.e. $C(LO)$, then a set of low-criticality tasks can also be guaranteed. Again only two criticality levels are assumed. In effect low-criticality tasks run in the capacity that is 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 and Zhu [2013] and Su et al. [2013], exploiting the *elastic task model* [Buttazzo et al. 1998] in which the period of a task can change. They proposed a minimum level of service for each low-criticality task τ_i that is defined by a maximum period, T_i^{max}. The complete system must be schedulable when all high-criticality tasks execute for their $C(HI)$ WCETs and all low-criticality tasks for their $C(LO)$ WCETs and T^{max} values. At run-time if high-criticality tasks use less than their $C(HI)$ WCETs then the low-criticality tasks can run more frequently. Su and Zhu [2013]; Su et al. [2013] demonstrate that for certain parameter sets their approach outperforms EDF-VD.

3.3. Shared Resources

With MCS it is not clear to what extent data should be permitted to flow between components of different 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 [Sha 2009], which happens with some security protocols [Biba 1977]. Even with data flowing in the other direction there is still the issue of not allowing a high-criticality task to be delayed by a low-criticality one that has either locked a shared resource for longer than expected or is executing at a raised priority level for too long.

Sharing resources within a criticality level is however a necessary part of any practical tasking model. In single criticality systems a number of priority ceiling protocols have been developed [Sha et al. 1990; Baker 1990]. These are beginning to be assessed in terms of their effectiveness for use in MCS. Burns [2013] extended 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) [Sha et al. 1990] has some useful properties when applied to MCS. Resources can be easily partitioned between criticality levels and starvation of low-criticality tasks while holding a lock on a resource can be prevented. With the 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 [2016] employs Hardware Transactional Memory to roll back any shared object to a previous state if a low-criticality task overruns its budget while accessing the object.

For EDF-based scheduling Zhao et al. [2013a, 2015] attempt to integrate the Stack Resource Protocol (SRP) [Baker 1990] and Preemption Threshold Scheduling [Wang and Saksena 1999] 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 have been proposed by Lakshmanan et al. [2011] extending their single processor zero slack scheduling approach 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 protocols use the concept of criticality inheritance. This is also used by Zhao et al. [2014] in their HLC-PCP (Highest-Locker Criticality Priority Ceiling Protocol) which is applied to the fixed priority AMC scheme. For a dual-criticality system they define three modes of execution, low- and high-criticality modes plus an intermediate mode which covers the time during which low-criticality tasks are allowed to continue to execute if they are holding a lock on a resource that is shared with a high-criticality task.

A more systematic scheme has been proposed by Brandenburg [2014]. 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 standards required by 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 addressed by Chisholm et al. [2016].

3.4. Static and Synchronous Scheduling

The change from one criticality mode to another can be captured in a static schedule by switching between previously computed schedules; one per criticality level. This idea was first explored by Baruah and Fohler [2011]. Later, Socci et al. [2013b] showed how these Time-Triggered (TT) tables can be produced via first simulating the behaviour one would obtain from the equivalent fixed priority schedule. Construction of the tables via tree search is addressed by Theis et al. [2013] and Socci et al. [2015b], and via the use of linear programming (LP) by Jan et al. [2014]. For legacy systems, Theis and Fohler [2013] 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 for multiprocessor systems [Baruah and Burns 2014; Burns et al. 2015; Burns and Baruah 2015; Fleming and Burns 2015; Fleming et al. 2016; Fleming and Burns 2016] in which the change from minor cycle to minor cycle is synchronised as is the change from executing code of one criticality level to that of another. Both global and partitioned approaches are investigated, as are systems that use fewer processors for the high-criticality work [Fleming and Burns 2016].

This use of tables is extended to synchronous reactive programs by Baruah [2012b, 2013a]. Here a DAG (Directed Acyclic Graph) of basic blocks that execute according to the synchrony assumption is produced that implements a dual-criticality program. The synchronous approach is also considered by Yip et al. [2014] and by Cohen et al. [2014]; the latter proving an application of mixed criticality from the railway industry, and giving an example of why data needs to flow between criticality levels. An initial study, within the context of multiprocessor federated systems is provided by Baruah [2016a].

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 [2013] consider power issues that could lead to a processor having variable speed. As the processor slows down, so the execution times of the tasks increase. Baruah and Guo [2013] 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 high-criticality jobs are executed under EDF scheduling. This work has been extended [Guo and Baruah 2014; Baruah and Guo 2014; Guo 2016] to include a more expressive model and issues of processor self-monitoring (or not), and a probabilistic approach to performance variation. Guo and Baruah [2015] have also considered systems which have uncertainty in both execution times and processor speed.

4. MULTIPROCESSOR ANALYSIS

The first paper to discuss mixed criticality within the context of multiprocessor or multi-core platforms was by Anderson et al. [2009], later extended by Mollison et al. [2010]. Five levels of criticality were identified; going from level-A (the highest) to level-E (the lowest). They envisaged an implementation scheme, which they called MC^2, that uses a static schedule for level-A, partitioned preemptive EDF for level-B, global preemptive EDF for levels C and D, and finally global best-effort scheduling for level-E. They considered only harmonic workloads, but allowed slack to move between containers. Each processor has a container (server) for each criticality level, and a two-level hierarchical scheduler (see Section 7.3). Later work by Herman et al. [2012] and Chisholm et al. [2015] evaluates the OS-induced overheads associated with multiprocessor platforms. They also experimented with isolation techniques for LLC (last level cache) and DRAM. Further, Kim et al. [2016] demonstrated the benefits of having different isolation techniques for each criticality level using MC^2. The MC^2 framework was also used by Bommert [2013] to support segmented mixed criticality parallel tasks. Parallel jobs are also considered by Liu et al. [2014].

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.

4.1. Task Allocation

The issue of allocation was addressed by Lakshmanan et al. [2010] by extending their single processor slack scheduling approach to partitioned multiprocessor systems employing a Compress-on-Overload packing scheme. Allocation in a distributed architecture was addressed by Tamas-Selicean and Pop [2011a,c,b, 2015] in the context of static schedules 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 such as Simulated Annealing [Tamas-Selicean and Pop 2011b; Giannopoulou et al. 2015] and Tabu search [Tamas-Selicean and Pop 2011a,c]) to obtain schedulability with minimum resource usage. Zhang et al. [2013] used genetic algorithms for task placement in security-sensitive MCS, with the objective of minimising energy consumption “while satisfying strict security and timing constraints”. A tool-set to aid partitioning was provided by Alonso et al. [2014].

A more straightforward investigation of task allocation was undertaken by Kelly et al. [2011]. They considered partitioned homogeneous multiprocessors and compared First-Fit and Best-Fit schemes, with pre-ordering of the tasks based on either Decreasing Utilization or Decreasing Criticality. They used the original analysis of Vestal [2007] 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 was reported by Rodriguez et al. [2013]. They consider EDF scheduling and used the analysis framework of EDF-VD (see Section 3.2). One of their conclusions was to highlight the effectiveness of a combined criticality-aware scheme in which high-criticality tasks are allocated Worst-Fit and low-criticality tasks First-Fit, both in Decreasing Density order. The same result is reported by Gu et al. [2014]. They additionally note that if there are some very ‘heavy’ low-criticality tasks (i.e. high utilisation or density) then space must be reserved for them before the high-criticality tasks are allocated. Partitioning with EDF-VD is also addressed by Han et al. [2016].

A global allocation scheme for MCS is proposed by Gratia et al. [2014, 2015]. They adapt the RUN scheduler [Regnier et al. 2011], which uses a hierarchy of servers, to accommodate high- and low-criticality tasks. The latest version of their scheduler (GMC-RUN) [Gratia et al. 2015] has been extended to deal with more than two criticality levels.

Between fully partitioned and fully global scheduling is the class of schemes referred to as *semi-partitioned*. Adaptation of semi-partitioned schemes to MCS has been addressed by Al-Bayati et al. [2015]. This approach uses two allocations for the two criticality modes. high-criticality tasks do not migrate. During a mode change, carry-over low-criticality jobs are dropped and new low-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 [2015]; here a mode change in one processor results in low-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 requirements of all high-criticality tasks are still met.

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. [2011] for independent periodic tasks running on a multiprocessor system-on-chip. They provide reliability analysis that is used to inform task allocation.

A more theoretical approach, that is not directly implementable, is proposed by Lee et al. [2014] with their MC-Fluid model. A fluid task model [Baruah et al. 1996; Holman and Anderson 2005] 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. This is done by Lee et al. [2014] who also produce an implementable version of the model that performs well in simulation studies. Lee et al. [2014] showed that the speedup factor for the MC-Fluid scheduling algorithm is $\phi \approx 1.618$. Baruah et al. [2015b, 2016] derived a simplified fluid scheduling algorithm which they call MCF. Two further algorithms, MC-Sort and MC-slope, were later proposed by Ramanathan and Easwaran [2015] and Ramanathan et al. [2016].

All the above work is focussed on standard single threaded tasks. In addition there has been some studies on MCS with parallel tasks [Liu et al. 2014; Li et al. 2016].

4.2. Schedulability Analysis

For globally scheduled systems, where jobs can migrate from one processor to another, Li and Baruah [2012] take the multiprocessor scheme fpEDF [Baruah 2004] and combine it with their EDF-VD approach (see Section 3.2). Extensions of this work by Baruah et al. [2014] compare the use of partitioned versus global scheduling for MCS. They show that partitioning combined with EDF-VD yields a speedup factor of $8/3 - 4/3m$ for the dual-criticality scheduling problem on m processors. As noted in van der Ster's abstract in [Baruah et al. 2015a], an alternative theoretical approach, is to view the problem as one of vector scheduling [Chekuri and Khanna 2004] where each dimension corresponds to a criticality level. Combined with EDF-VD, this yields a speedup factor of $4/3 \approx 1.333$ for any m. By comparison, Baruah et al. [2014] show that combining EDF-VD with the global scheduling algorithm fpEDF yields a speedup factor of $\sqrt{5}-1 \approx 1.236$. Despite the global approach having a better speedup factor, the interim conclusion from empirical evaluation is that partitioning is the more effective approach in practice [Baruah et al. 2014].

Notwithstanding this result, Pathan [2012] derives analysis for globally scheduled fixed priority systems. He adopts the AMC approach [Baruah et al. 2011b] (see Section 3.1) and integrates this analysis for multiprocessor scheduling.

A different and novel approach to multi-core scheduling of MCS is provided by Kritikakou et al. [2013]. They identify that a high-criticality task will suffer interference from a low-criticality task running on a different core due to the use of shared buses and memory controllers etc. They monitor the execution time of the high-criticality task and can identify when no further interference can be tolerated. At this point they abort the low-criticality task even though it is not directly interfering. An implementation on a multi-core platform demonstrated the effective performance of their scheme [Kritikakou et al. 2014b]. Extensions to deal with precedence constraints were given by Socci et al. [2015a] 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), for example with a NoC (Network-on-Chip), more resources have to be shared between criticality levels. The first design issue is therefore one of partitioning, how to ensure the behaviour of lower criticality components does not adversely impact on the behaviour of higher criticality components. Pellizzoni et al. [2009] were the first to consider the deployment of MCS on multi-core and many-core platforms. They defined an Architectural Analysis and Design Language (AADL) for mixed criticality applications that facilitates system monitoring and budget enforcement of all computation and communication. Later Obermaisser et al. [2014] and Obermaisser and Weber [2014] introduced 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. [2010] showed that a task can suffer a 300% increase in its WCET 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. [2012] proposed a memory throttling scheme for MCS. Kotaba et al. [2013] also proposed a monitoring and control protocol to prevent processes flooding any shared communication media be it a bus or network. Kritikakou et al. [2014a] considered a scenario in which there are a few high-criticality tasks that can suffer indirect interference from many lower criticality tasks. Their approach attempts to allow as much parallelism as possible, commensurate with the high-criticality tasks retaining their temporal validity. Hassan and Patel [2016] 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. [2014].

Within the time-triggered model of distributed computation and communication MCS are often viewed as having both time-triggered and event-triggered activities, also referred to as synchronous and asynchronous [Pop et al. 2002; Steiner 2011]. The time-triggered traffic is deemed to have the highest criticality. The event-triggered traffic can either be viewed as best-effort or can be given some level of assurance if its impact on the system is bounded, referred to by Steiner [2011] as *rate-constrained*. Protocols that support this distinction can be implemented on TTEthernet.

Another TDMA-based approach, though this time built into the Real-Time Ethernet protocol, is proposed by Carvajal and Fischmeister [2013] in their open-source framework, *Atacama*. Cilku et al. [2015] describe a TDMA-based bus arbitration scheme. Novak et al. [2016a] propose a scheduling algorithm for time-triggered traffic that minimises jitter while allowing high-criticality messages to be re-transmitted following failure at the expense of low-criticality messages which are abandoned. Novak et al. [2016b] also consider how to produce an effective static schedule when there are unforeseen re-transmissions.

A reconfigurable SDRAM controller is proposed by Goossens et al. [2013c] to schedule concurrent memory requests to the same physical memory. They 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 applications. For example, a criticality mode change which results in more bandwidth being assigned to the higher criticality tasks can be accommodated. Criticality aware DRAMs are also addressed by Jalle et al. [2014] in the context of a Space case study in which there are two criticality levels.

Virtual devices are adopted by Ecco et al. [2014] to isolate critical tasks (which are guaranteed) from non-critical tasks that, although not guaranteed, perform adequately. Each virtual device represents a group of DRAM banks and 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 interference between criticality levels is from the interconnect. By use of virtual devices, the critical tasks benefit from interference-free memory access. DRAMs are also the focus of the work by Hassan et al. [2015] and Awan et al. [2016].

Kim et al. [2015] 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. A similar approach and result is provided by Goossens et al. [2013a] with their open-page policy.

Giannopoulou et al. [2013, 2015] use a different time-triggered approach. They partition access to the multiprocessor bus so that at any given time only memory accesses from tasks of the same criticality can occur. This may introduce some inefficiencies; however, it has the advantage that it reduces the temporal modelling of a mixed criticality shared bus to that of a single criticality shared bus. In later work Huang et al. [2015] generalise the approach by introducing the notion of isolation scheduling. The problems involved in using a shared bus has lead Giannopoulou et al. [2015] to also include a Network-on-Chip (NoC) in their later work.

Baruah and Burns [2014], Burns et al. [2015], Burns and Baruah [2015], and Fleming and Burns [2015] apply a one criticality at a time approach to MCS scheduled using a cyclic executive; they considered both partitioned and global allocation of jobs to frames.

Tobuschat et al. [2013] have developed a NoC explicitly to support MCS. Their IDAMC protocol uses a *back suction* technique [Diemer and Ernst 2010] to maximise the bandwidth given to low-criticality or non-critical messages while ensuring that high-criticality messages arrive by their deadlines. The more familiar wormhole routing scheme [Ni and McKinley 1993] for a NoC has been expanded by Burns et al. [2014] and Indrusiak et al. [2015] to provide support for mixed criticality traffic. Response-time analysis, already available for such protocols [Shi and Burns 2008], is augmented to allow the size and frequency of traffic to be criticality dependent. Wormhole routing is also used by Hollstein et al. [2015] 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 [Broster and Burns 2003]. This is provided via a time-triggered extension layer for a mixed-criticality NoC by Ahmadian and Obermaisser [2015]. Dynamic control of a mixed-criticality NoC is considered by Kostrzewa et al. [2015]. Control over I/O contention via an Ethernet-based criticality-aware NoC is advocated by Abdallah et al. [2016]. NoC security, in which high-criticality messages need more protection than those of low-criticality, is considered by Papastefanakis et al. [2016].

An alternative to using a NoC for all traffic (task to task communication and task to off-chip memory) was proposed by Audsley [2013] and Gomony et al. [2016]. They advocate the use of a separate memory hierarchy to link each core to off-chip memory. A criticality aware protocol is then 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 applied. The separation of execution time from memory-access time is explored by Li and Wang [2016]. They demonstrate that this distinction improves schedulability.

Controller Area Network (CAN) [Bosch 1991] 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 by Davis et al. [2007] for what is effectively a fixed priority non-preemptive protocol. The use of CAN in mixed criticality applications has been addressed by Burns and Davis [2013]. 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 uses a Trusted Network Component to police the traffic that nodes are allowed to send over the network.

Herber et al. [2013] 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 MCS include FlexRay [Goswami et al. 2012] and switched Ethernet [Cros et al. 2014, 2015]. 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).

Addisu et al. [2013] 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.

Finally, Jin et al. [2015, 2016] provide delay analysis for fixed priority scheduling in sensor networks.

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

In some MCS it may be acceptable to provide only limited timing guarantees following a criticality mode change, and hence no online controls are required [von der Brüggen et al. 2016]; however, where abandonment of all low-criticality tasks is not acceptable, then a number of reconfigurations are possible:

(1) Let any low criticality job that has started run to completion; this is in effect what is assumed by many forms of analysis [Baruah et al. 2011b].
(2) Reduce the priorities of the low-criticality tasks [Baruah and Burns 2011], or similar for EDF scheduling by changing task deadlines [Huang et al. 2013, 2014].
(3) Increase the periods and deadlines of low-criticality jobs [Su et al. 2013; Su and Zhu 2013; Jan et al. 2013; Su et al. 2014, 2016a], referred to as *task stretching* or the *elastic task model*.
(4) Impose only weakly-hard constraints on the low-criticality tasks [Gettings et al. 2015].
(5) Decrease the computation times of low-criticality tasks [Burns and Baruah 2013]
(6) Move some low-criticality tasks to a different processor that has not experienced a criticality mode change [Xu and Burns 2015].
(7) Abandon low-criticality work in a disciplined sequence [Fleming and Burns 2014; Huang et al. 2013; Gu et al. 2015; Ren and Phan 2015].

The fifth approach leads to a modification to the system model; whereas for high-criticality tasks we have $C(HI) \geq C(LO)$, for low-criticality tasks we now have $C(HI) \leq C(LO)$. For some $C(HI) = 0$ i.e. they are abandoned, for others a lower level of service can be guaranteed, while yet others may be able to continue with no change in budget.

The final approach is addressed by Fleming and Burns [2014]; they introduce a further notion into the standard model; tasks are allocated to *applications* and each application is assigned an *importance* level by the system designer. Low-criticality tasks are then abandoned in inverse order of importance. Huang et al. [2013] also introduce an extension to the standard model via an ICG (Interference Constraint Graph) used to capture which tasks need to be dropped when particular higher criticality tasks exceed their allocated criticality-aware execution times. Controlled abandonment via the use of partitioning is advocated by Mahdiani and Masrur [2016] in the context of EDF-VD scheduling.

A specific approach is advocated by Su et al. [2016b], whereby low-criticality tasks have two periods, short and long, and two priorities. On the criticality mode change the tasks switch to their longer periods and new priorities. Analysis is provided to show that all modes are schedulable.

A flexible scheme utilising hierarchical scheduling is proposed by Easwaran and Shin [2014]. They differentiate between minor violations of low-criticality execution time which can be dealt with within a component via 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 compositionality.

In keeping with operational mode changes a simple protocol for controlling the time of the change of mode back to low criticality is to wait until the system is idle, i.e. has no application tasks to run, and then the change can safely be made [Tindell and Alonso 1996]. This approach is extended by Santy et al. [2013] to produce a somewhat more efficient scheme that can be applied to globally scheduled multiprocessor systems in which an idle tick may never occur. With a dual criticality system that has just transitioned into the high-criticality mode, and hence no low-criticality jobs are executing, the protocol first waits until the highest priority high-criticality job completes, then it 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 of the low-criticality tasks. Obviously if there is a further violation of the $C(LO)$ bound then the recovery protocol is restarted. The authors call this Safe Criticality Reduction.

A more aggressive scheme for returning a MCS back to its low-criticality mode is proposed by Bate et al. [2015, 2016], referred to as the *bailout protocol*. Here, high-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 time than budgeted. When the loan is repaid and a further condition met, the system returns to low-criticality mode. The authors demonstrate that the bailout protocol returns the system to the low-criticality mode much quicker than the simple 'wait for idle tick' scheme, and results in far few deadline misses and un-started jobs for low-criticality tasks.

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 [2014] compares and contrasts these two forms of mode change. Niz and Phan [2014] note that the criticality of a task can depend on the behavioural mode of the system. They develop schedulability 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 five criticality levels would also have five different estimates of WCET for its most critical tasks. An augmented model has been proposed [Burns 2015; Niz et al. 2009] that restricts each task to having just two WCET estimates. 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 WCET at its own criticality level ($C_i(L_i)$) and the other is 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 seem to be sufficiently expressive to capture most of the key properties of MCS. However, Baruah and Guo [2015] showed 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."*

6. SYSTEMS ISSUES

A fundamental issue with MCS is separation. Many of the theoretical papers reviewed here assume various levels of run-time monitoring and control; however, few consider how the required mechanisms can be implemented. Neukirchner et al. [2011] addressed this issue, focusing on memory protection, timing fault containment, admission control and (re-)configuration middleware. Their framework [Farrall et al. 2013] is aimed at supporting AUTOSAR conforming applications within the automotive domain.

A detailed study of the overheads for two common implementation schemes for MCS is presented by Sigrist et al. [2014]. They conclude that overheads of up to 97% can be encountered and recommend that scheduling models be extended to include parameters that capture the impact of run-time overheads.

The issue of monitoring is also addressed by Motruk et al. [2012] in the context of their IDAMC (Integrated Dependable Architecture for Many Cores). This work builds on more general work on separation, isolation and monitoring for SoC/NoC architectures. Alonso et al. [2013], Trujillo et al. [2013, 2014], and Cilku et al. [2014, 2015] address virtualisation in terms of Model Driven Engineering for MCS. Virtualisation has also been investigated by Goossens et al. [2013b] "to allow independent design, verification and execution" using their CompSOC architecture. Paravirtualisation of legacy RTOSs to provide the necessary memory isolation is considered by Armbrust et al. [2014].

Hypervisor technology is also being used to provide an appropriate level of isolation in MCS. Larrucea et al. [2016] use it to minimise interference via modelling patterns of execution. Evripidou and Burns [2016] employ different execution-time servers (deferrable server for event-triggered work, and periodic server for periodic work) under the control of a hypervisor to bound the overheads associated with using server technology. If there is a criticality mode change then the deferrable servers are transposed to more efficient but less responsive periodic servers. A general hypervisor architecture for multi-core MCS is presented by Pérez et al. [2017].

The development of purpose built FPGA-based hardware is being undertaken with the aim of reducing the cost of certification for MCS on multiprocessor architectures via the use of open source hardware and software [Pop et al. 2013; Nevalainen et al. 2013; Mendez et al. 2013]. Other research looking at systems built on FPGA platforms includes the development of a criticality-aware scrubbing mechanism that improves system reliability by up to 79% [Santos et al. 2014]. Scrubbing is a technique for recovering from single event upsets that affect 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, hence much of the extensive literature on security is relevant, but outside of the scope of this survey. Some work is nevertheless applicable to both safety and security, for example the definition of a *separation kernel* for a system-on-chip built using a time-triggered architecture [Wasicek et al. 2010]. Such a separation kernel has been developed by Li et al. [2013], West et al. [2016], and Missimer et al. [2016]. This kernel can host guest operating systems, such as Linux or their own RTOS QUEST-V. The kernel partitions the available cores into Sandboxes that have different criticality levels. The 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, so that they do not have to be first handled by the hypervisor.

PikeOS [Kaiser 2007] also employs a separation microkernel [Saidi et al. 2015] to provide "a powerful and efficient paravirtualization real-time operating system" for a partitioned multi-core platform. Lyons and Heiser [2014] show how the high-assurance microkernel sel_4. model can be extended to cater for mixed criticality. Virtual Machines (VMs) that are appropriate for real-time Java-based MCS have been designed by Ziarek and Blanton [2015] and Hamza et al. [2015].

To implement the criticality mode change it is necessary for the run-time system to support execution time monitoring, the set of modes, and the mode changes. Baruah and Burns [2011] show how this can be achieved within the facilities provided by the Ada programming language. Kim and Jin [2014] do the same for a standard RTOS.

A further operating system designed to support mixed criticality is Kron-OS [David et al. 2014]. This controls the execution of RSFs (Repetitive Sequence of Frames) that are partitioned between two criticality levels.

As an alternative to using an RTOS to give the right level of protection and resource sharing, Zimmer et al. [2014] designed a processor (FlexPRET) to directly support MCS. They use fine-grained multi-threading and scratchpad memory to give protection to hard real-time tasks while increasing the resource utilisation of soft tasks. In effect soft tasks can safely exploit the spare capacity generated by the hard tasks at the cycle level. A more focused scheme aimed at partitioning the cache is described by Lesage et al. [2012]. A Least Critical (LC) cache replacement policy is evaluated by Kumar et al. [2014]. The effective use of cache is also considered by Chrisholm et al. [2015] for a multi-core platform.

A hardware platform that supports applications of different criticality levels must manage its I/O functions in a partitioned and hence safe and secure way. If lower criticality work can cause interrupts to occur at any time then unpredictable overheads may be incurred by high-criticality applications. This often overlooked topic is addressed by Paulitsch et al. [2015].

Although research on MCS has generated many different approaches, there have been few empirical benchmarks or comparative studies. One useful study was published by Huang et al. [2012]. They compared the scheme of Vestal [2007] with its optimal priority assignment, their improved slack scheduling scheme and period transformation. They conclude that Vestal's approach and period transformation usually, although not always, outperform slack scheduling; and that there are additional though not excessive overheads with period transformation and slack scheduling. Later Fleming and Burns [2013] compared Vestal's approach, AMC (see Section 3.1) and period transformation for multiple criticality levels. As the number of criticality levels increased the relative advantage of period transformation was observed to decrease, even when overheads are ignored. This observation was also supported by Huang et al. [2014a] 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 a user-space implementation of AMC on top of Linux (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. [2015]. The use of realistic simulations to evaluate scheduling schemes is discussed by Bate et al. [2015, 2016], Griffin et al. [2015] and Ittershagen et al. [2015]. A brief comparison of approaches to multiprocessor scheduling of MCS is provided by Osmolovskiy et al. [2016]. The evaluation of communications within MCS is considered by the work of Napier et al. [2016] and Petrakis et al. [2016].

Another systems issue of crucial importance in many mobile embedded systems is power consumption. The work of Broekaert et al. [2013] allocates and monitors power budgets for different criticality levels. If a crucial Virtual Machine (VM) "*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. [2013]. 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. [2013] to drive task allocation in a multiprocessor system. A slightly different approach is taken by Huang et al. [2014b]. They advocate the use of Dynamic Voltage and Frequency Scaling (DVFS) to increase the speed of the processor if high-criticality tasks need more than their $C(LO)$ requirement. Hence low-criticality jobs are not abandoned, but more energy is used. They integrate their approach with the EDF-VD scheduling scheme (see Section 3.2) and have since addressed multi-core platforms [Narayana et al. 2016]. This approach is extended by Ali et al. [2015] who propose a new dynamic power-aware scheduling scheme for hardware with discrete frequency levels. DVFS management is also addressed by Haririan and Garcia-Ortiz [2015] in their provision of a simulation framework for power management.

Where energy is limited or indeed the system is energy neutral, then criticality-aware energy usage becomes crucially important [Völp et al. 2014]. ENOS [Wagemann et al. 2016] 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. Energy harvesting in the context of a battery-less real-time system is considered by Asyaban et al. [2016]. They propose a scheduling scheme that satisfies both temporal and success-ratio constraints while 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 that has been addressed in the context of heterogeneous MPSoCs [Grüttner 2017].

Complex MCS also present a number of significant challenges at the specification and design stage. Herrera et al. [2013, 2015] 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. [2016] support the development of MCS on multi-core platforms via the development of an appropriate tool chain. This group has also considered the mapping and design of fault-tolerant MCS to multicore platforms [Giannopoulou et al. 2014; Zeng et al. 2016].

7. LINKS TO OTHER RESEARCH TOPICS

In this section, we explore the links between research into MCS, and other related topics.

7.1. Hard and Soft Tasks

Although the label "Mixed Criticality Systems (MCS)" is relatively new, many older results and approaches can be reused and reinterpreted under this umbrella term. In particular, systems in which there are tasks with hard and soft real-time constraints have been studied since the 1980s. Hard tasks must be guaranteed, while soft tasks are then given the best possible service. Soft tasks are usually unbounded, in terms of their execution time or their arrival frequency, and hence must be constrained to execute only from within some form of execution-time server. Such servers have a bounded impact on the hard tasks. A number of different servers have been proposed in the literature. The major ones for fixed priority systems are the Periodic Server, the Deferrable Server, the Priority Exchange Server (all described by Lehoczky et al. [1987]), and the Sporadic Server [Sprunt et al. 1988]. These servers have equivalent protocols for dynamic priority (EDF) systems; and some EDF specific ones exist such as the Constant Bandwidth server [Abeni and Buttazzo 1998]. The ability to run soft tasks in the slack provided by the hard tasks is also supported by Static Slack Stealing [Lehoczky and Ramos-Thuel 1992], Dynamic Slack Stealing [Davis et al. 1993], and Dual-Priority schemes [Davis and Wellings 1995].

The standard servers only deal with the isolation / partitioning aspect of MCS. To support sharing of processing 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 [Sprunt et al. 1988] as well as work on making use of gain time [Davis 1995], show how this can be achieved. These ideas have been incorporated into protocols aimed at providing support for low criticality tasks in MCS [Niz et al. 2009; Bate et al. 2015]. The idea of slack stealing has been used as a basis for scheduling MCS (see Section 3.1.2).

Run-time adaptability for MCS has been addressed by Hu et al. [2015, 2016a]. They present an approach to adaptively shape at runtime the inflowing workload from low-criticality tasks based on the actual demand of high-criticality tasks. This improves the QoS of low-criticality tasks, but it not clear what level of guarantee is provided for these tasks. An alternative scheme, with the same aim, is given by Hikmet et al. [2016].

7.2. 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. Further, this work must be less important than the tasks that are being re-executed. It follows that many fault tolerant systems are in effect MCS.

Timeouts are often used to identify a fault. A job not completing before a deadline is evidence of some internal problem. Earlier warning can come from noting that a job is trying to execute for more than its execution time budget. Execution time monitoring is therefore common in safety critical systems that are required to have some level of fault tolerance. Again this points to common techniques being required in FTS and MCS.

As noted earlier in the discussion on CAN (Section 4.3), a fault model can be criticality dependent [Burns and Davis 2013]. 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 execution time budgets at different criticality levels, may be a result of the inclusion or not of recovery techniques in the WCET estimates of the tasks.

Although there is this clear link between FTS and MCS there has not as yet been much work published that directly addresses fault-tolerant MCS. Exceptions being the work of Huang et al. [2014] and a paper by Pathan [2014] that both focus on service adaptation and the scheduling of fault-tolerant MCS; and a four mode model developed by Al-Bayati et al. [2016]. Zonal Hazard Analysis and Fault Hazard Analysis [Thekkilakattil et al. 2014b] and Error-Burst models [Thekkilakattil et al. 2014a] can be used to deliver both flexibility and real-time guarantees for the most critical tasks. Thekkilakattl et al. [2015] also consider the link between MCS and the tolerance of permanent faults. Lin et al. [2014] attempt to integrate mixed criticality with the use of primary and backup executions in both of the two criticality modes they consider. Islam et al. [2006], in a paper that preceded that of Vestal [2007], looked at combining different levels of replication for different levels of criticality.

As highlighted already in this survey, many models and protocols for mixed criticality behaviour allow the system to move through a sequence of criticality modes. With a dual-criticality system the system starts in the low-criticality mode in which all deadlines of all tasks are guaranteed, but can then transition to the high-criticality mode in which only the high-criticality tasks are guaranteed It may, or may not, later return to the low-criticality mode when it is safe to do so. Burns [2014] compared these criticality mode changes with the more familiar system (operational) mode changes. He concludes that the low-criticality mode behaviour should be considered as the 'normal' expected behaviour. A move away from this mode is best classified as a fault; with all other modes being considered forms of *graceful degradation*. A move back to the fully functional low-criticality mode is closest in nature to an *operational* mode change. Such a mode change is *planned for but may never occur*.

7.3. Hierarchical Scheduling

One means of implementing 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 tasks of the application. There are a number of relevant results for such resource containment schemes (e.g. [Saewong et al. 2002; Shin and Lee 2003; Davis and Burns 2005; Lipari and Bini 2005; Davis and Burns 2006; Zhang and Burns 2007; Checconi et al. 2009]). Both single processor and multiprocessor platforms can support hierarchical scheduling.

Unfortunately when hierarchical scheduling is applied to MCS there is a loss of performance [Lackorzynski et al. 2012]. A simple interface providing a single budget and replenishment period, which is often associated with virtualisation or the use of a hypervisor [Alonso et al. 2013]), is too inflexible to cater for a system that needs to switch between criticality modes. To provide a more efficient scheme, Lackorzynski et al. [2012] proposed 'flattening' the hierarchy by exposing some of the internal structure of the scheduled applications. They developed the notion of a scheduling context, which they apply to MCS [Volp et al. 2013]. 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 approach is provided by Groesbrink et al. [2013, 2014]. 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. [2012]. They are more concerned with partitioning as opposed to resource usage, and address distributed heterogeneous architectures. Hypervisors are also used by Cilku and Puschner [2013] to give temporal and spacial separation on a multiprocessor platform. Perez et al. [2013] use a hierarchical scheduler to statically partition a mixed criticality wind power system requiring certification under the IEC-61508 standard. A hypervisor for a mixed criticality on-board satellite software system is discussed by Salazar et al. [2014], and Alonso et al. [2015], and one for general control systems is addressed by Crespo et al. [2014]. The issue of minimising the overheads of a hypervisor is considered by Blin et al. [2016].

7.4. Cyber Physical Systems and Internet of Things

In parallel with the development of a distinct branch of research covering MCS has been the identification of Cyber Physical Systems (CPS) as a useful focus for system development. Not surprisingly it has been noted that many CPS are also MCS. For example Schneider et al. [2013] note that many CPS contain 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 [2015] use safety-critical CPS to explore how metrics can be used to map potential hazards and risks from top-level design down to mixed criticality components on a multi-core architecture. Issues of composability within open CPS are introduced in a short paper by Lee et al. [2016].

Maurer and Kirner [2015] consider the specification of cross-criticality interfaces (CCI) that define the level of communication allowed between open subsystems/components in CPS. Lee et al. [2015] also look at interfaces and composition for mixed criticality CPS.

The link between the Internet of Things (IoT) and MCS is made by Kamienski et al. [2016] in the context of development methods for energy management in public buildings. Smart buildings are also the focus of the work by Dimopoulos et al. [2016] on a context-aware management architecture.

7.5. Probabilistic real-time systems

In MCS, the WCET 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 estimate 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 [Edgar and Burns 2001]. Instead of a number of single values for WCET estimates with different levels of confidence, the worst-case execution time is expressed as a probability distribution, referred to as a pWCET.

The exceedance function (or 1 - CDF[9]) for the pWCET gives the probability that the task will exceed the specified execution time budget on any given run [Cucu-Grosjean 2013]. 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.

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 [Cucu-Grosjean et al. 2012; Cazorla et al. 2013; Davis et al. 2013; Altmeyer et al. 2015] and the probabilistic worst-case response times [Díaz et al. 2002; López et al. 2008; Maxim and Cucu-Grosjean 2013] 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 very low probability of occurring, thus avoiding the need to over-provision compute resources to handle these cases.

Just as MCS research has expanded from a focus on execution times to one that includes arrival rates for sporadic tasks, so probabilistic analysis has been developed for the case where the arrival rate of tasks is described by a probability distribution [Maxim and Cucu-Grosjean 2013]. This work forms the basis for a further link between MCS and probabilistic analysis. Indeed Masrur [2016] uses random jitter on the arrival time of low-criticality tasks to improve schedulability.

[9]Cumulative Distribution Function.

Guo et al. [2015] 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 high-criticality task executing for more than its low-criticality execution time budget $C(LO)$ is also expressed as a probability. Santinelli and George [2015] also explore the probability space of WCETs for MCS. Probabilistic analyses for the SMC and AMC schemes are derived by Maxim et al. [2016]. A Markov decision process is used by Alahmad and Gopalakrishnan [2016] to model job releases in MCS. Probabilistic analysis is also used to investigate the safety of each criticality level by Draskovic et al. [2016].

7.6. Industry Practice and Safety Standards

This survey covers the considerable body of research into MCS stemming from the model presented by Vestal [2007]. Industry practice and safety standards; however, provide a somewhat different perspective on MCS. There are different meanings assumed for some of the commonly used terms, and different objectives. This disconnect has been discussed in a number of papers [Graydon and Bate 2013], [Esper et al. 2015], [Paulitsch et al. 2015] and [Ernst and Natale 2016].

From an industry perspective, *criticality* relates to the functional safety of an application, see, for example, the IEC 61508, DO-178B and DO-178C, DO-254 and ISO 26262 standards. Typical names for criticality levels are ASILs (Automotive Safety and Integrity Levels), DALs (Design Assurance Levels or Development Assurance Levels) and SILs (Safety Integrity Levels).

Determining the criticality of an application (or system function implemented via both hardware and software) is done via a system safety assessment that involves Failure Modes and Effects Analysis (FMAE). The criticality level typically depends on (i) an evaluation of the consequences of a failure, (ii) the probability that the failure occurs, and (iii) the provision of means to mitigate or cope with the fault. Hence the criticality level of an application may not necessary reflect the severity or consequences of failure. An example given by Esper et al. [2015] and Ernst and Natale [2016] comes from ISO 26262. If the probability of failure occurrence is very low, the ASIL level assigned may be low, despite severe consequences if a failure actually happens. A different application with a high probability of failure may be assigned a higher ASIL despite having lower severity consequences. With this interpretation, the idea of dropping low-criticality functionality in favour of completing that of higher criticality does not hold; the consequences would be more severe. ISO 26262 also permits high-criticality applications to be composed from lower criticality components with diverse implementations, again dropping one of the lower criticality components would remove the diversity and undermine the safety argument for the high-criticality function. The message here is that the criticality level is not the same as the importance of the application. Functionality that has low criticality cannot simply be dropped.

The standards require that *"sufficient independence"* or *"freedom from interference"* is demonstrated between functions of different criticality levels in both spatial and timing domains. If this is not done, then the whole system needs to be designed and developed according to methods appropriate for the highest criticality level involved, which would be untenable in practice for cost reasons. It remains a significant challenge to achieve the necessary separation, while also providing an efficient means of sharing resources. This is particularly apposite with the advent of multi-core and many-core platforms.

8. DIRECTIONS FOR FUTURE WORK

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 has concentrated on how to safely partition a system so that high-criticality components can at least run on hardware systems where computational and communication resources are otherwise shared. By comparison, the more theoretical and scheduling research has largely focused on how criticality-specific WCETs can be utilised to deliver systems that are schedulable at each criticality level with a high processor utilisation. Unfortunately these two areas of research are not easily integrated. Flexible scheduling requires as a minimum 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 has only recently begun to address these issues.

What becomes clear from reading the extensive literature that has been produced since the seminal paper by Vestal [2007], is that MCS present a collection of interesting issues that are both theoretically intriguing and challenging from the perspective of implementation. We finish this survey by listing open issues identified from reading the extensive research literature. (Many of these issues were presented by Alan Burns in his keynote talk at the Dagsthul Seminar on Mixed Criticality Systems on Multicore/Manycore Platforms in March 2015).

(1) Holistic analysis is needed considering *all* system resources, particularly communications buses, networks, and access to memory, as well as the processor(s).
(2) Appropriate models of system overheads and task dependencies are required, and need to be integrated into the analysis. In particular, attention needs to be paid to how overheads arising from tasks of one criticality level may impact tasks of different (particularly higher) criticality.
(3) More work is needed to integrate run-time behaviour, i.e. monitoring and control, with the assumptions made during static analysis and verification.
(4) Effective protocols are needed for sharing information between criticality levels.
(5) There are a number of open issues with regards to graceful degradation and fault recovery. These include timely recovery back to the low-criticality mode of operation, and support for limited low-criticality functionality in higher criticality modes, avoiding the abandonment problem.
(6) To be of practical use, techniques need to scale to more than two (possibly up to five) levels of criticality.
(7) Better WCET analysis is needed to reduce the sound $C(HI)$ and $C(LO)$ estimates used, and to improve the confidence in these values.
(8) Much of the existing research has looked at mixed criticality within a single scheduling scheme; however, further work is needed on integrating different schemes (e.g. cyclic executives for safety-critical applications, fixed priority for mission-critical applications, on the same processor).
(9) Mechanisms are needed to tightly bound the impact of lower criticality tasks on those of higher criticality, independent of the behaviour or misbehaviour of the former, without significantly compromising performance, which may happen if strict isolation is enforced.
(10) Time composability is needed across different criticality levels, so that the timing behaviour of tasks determined in isolation can be used when they are composed during system integration.
(11) So far there has been little work on security as an aspect of criticality in real-time systems.
(12) Probabilistic and statistical methods are a good match to requirements specified in terms of failure rates for different criticality levels; however, little work has been done on applying these techniques to MCS.
(13) Openly available benchmarks and case studies are needed for the evaluation of MCS techniques and analysis.
(14) For research on MCS to have real impact it will be necessary to influence the relevant standards in the various application domains (e.g. automotive, aerospace).

Returning to the fundamental question underlying MCS research: how, in a disciplined way, to reconcile the conflicting requirements of *partitioning* for safety assurance and *sharing* for efficient resource usage. As yet we do not have the structures (models, methods, protocols, analysis etc.) needed to allow the tradeoffs between partitioning and separation to be properly evaluated. It is clear 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 MCS.

REFERENCES

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

L. Abeni and G. Buttazzo. 1998. Integrating Multimedia Applications in Hard Real-Time Systems. In *Proc. of the Real-Time Systems Symposium*. Madrid, Spain, 3–13.

A. Addisu, L. George, V. Sciandra, and M. Agueh. 2013. Mixed Criticality Scheduling Applied to JPEG2000 Video Streaming Over Wireless Multimedia Sensor Networks. In *Proc. WMC, RTSS*. 55–60.

H. Ahmadian and R. Obermaisser. 2015. Time-Triggered Extension Layer for On-Chip Network Interfaces in Mixed-Criticality Systems. In *Proc. Digital System Design (DSD)*. IEEE, 693–699.

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

Z. Al-Bayati, Q. Zhao, A. Youssef, H. Zeng, and Z. Gu. 2015. Enhanced partitioned scheduling of Mixed-Criticality Systems on multicore platforms. In *20th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 630–635.

B. Alahmad and S. Gopalakrishnan. 2016. A Risk-Constrained Markov Decision Process Approach to Scheduling Mixed-Criticality Job Sets. In *Proc 4th WMC (RTSS)*. https://hal.archives-ouvertes.fr/hal-01403223

I. Ali, J. Seo, and K.H. Kim. 2015. 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)*. 438–445.

A. Alonso, J.A. de la Puente, J. Zamorano, M.A. de Miguel, E. Salazar, and J. Garrido. 2015. Safety Concept for a Mixed Criticality On-Board Software System. *IFAC-PapersOnLine* 48, 10 (2015), 240–245.

A. Alonso, C. Jouvray, S. Trujillo, M.A. de Miguel, C. Grepet, and J. Simo. 2013. Towards Model-Driven Engineering for Mixed-Criticality Systems: MultiPARTES Approach. In *Proceedings of the Conference on Design, Automation and Test in Europe, WICERT (DATE)*.

A. Alonso, E. Salazar, and M.A. de Miguel. 2014. A Toolset for the Development of Mixed-Criticality Partitioned Systems. In *HiPEAC Workshop*.

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

J.H. Anderson, S.K. Baruah, and B.B. Brandenburg. 2009. Multicore Operating-System Support for Mixed Criticality. In *Proc. of the Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification, San Francisco*.

E. Armbrust, J. Song, G. Bloom, and G. Parmer. 2014. On Spatial Isolation for Mixed-Criticality, Embedded Systems. In *Proc. 2nd Workshop on Mixed Criticality Systems (WMC), RTSS*, L. Cucu-Grosjean and R. Davis (Eds.). 15–20.

S. Asyaban, M. Kargahi, L. Thiele, and M. Mohaqeqi. 2016. Analysis and Scheduling of a Battery-Less Mixed-Criticality System with Energy Uncertainty. *ACM Transactions on Embedded Computing Systems (TECS)* 16, 1 (2016), 23.

N.C. Audsley. 2001. On Priority Assignment in Fixed Priority Scheduling. *Inform. Process. Lett.* 79, 1 (2001), 39–44.

N.C. Audsley. 2013. Memory Architectures for NoC-based Real-Time Mixed Criticality Systems. In *Proc. WMC, RTSS*. 37–42.

N.C. Audsley, A. Burns, M. Richardson, K. Tindell, and A.J. Wellings. 1993. Applying New Scheduling Theory to Static Priority Preemptive Scheduling. *Software Engineering Journal* 8, 5 (1993), 284–292.

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

P. Axer, M. Sebastian, and R. Ernst. 2011. 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)*. ACM, 149–158.

T.P. Baker. 1990. A Stack-Based Resource Allocation Policy for Realtime Processes. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*. 191–200.

J. Barhorst, T. Belote, P. Binns, J. Hoffman, J. Paunicka, P. Sarathy, J. Scoredos, P. Stanfill, D. Stuart, and R. Urzi. 2009. White Paper: A Research Agenda for Mixed-Criticality Systems. (April 2009). Available at http://www.cse.wustl.edu/~cdgill/CPSWEEK09_MCAR.

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

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

S.K. Baruah. 2012b. Semantic-preserving implementation of multirate mixed criticality synchronous programs. In *Proc. RTNS*.

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

S.K. Baruah. 2013b. *Response-time analysis of mixed criticality systems with pessimistic frequency specification*. Technical Report. University of North Carolina at Chapel Hill.

S.K Baruah. 2016a. The Federated Scheduling of Systems of Mixed-Criticality Sporadic DAG Tasks. In *Proc. Real-Time Systems Symposium (RTSS)*. IEEE, 227–236.

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

S.K. Baruah. 2016c. Scheduling analysis for a general model of mixed-criticality recurrent real-time tasks. In *Proc. IEEE RTSS*. 25–34.

S.K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. 2010. Scheduling Real-Time Mixed-Criticality Jobs. In *Proc. of the 35th International Symposium on the Mathematical Foundations of Computer Science (Lecture Notes in Computer Science)*, P. Hlinený and A.ín Kucera (Eds.), Vol. 6281. Springer, 90–101.

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

S.K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. 2012. Scheduling Real-Time Mixed-Criticality Jobs. *IEEE Trans. Comput.* 61, 8 (2012), 1140–1152.

S. Baruah, V. Bonifaci, G. D'angelo, H. Li, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie. 2015. Preemptive Uniprocessor Scheduling of Mixed-Criticality Sporadic Task Systems. *Journal of the ACM (JACM)* 62, 2 (2015), 14.

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

S.K. Baruah, V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. 2011a. Mixed-Criticality Scheduling of Sporadic Task Systems. In *Proc. of the 19th Annual European Symposium on Algorithms (ESA 2011) LNCS 6942, Saarbruecken, Germany*. 555–566.

S.K. Baruah and A. Burns. 2011. Implementing Mixed Criticality Systems in Ada. In *Proc. of Reliable Software Technologies - Ada-Europe 2011*, A. Romanovsky (Ed.). Springer, 174–188.

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

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

S. Baruah, A. Burns, and R.I. Davis. 2013. An Extended Fixed Priority Scheme for Mixed Criticality Systems. In *Proc. ReTiMiCS, RTCSA*, L. George and G. Lipari (Eds.). 18–24.

S.K. Baruah, A. Burns, and R. I. Davis. 2011b. Response-Time Analysis for Mixed Criticality Systems. In *IEEE Real-Time Systems Symposium (RTSS)*. 34–43.

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

S.K. Baruah, B. Chattopadhyay, H. Li, and I. Shin. 2014. Mixed-criticality Scheduling on Multiprocessors. *Real-Time Systems Journal* 50 (2014), 142–177.

S.K. Baruah, A. Easwaran, and Z. Guo. 2015b. MC-Fluid: Simplified and Optimally Quantified. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*. 327–337.

S.K. Baruah, A. Easwaran, and Z. Guo. 2016. Mixed-Criticality Scheduling to Minimize Makespan. In *LIPIcs-Leibniz International Proceedings in Informatics*, Vol. 65.

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

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

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

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

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

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

S.K. Baruah and S. Vestal. 2008. Schedulability Analysis of Sporadic Tasks with Multiple Criticality Specifications. In *ECRTS*. 147–155.

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

S. K Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. 1996. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica* 15, 6 (1996), 600–625.

S. K. Baruah, L. Cucu-Grosjean, R. I. Davis, and C. Maiza. 2015a. Mixed Criticality on Multicore/Manycore Platforms (Dagstuhl Seminar 15121). *Dagstuhl Reports* 5, 3 (2015), 84–142. DOI:http://dx.doi.org/10.4230/DagRep.5.3.84

I. Bate, A. Burns, and R.I. Davis. 2015. A Bailout Protocol for Mixed Criticality Systems. In *Proc. 27th ECRTS*. 259–268.

I. Bate, A. Burns, and R. I. Davis. 2016. An Enhanced Bailout Protocol for Mixed Criticality Embedded Software. *IEEE Transactions on Software Engineering* PP, 99 (2016).

K.J. Biba. 1977. *Integrity Considerations for Secure Computer Systems*. Mtr-3153. Mitre Corporation.

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

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

M. Bommert. 2013. Schedule-aware Distributed of Parallel Load in a Mixed Criticality Environment. In *Proc. JRWRTC, RTNS*. 21–24.

Bosch. 1991. *CAN Specification version 2.0.* Technical Report. Postfach 30 02 40, D-70442 Stuttgart.

B.B. Brandenburg. 2014. A Synchronous IPC Protocol for Predicatable Access to Shared Resources in Mixed-Criticality Systems. In *Proc. IEEE Real-Time Systems Symposium*. IEEE, 196–206.

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

I. Broster and A. Burns. 2003. An Analysable Bus-Guardian for Event-Triggered Communication. In *Proc. of the 24th Real-time Systems Symposium*. Computer Society, IEEE, Cancun, Mexico, 410–419.

A. Burns. 1994. Preemptive Priority Based Scheduling: An Appropriate Engineering Approach. In *Advances in Real-Time Systems*, S.H. Son (Ed.). Prentice-Hall, 225–248.

A. Burns. 2013. The Application of the Original Priority Ceiling Protocol to Mixed Criticality Systems. In *Proc. ReTiMiCS, RTCSA*, L. George and G. Lipari (Eds.). 7–11.

A. Burns. 2014. System Mode Changes - General and Criticality-Based. In *Proc. 2nd Workshop on Mixed Criticality Systems (WMC), RTSS*, L. Cucu-Grosjean and R. Davis (Eds.). 3–8.

A. Burns. 2015. An Augmented Model for Mixed Criticality. In *Mixed Criticality on Multicore/Manycore Platforms (Dagstuhl Seminar 15121)*, Davis Baruah, Cucu-Grosjean and Maiza (Eds.). Vol. 5(3). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 92–93.

A. Burns and S. Baruah. 2011. Timing Faults and Mixed Criticality Systems. In *Dependable and Historic Computing*, Jones and Lloyd (Eds.). Vol. LNCS 6875. Springer, 147–166.

A. Burns and S. Baruah. 2013. Towards A More Practical Model for Mixed Criticality Systems. In *Proc. 1st Workshop on Mixed Criticality Systems (WMC), RTSS*. 1–6.

A. Burns and S.K. Baruah. 2015. Semi-partitioned Cyclic Executives for Mixed Criticality Systems. In *Proc. 3rd Workshop on Mixed Criticality Systems (WMC), RTSS*, L. Cucu-Grosjean and R. Davis (Eds.). 36–41.

A. Burns and R.I. Davis. 2013. Mixed Criticality on Controller Area Network. In *Proc. Euromicro Conference on Real-Time Systems (ECRTS)*. 125–134.

A. Burns and R.I. Davis. 2014. Adaptive Mixed Criticality Scheduling with Deferred Preemption. In *Proc. IEEE Real-Time Systems Symposium*. IEEE, 21–30.

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

A. Burns, J. Harbin, and L.S. Indrusiak. 2014. A Wormhole NoC Protocol for Mixed Criticality Systems. In *Proc. IEEE Real-Time Systems Symposium*. IEEE, 184–195.

G. Buttazzo, G. Lipari, and L. Abeni. 1998. Elastic Task Model for Adaptive Rate Control. In *IEEE Real-Time Systems Symposium*. 286–295.

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

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. 2013. PROARTIS: Probabilistically Analyzable Real-Time Systems. *ACM Trans. Embedded Comput. Syst.* 12, 2 (2013), 94.

F. Checconi, T. Cucinotta, D. Faggioli, and G. Lipari. 2009. 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)*.

C. Chekuri and S. Khanna. 2004. On Multidimensional Packing Problems. *SIAM J. Comput.* 33, 4 (April 2004), 837–851. DOI:http://dx.doi.org/10.1137/S0097539799356265

Y. Chen, K.G. Shin, and H. Xiong. 2016. Generalizing fixed-priority scheduling for better schedulability in mixed-criticality systems. *Inform. Process. Lett.* 116, 8 (2016), 508–512.

H. Chetto and M. Chetto. 1989. Some Results of the Earliest Deadline Scheduling Algorithm. *IEEE Transactions on Software Engineering* 15, 10 (1989), 1261–1269.

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

M. Chisholm, B. C. Ward, N. Kim, and J. H. Anderson. 2015. Cache Sharing and Isolation Tradeoffs in Multicore Mixed-Criticality Systems. In *2015 IEEE Real-Time Systems Symposium*. 305–316. DOI:http://dx.doi.org/10.1109/RTSS.2015.36

M. Chrisholm, B. Ward, N. Kim, and J. Anderson. 2015. Cache-Sharing and Isolation Tradeoffs in Multicore Mixed-Criticality Systems. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*. 305–316.

B. Cilku, A. Crespo, P. Puschner, J. Coronel, and S. Peiro. 2014. A Memory Arbitration Scheme for Mixed-Criticality Multocore Platforms. In *Proc. 2nd Workshop on Mixed Criticality Systems (WMC), RTSS*, L. Cucu-Grosjean and R. Davis (Eds.). 27–32.

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

B. Cilku and P. Puschner. 2013. Towards Temporal and Spatial Isolation in Memory Hierarchies for Mixed-Criticality Systems with Hypervisors. In *Proc. ReTiMiCS, RTCSA*, L. George and G. Lipari (Eds.). 25–28.

A. Cohen, V. Perrelle, D. Potop-Butucaru, E. Soubiran, and Z. Zhang. 2014. 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 (2014), 138–143.

A. Crespo, A. Alonso, M. Marcos, J.A. Puente, and P. Balbastre. 2014. Mixed Criticality in Control Systems. In *Proc. 19th World Congress The Federation of Automatic Control*. 12261–12271.

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

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

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

L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiones, and F.J. Cazorla. 2012. Measurement-Based Probabilistic Timing Analysis for Multi-path Programs.. In *Proc. 24th Euromicro Conference on Real-Time Systems (ECRTS)*. 91–101.

V. David, A. Barbot, and D. Chabrol. 2014. Dependable Real-Time System and Mixed Criticality: Seeking Safety, Flexibility and Efficiency with Kron-OS. *Ada User Journal* 35, 4 (2014), 259–265.

R.I. Davis. 1995. *On exploiting spare capacity in hard real-time systems.* Ph.D. Dissertation. University of York, UK.

R.I. Davis and M. Bertogna. 2012. Optimal Fixed Priority Scheduling with Deferred Pre-emption. In *Proc. IEEE Real-Time Systems Symposium*. 39–50.

R.I. Davis and A. Burns. 2005. Hierarchical Fixed Priority Preemptive Scheduling. In *Proc. of IEEE Real-Time Systems Symposium (RTSS)*. 389–398.

R.I. Davis and A. Burns. 2006. Resource Sharing in Hierarchical Fixed Priority Preemptive Systems. In *Proceeding IEEE Real-Time Systems Symposium (RTSS)*.

R.I. Davis and A. Burns. 2007. Robust Priority Assignment for Fixed Priority Real-Time Systems. In *Proc. of IEEE Real-Time Systems Symposium (RTSS)*.

R.I. Davis, A. Burns, R.J. Bril, and J.J. Lukkien. 2007. Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised. *Journal of Real-Time Systems* 35, 3 (2007), 239–272.

R.I. Davis, L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean. 2013. Analysis of Probabilistic Cache Related Pre-emption Delays. In *ECRTS*. 129–138.

R.I. Davis, K. Tindell, and A. Burns. 1993. Scheduling Slack Time in Fixed Priority Preemptive Systems. In *Proc. 14th IEEE Real-Time Systems Symposium*.

R.I. Davis and A. J. Wellings. 1995. Dual Priority Scheduling. In *Proc. 16th IEEE Real-Time Systems Symposium*. 100–109.

M. L. Dertouzos. 1974. Control Robotics: The Procedural Control of Physical Processes.. In *IFIP Congress* (2002-01-03). 807–813.

J.L. Díaz, D.F. Garcia, K. Kim, C.G. Lee, L.L. Bello, J.M. López, and O. Mirabella. 2002. Stochastic Analysis of Periodic Real-Time Systems. In *IEEE Real-Time Systems Symposium (RTSS)*.

J. Diemer and R. Ernst. 2010. 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)*. IEEE Computer Society, 155–162.

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

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

S. Draskovic, P. Huang, and L. Thiele. 2016. On the Safety of Mixed-Criticality Scheduling. In *Proc. 4th WMC (RTSS)*. 6.

A. Easwaran. 2013. Demand-based Scheduling of Mixed-Criticality Sporadic Tasks on One Processor. In *Proc. IEEE 34th Real-Time Systems Symposium*. 78–87.

A. Easwaran and I. Shin. 2014. Compositional Mixed-Criticality Scheduling. *CRTS 2014* (2014).

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

S. Edgar and A. Burns. 2001. Statistical Analysis of WCET for Scheduling. In *Proc. 22nd IEEE Real-Time Systems Symposium*.

P. Ekberg, M. Stigge, N. Guan, and W. Yi. 2013. State-Based Mode Switching with Applications to Mixed Criticality Systems. In *Proc. WMC, RTSS*. 61–66.

P. Ekberg and W. Yi. 2012. Bounding and Shaping the Demand of Mixed-Criticality Sporadic Task Systems. In *ECRTS*. 135–144.

P. Ekberg and W. Yi. 2014. Bounding and Shaping the Demand of Generalized Mixed-Criticality Sporadic Task Systems. *Journal of Real-Time Systems* 50 (2014), 48–86.

P. Ekberg and W. Yi. 2015a. A Note on Some Open Problems in Mixed-Criticality Scheduling. In *Proc. RTOPS, 27th ECRTS*. 1–2.

P. Ekberg and W. Yi. 2015b. Schedulability analysis of a graph-based task model for mixed-criticality systems. *Real-Time Systems* (2015), 1–37.

B. Engel. 2016. Tightening Critical Section Bounds in Mixed-Criticality Systems through Preemptible Hardware Transactional Memory. In *Proc. OSPERT*. 17–22.

R. Ernst and M. Di Natale. 2016. Mixed Criticality Systems?A History of Misconceptions? *IEEE Design & Test* 33, 5 (2016), 65–74.

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

C. Evripidou and A. Burns. 2016. Scheduling for Mixed-Criticality Hypervisor Systems in the Automotive Domain. In *Proc. 4th WMC (RTSS)*. 6.

G. Farrall, C. Stellwag, J. Diemer, and R. Ernst. 2013. Hardware and Software Support for Mixed-Criticality Multicore Systems. In *Proceedings of the Conference on Design, Automation and Test in Europe, WICERT (DATE)*.

T. Fleming, S.K. Baruah, and A. Burns. 2016. Improving the Schedulability of Mixed Criticality Cyclic Executives via Limited Task Splitting. In *Proc. of the 24th International Conference RTNS*. ACM, 277–286.

T. Fleming and A. Burns. 2013. Extending Mixed Criticality Scheduling. In *Proc. WMC, RTSS*. 7–12.

T. Fleming and A. Burns. 2014. Incorporating The Notion of Importance into Mixed Criticality Systems. In *Proc. 2nd Workshop on Mixed Criticality Systems (WMC), RTSS*, L. Cucu-Grosjean and R. Davis (Eds.). 33–38.

T. Fleming and A. Burns. 2015. Investigating Mixed Criticality Cyclic Executive Schedule Generation. In *Proc. 3rd Workshop on Mixed Criticality Systems (WMC), RTSS*, L. Cucu-Grosjean and R. Davis (Eds.). 42–47.

T. Fleming and A. Burns. 2016. Utilising Asymmetric Parallelism in Multi-Core MCS Implemented via Cyclic Executives. In *Proc. 4th WMC (RTSS)*. 6.

O. Gettings, S. Quinton, and R.I. Davis. 2015. Mixed Criticality Systems with Weakly-Hard Constraints. In *23rd International Conference on Real-Time Networks and Systems (RTNS 2015)*. 237–246.

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

Georgia Giannopoulou, Peter Poplavko, Dario Socci, Pengcheng Huang, Nikolay Stoimenov, Paraskevas Bourgos, Lothar Thiele, Marius Bozga, Saddek Bensalem, Sylvain Girbal, and others. 2016. *DOL-BIP-Critical: A tool chain for rigorous design and implementation of mixed-criticality multi-core systems*. Technical Report. Technical report 363, ETH Zurich, Laboratory TIK.

G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele. 2013. Scheduling of Mixed-Criticality Applications on Resource-Sharing Multicore Systems. In *Proc. Int. Conference on Embedded Software (EMSOFT)*. Montreal.

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

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

M. Gomony, J. Garside, B. Akesson, N. Audsley, and K. Goossens. 2016. A Globally Arbitrated Memory Tree for Mixed-Time-Criticality Systems. *IEEE Trans. Comput.* (2016).

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. 2013b. Virtual Execution Platforms for Mixed-Time-Criticality Systems: The CompSOC Architecture and Design Flow. *SIGBED Rev.* 10, 3 (2013), 23–34.

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

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

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

R. Gratia, T. Robert, and L. Pautet. 2014. Adaptation of RUN to Mixed-Criticality Systems. In *Proc. 8th Junior Researcher Workshop on Real-Time Computing, RTNS*.

R. Gratia, T. Robert, and L. Pautet. 2015. Generalized Mixed-Criticality Scheduling based on RUN. In *23rd International Conference on Real-Time Networks and Systems (RTNS 2015)*. 267–276.

P. Graydon and I. Bate. 2013. Safety Assurance Driven Problem Formulation for Mixed-Criticality Scheduling. In *Proc. WMC, RTSS*. 19–24.

D. Griffin, I. Bate, B. Lesage, and F. Soboczenski. 2015. Evaluating Mixed Criticality Scheduling Algorithms with Realistic Workloads. In *Proc. 3rd Workshop on Mixed Criticality Systems (WMC), RTSS*, L. Cucu-Grosjean and R. Davis (Eds.). 24–29.

S. Groesbrink, L. Almeida, M. de Sousa, and S.M. Petters. 2014. Towards Certifiable Adaptive Reservations for Hypervisor-based Virtualization. In *Proc. of the 20th Real-Time and Embedded Technology and Applications Symposium (RTAS)*.

S. Groesbrink, S. Oberthr, and D. Baldin. 2013. 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)*, Vol. 10(1). ACM SIGBED Review.

K. Grüttner. 2017. 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*. Springer, 57–90.

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

C. Gu, N. Guan, Q. Deng1, and W. Yi. 2013. Improving OCBP-based Scheduling for Mixed-Criticality Sporadic Task Systems. In *Proc. RTCSA*.

X. Gu and A. Easwaran. 2014. Optimal Speedup Bound for 2-Level Mixed-Criticality Arbitrary Deadline Systems. In *Proceedings RTSOPS (ECRTS)*. 15–16.

X. Gu and A. Easwaran. 2016. Dynamic Budget Management with Service Guarantees for Mixed-Criticality Systems. In *Proc. Real-Time Systems Symposium (RTSS)*. IEEE, 47–56.

X. Gu, K.-M. Phan, A. Easwaran, and I. Shin. 2015. Resource Efficient Isolation Mechanisms in Mixed-Criticality Scheduling. In *Proc. 27th ECRTS*. IEEE, 13–24.

N. Guan, P. Ekberg, M. Stigge, and W. Yi. 2011. Effective and Efficient Scheduling of Certifiable Mixed-Criticality Sporadic Task Systems. In *IEEE RTSS*. 13–23.

Z. Guo. 2016. Mixed-Criticality Scheduling on Varying-Speed Platforms with Bounded Performance Drop Rate. In *Proc SMARTCOMP*. IEEE, 1–3.

Z. Guo and S. Baruah. 2014. Implementing Mixed-criticality Systems Upon a Preemptive Varying-speed Processor. *Leibniz Transactions on Embedded Systems* 1, 2 (2014), 03–103:19.

Z. Guo and S.K. Baruah. 2015. 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)*. 247–256.

Z. Guo, L. Santinelli, and K. Yang. 2015. EDF Schedulability Analysis on Mixed-Criticality Systems with Permitted Failure Probability. In *Proc. RTCSA*.

H. Hamza, A. Hughes, and R. Kirner. 2015. On the Design of a Java Virtual Machine for Mixed-Criticality Systems. In *Proc. JTRES*. ACM.

J.J. Han, X. Tao, D. Zhu, and H. Aydin. 2016. Criticality-Aware Partitioning for Multicore Mixed-Criticality Systems. In *Proc. Parallel Processing (ICPP)*. IEEE, 227–235.

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

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

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

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

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

C. Herber, A. Richter, H. Rauchfuss, and A. Herkersdorf. 2013. Spatial and Temporal Isolation of Virtual CAN Controllers. In *Proc. VtRES, RTCSA*.

J. Herman, C. Kenna, M. Mollison, J. Anderson, and D. Johnson. 2012. RTOS Support for Multicore Mixed-Criticality Systems. In *Proc. of the 18th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*.

F. Herrera, S.H.A. Niaki, and I. Sander. 2013. Towards a Modelling and Design Framework for Mixed-Criticality SoCs and Systems-of-Systems. In *Proc. 16th Euromicro Conf. on Digital Systems Design*. 989–996.

F. Herrera, P. Penil, and E. Villar. 2015. A Model-based, Single-Source approach to Design-Space Exploration and Synthesis of Mixed-Criticality Systems. In *Proc. SCOPES*. 88–91.

M. Hikmet, M.M. Kuo, P.S. Roop, and P. Ranjitkar. 2016. Mixed-Criticality Systems as a Service for Non-Critical Tasks. In *Proc. ISORC*. 221–228.

M.G. Hill and T.W. Lake. 2000. Non-Interference Analysis for Mixed Criticality Code in Avionics Systems. In *Proceedings of the 15th IEEE international conference on Automated software engineering*. IEEE Computer Society, 257–260.

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

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

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

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

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

H-M. Huang, C. Gill, and C. Lu. 2012. Implementation and Evaluation of Mixed Criticality Scheduling Approaches for Periodic Tasks. In *Proc. of the IEEE Real-Time Technology and Applications Symposium (RTAS)*. 23–32.

H-M. Huang, C. Gill, and C. Lu. 2014a. Implementation and Evaluation of Mixed Criticality Scheduling Approaches for Sporadic Tasks. *ACM Trans. Embedded Systems* 13 (2014), 126:1– 126:25.

P. Huang, G. Giannopoulou, R. Ahmed, D.B. Bartolini, and L. Thiele. 2015. An Isolation Scheduling Model for Multicores. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*. 141–152.

P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele. 2013. *Service Adaptions for Mixed-Criticality Systems*. Technical Report 350. ETH Zurich, Laboratory TIK.

P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele. 2014. Service Adaptions for Mixed-Criticality Systems. In *19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Singapore.

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

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

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

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

L.S. Indrusiak, J. Harbin, and A. Burns. 2015. Average and Worst-Case Latency Improvements in Mixed-Criticality Wormhole Networks-on-Chip. In *Proc. 27th ECRTS*. IEEE, 47–56.

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

P. Ittershagen, K. Gruttner, and W. Nebel. 2015. Mixed-criticality system modelling with dynamic execution mode switching. In *2015 Forum on Specification and Design Languages (FDL)*. 1–6.

V. Izosimov and E. Levholt. 2015. Mixed Criticality metric for safety-critical Cyber-Physical systems on multicore archiectures. *MEDIAN: Methods* 2, 8 (2015).

J. Jalle, E. Quinones, J. Abella, L. Fossati, M. Zulianello, and F.J. Cazorla. 2014. A Dual-Criticality Memory Controler (DCmc): Proposal and Evaluation of a Space Case Study. In *Proc. IEEE Real-Time Systems Symposium*. IEEE, 207–217.

M. Jan, L. Zaourar, V. Legout, and L. Pautet. 2014. 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 (2014), 138–143.

M. Jan, L. Zaourar, and M. Pitel. 2013. Maximizing the Execution Rate of Low Criticality Tasks in Mixed Criticality System. In *Proc. 1st WMC, RTSS*. 43–48.

X. Jin, J. Wang, and P. Zeng. 2015. End-to-end delay analysis for mixed-criticality WirelessHART networks. *Automatica Sinica, IEEE/CAA Journal of* 2, 3 (2015), 282–289.

X. Jin, C. Xia, H. Xu, J. Wang, and P. Zeng. 2016. Mixed Criticality Scheduling for Industrial Wireless Sensor Networks. *Sensors* 16, 9 (2016), 1376.

C.B. Jones. 1983. Tentative Steps Toward a Development Method for Interfering Programs. *Transactions on Programming Languages and System* 5, 4 (1983), 596–619.

M. Joseph and P. Pandya. 1986. Finding Response Times in a Real-Time System. *BCS Computer Journal* 29, 5 (1986), 390–395.

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

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

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

O.R. Kelly, H. Aydin, and B. Zhao. 2011. On Partitioned Scheduling of Fixed-Priority Mixed-Criticality Task Sets. In *IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*. 1051–1059.

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

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

Young-Seung Kim and Hyun-Wook Jin. 2014. Towards a practical implementation of criticality mode change in RTOS. In *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*. 1–4. DOI:http://dx.doi.org/10.1109/ETFA.2014.7005318

A. Kostrzewa, S. Saidi, and R. Ernst. 2015. Dynamic Control for Mixed-Criticality Networks-on-Chip. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*. 317–326.

O. Kotaba, J. Nowotschy, M. Paulitschy, S.M. Petters, and H. Theiling. 2013. 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)*.

A. Kritikakou, O. Baldellon, C. Pagetti, C. Rochange, M. Roy, and F. Vargas. 2013. MONITORING ON-LINE TIMING INFORMATION TO SUPPORT MIXED-CRITICAL WORKLOADS. In *WiP, RTSS*. 9–10.

A. Kritikakou, C. Pagetti, O. Baldellon, M. Roy, and C. Rochange. 2014a. Run-time Control to Increase Task Parallelism in Mixed-Critical Systems. In *ECRTS*. 119–128.

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

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

A. Lackorzynski, A. Warg, M. Voelp, and H. Haertig. 2012. Flattening Hierarchical Scheduling. In *Proc. ACM EMSOFT*. 93–102.

K. Lakshmanan, D. de Niz, and R. Rajkumar. 2011. Mixed-Criticality Task Synchronization in Zero-Slack Scheduling. In *IEEE RTAS*. 47–56.

K. Lakshmanan, D. de Niz, R. Rajkumar, and G. Moreno. 2010. Resource Allocation in Distributed Mixed-Criticality Cyber-Physical Systems. In *ICDCS*. 169–178.

A. Larrucea, I. Martinez, V. Brocal, S. Peirò, H. Ahmadian, J. Perez, and R. Obermaisser. 2016. DREAMS: Cross-Domain Mixed-Criticality Patterns. In *Proc. 4th WMC (RTSS)*. 6.

J. Lee, H.S. Chwa, A. Easwaran, I. Shin, and I. Lee. 2015. Towards Compositional Mixed-Criticality Real-Time scheduling in Open Systems. In *Proc. 8th Workshop on Compositional Real-Time Systems (CRTS), RTSS*, L. Almeida and D. de Niz (Eds.).

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

J. Lee, K-M. P, Z Gu, J. Lee, A. Easwaran, I. Shin, and I. Lee. 2014. MC-Fluid: Fluid Model-based Mixed-Criticality Scheduling on Multiprocessors. In *Proc. IEEE Real-Time Systems Symposium*. IEEE, 41–52.

V. Legout, M. Jan, and L. Pautet. 2013. Mixed-Criticality Multiprocessor Real-Time Systems: Energy Consumption vs Deadline Misses. In *Proc. ReTiMiCS, RTCSA*, L. George and G. Lipari (Eds.). 1–6.

J.P. Lehoczky and S. Ramos-Thuel. 1992. An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks Fixed-Priority Preemptive systems. In *Proc. 13th IEEE Real-Time Systems Symposium*. 110–123.

J.P. Lehoczky, L. Sha, and J.K. Strosnider. 1987. Enhanced Aperiodic Responsiveness in a Hard Real-Time Environment. In *Proc. 8th IEEE Real-Time Systems Symposium*. 261–270.

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

J. Y-T. Leung and J. Whitehead. 1982. On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks. *Performance Evaluation (Netherlands)* 2, 4 (Dec. 1982), 237–250.

H. Li. 2013. *Scheduling Mixed-Criticality Real-Time Systems*. Ph.D. Dissertation. The University of North Carolina at Chapel Hill.

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

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

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

J. Li, D. Ferry, S. Ahuja, K. Agrawal, C. Gill, and C. Lu. 2016. Mixed-Criticality Federated Scheduling for Parallel Real-Time Tasks. In *Proc. RTAS*. IEEE, 1–12.

Y. Li, R. West, and E. Missimer. 2013. The Quest-V Separation Kernel for Mixed Criticality Systems. In *Proc. 1st WMC, RTSS*. 31–36.

Z. Li and L. Wang. 2016. Memory-Aware Scheduling for Mixed-Criticality Systems. In *IProc ICCSA*. Springer, LNCS 9787, 140–156.

J. Lin, A.M.K. Cheng, D. Steel, and M.Y.-C. Wu. 2014. Scheduling Mixed-Criticality Real-Time Tasks with Fault Tolerance. In *Proc. 2nd Workshop on Mixed Criticality Systems (WMC), RTSS*, L. Cucu-Grosjean and R. Davis (Eds.). 39–44.

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

G. Lipari and G. Buttazzo. 2013. Resource Reservation for Mixed Criticality Systems. In *Proceeding of Workshop on Real-Time Systems: the past, the present, and the future*. York, UK, 60–74.

C.L. Liu and J.W. Layland. 1973. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *JACM* 20, 1 (1973), 46–61.

G. Liu, Y. Lu, S. Wang, and Z. Gu. 2014. Partitioned Multiprocessor Scheduling of Mixed-Criticality Parallel Jobs. In *Proc. Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE.

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

A. Lyons and G. Heiser. 2014. Mixed-Criticality Support in a High-Assurance, General-Purpose Microkernel. In *Proc. 2nd Workshop on Mixed Criticality Systems (WMC), RTSS*, L. Cucu-Grosjean and R. Davis (Eds.). 9–14.

M. Mahdiani and A. Masrur. 2016. Introducing Utilization Caps into Mixed-Criticality Scheduling. In *Proc. Digital System Design (DSD)*. IEEE, 388–395.

S.O. Marinescu, D. Tamas-Selicean, V. Acretoaie, and P. Pop. 2012. 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*.

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

A. Masrur, D. Muller, and M. Werner. 2015. Bi-Level Deadline Scaling for Admission Control in Mixed-Criticality Systems. In *Proc. 21st IEEE Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 100–109.

S. Maurer and R. Kirner. 2015. Cross-criticality interfaces for cyber-physical systems. In *2015 International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*. 1–8. DOI:http://dx.doi.org/10.1109/EBCCSP.2015.7300670

D. Maxim and L. Cucu-Grosjean. 2013. Response time analysis for fixed-priority tasks with multiple probabilistic parameters. In *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*. IEEE, 224–235.

D. Maxim, R.I. Davis, L. Cucu-Grosjean, and A. Easwaran. 2016. Probabilistic Analysis for Mixed Criticality Scheduling with SMC and AMC. In *Proc. 4th WMC (RTSS)*. 6.

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

E. Missimer, K. Missimer, and R. West. 2016. Mixed-Criticality Scheduling with I/O. In *Proc. ECRTS*. 120–130.

M. Mollison, J. Erickson, J. Anderson, S.K. Baruah, and J. Scoredos. 2010. Mixed Criticality Real-Time Scheduling for Multicore Systems. In *Proc. of the 7th IEEE International Conference on Embedded Software and Systems*. 1864–1871.

B. Motruk, J. Diemer, R. Buchty, R. Ernst, and M. Berekovic. 2012. IDAMC: A Many-Core Platform with Run-Time Monitoring for Mixed-Criticality. *Ninth IEEE International Symposium on High-Assurance Systems Engineering (HASE'05)* (2012), 24–31.

D. Muller and A. Masrur. 2014. 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)*. 978–981.

K. Napier, O. Horst, and C. Prehofer. 2016. Comparably Evaluating Communication Performance within Mixed-Criticality Systems. In *Proc. 4th WMC (RTSS)*. 6.

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

M. Neukirchner, P. Axer, T. Michaels, and R. Ernst. 2013a. Monitoring of Workload Arrival Functions for Mixed-Criticality Systems. In *Proc. IEEE 34th Real-Time Systems Symposium*. 88–96.

M. Neukirchner, S. Quinton, and K. Lampka. 2013b. Multi-Mode Monitoring for Mixed-Criticality Real-time Systems. In *Int'l Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*.

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

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

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

D.de Niz, K. Lakshmanan, and R. Rajkumar. 2009. On the Scheduling of Mixed-Criticality Real-Time Task Sets. In *Real-Time Systems Symposium*. IEEE Computer Society, 291–300.

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

D.de Niz, L. Wrage, A. Rowe, and R. Rajkumar. 2013. Utility-Based Resource Overbooking For Cyber-Physical Systems. In *Proc. RTCSA*.

A. Novak, P. Sucha, and Z. Hanzalek. 2016a. Efficient Algorithm for Jitter Minimization in Time-Triggered Periodic Mixed-Criticality Message Scheduling Problem. In *Proc. RTNS*. ACM, 23–31.

A. Novak, P. Sucha, and Z. Hanzalek. 2016b. On Solving Non-preemptive Mixed-criticality Match-up Scheduling Problem with Two and Three Criticality Levels. *arXiv preprint arXiv:1610.07384* (2016).

J. Nowotsch, M. Paulitsch, D. Bhler, H. Theiling, S. Wegener, and M. Schmidt. 2014. Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement. In *ECRTS*. 109–118.

R. Obermaisser, Z. Owda, M. Abuteir, H. Ahmadian, and D. Wber. 2014. End-to-end Real-Time COmmunication in Mixed-Criticality Systems based on Netowrked Multicore Chips. In *Proc 17th Euromicor Conference on Digital Systems Design*. IEEE, 293–302.

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

S. Osmolovskiy, I. Fedorov, V. Vinogradov, E. Ivanova, and D. Shakurov. 2016. 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)*. 257–265.

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

T. Park and S Kim. 2011. Dynamic Scheduling Algorithm and its Schedulability Analysis for Certifiable Dual-Criticality Systems. In *Proc. ACM EMSOFT*. 253–262.

R.M. Pathan. 2012. Schedulability analysis of Mixed Criticality Systems on Multiprocessors. In *Proc. of ECRTS, Pisa*. 309–320.

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

M. Paulitsch, O.M. Duarte, H. Karray, K. Mueller, D. Muench, and J. Nowotsch. 2015. Mixed-Criticality Embedded Systems–A Balance Ensuring Partitioning and Performance. In *Euromicro Conference on Digital System Design (DSD)*. IEEE, 453–461.

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

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

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

J. Perez, D. Gonzalez, S. Trujillo, T. Trapman, and J. M. Garate. 2013. A Safety Concept for a Wind Power Mixed Criticality Embedded System Based on Multicore Partitioning. In *Proc. 1st WMC, RTSS*. 25–30.

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

P. Pop, L. Tsiopoulos, S. Voss, O. Slotosch, C. Ficek, U. Nyman, and A. Ruiz. 2013. 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)*.

T. Pop, P. Eles, and Z. Peng. 2002. 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)*. ACM, 187–192.

S. Punnekkat, R.I Davis, and A. Burns. 1997. Sensitivity Analysis of Real-Time Task Sets. In *Proc. of the Conference of Advances in Computing Science - ASIAN '97*. Springer, 72–82.

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

S. Ramanathan, X. Gu, and A. Easwaran. 2016. The Feasibility Analysis of Mixed-Criticality Systems. In *Proc. RTOPS, ECRTS*.

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

J. Ren and L.T.X. Phan. 2015. Mixed-Criticality Scheduling on Multiprocessors using Task Grouping. In *Proc. 27th ECRTS*. IEEE, 25–36.

P. Rodriguez, L. George, Y. Abdeddaim, and J. Goossens. 2013. Multi-Criteria Evaluation of Partitioned EDF-VD for Mixed Criticality Systems Upon Identical Processors. In *Proc. 1st WMC, RTSS*. 49–54.

S. Saewong, R. Rajkumar, J.P. Lehoczky, and M.H. Klein. 2002. Analysis of Hierarchical Fixed- Priority Scheduling. In *Proc. of the 14th Euromicro Conference on Real-Time Systems (ECRTS)*. 173–181.

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

M. Saksena and Y. Wang. 2000. Scaleable Real-Time Systems Design Using Preemption Thresholds. In *Proceeding 21st IEEE Real-Time Systems Symposium.* 25–34.

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

L. Santinelli and L. George. 2015. Probabilities and Mixed-Criticalities: the Probabilistic C-Space. In *Proc. 3rd Workshop on Mixed Criticality Systems (WMC), RTSS*, L. Cucu-Grosjean and R. Davis (Eds.). 30–35.

R. Santos, S. Venkataraman, A. Das, and A. Kumar. 2014. Criticality-aware scrubbing mechanism for SRAM-based FPGAs. In *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*. 1–8. DOI: http://dx.doi.org/10.1109/FPL.2014.6927476

J. A. Santos-Jr, G. Lima, and K. Bletsas. 2015. Considerations on the Least Upper Bound for Mixed-Criticality Real-Time Systems. In *5th Brazilian Symposium on Computing Systems Engineering (SBESC)*.

F. Santy, L. George, P. Thierry, and J. Goossens. 2012. Relaxing Mixed-Criticality Scheduling Strictness for Task Sets Scheduled with FP. In *Proc. of the Euromicro Conference on Real-Time Systems*. 155–165.

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

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

L. Sha. 2009. Resilient Mixed Criticality Systems. *CrossTalk The Journal of Defense Software Engineering* (October 2009), 9–14.

L. Sha, J.P. Lehoczky, and R. Rajkumar. 1986. Solutions For Some Practical Problems in Prioritizing Preemptive Scheduling. In *Proc. 7th IEEE Real-Time Sytems Symposium*.

L. Sha, J.P. Lehoczky, and R. Rajkumar. 1987. Task Scheduling in Distributed Real-Time Systems. In *Proc. of IEEE Industrial Electronics Conference*.

L. Sha, R. Rajkumar, and J.P. Lehoczky. 1990. Priority Inheritance Protocols: An Approach to Real-Time Synchronisation. *IEEE Trans. Comput.* 39, 9 (1990), 1175–1185.

Z. Shi and A. Burns. 2008. 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)*. 161–170.

Insik Shin and Insup Lee. 2003. Periodic resource model for compositional real-time guarantees. In *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*. 2–13. DOI:http://dx.doi.org/10.1109/REAL.2003.1253249

L. Sigrist, G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele. 2014. Mapping Mixed-Criticality Applications on multi-core architectures. In *Proc. DATE*. 1–6.

D. Socci, P. Poplavko, S. Bensalem, and M. Bozga. 2013a. Mixed Critical Earliest Deadline First. In *Proc. Euromicro Conference on Real-Time Systems (ECRTS)*.

D. Socci, P. Poplavko, S. Bensalem, and M. Bozga. 2013b. Time-Triggered Mixed Critical Scheduler. In *Proc. WMC, RTSS*. 67–72.

D. Socci, P. Poplavko, S. Bensalem, and M. Bozga. 2015a. Multiprocessor Scheduling of Precedence-constrained Mixed-Critical Jobs. In *2015 IEEE 18th International Symposium on Real-Time Distributed Computing*. 198–207. DOI:http://dx.doi.org/10.1109/ISORC.2015.18

D. Socci, P. Poplavko, S. Bensalem, and M. Bozga. 2015b. *Time-Triggered Mixed-Critical Scheduler on Single- and Multi-processor Platforms*. Technical Report TR-2015-8. Verimag.

B. Sprunt, J. Lehoczky, and L. Sha. 1988. Exploiting Unused Periodic Time For Aperiodic Service Using the Extended Priority Exchange Algorithm. In *Proc. 9th IEEE Real-Time Systems Symposium*. 251–258.

W. Steiner. 2011. Synthesis of Static Communication Schedules for Mixed-Criticality Systems. *2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops* (2011), 11–18.

H. Su, P. Deng, D. Zhu, and Q. Zhu. 2016a. Fixed-Priority Dual-Rate Mixed-Criticality Systems: Schedulability Analysis and Performance Optimization. In *Proc. Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 59–68.

H. Su, P. Deng, D. Zhu, and Q. Zhu. 2016b. 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)*. 59–68.

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

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

H. Su, D. Zhu, and D. Mosse. 2013. Scheduling Algorithms for Elastic Mixed-Criticality Tasks in Multicore Systems. In *Proc. RTCSA*.

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

D. Tamas-Selicean and P. Pop. 2011b. 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*. 2–10.

D. Tamas-Selicean and P. Pop. 2011c. Task Mapping and Partition Allocation for mixed criticality real-time systems. In *IEEE Pacific Rim Int. Sym. on Dependable Computing*. 282–283.

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

J. Theis and G. Fohler. 2013. Mixed Criticality Scheduling in Time-Triggered Legacy Systems. In *Proc. WMC, RTSS*. 73–78.

J. Theis, G. Fohler, and S. Baruah. 2013. Schedule Table Generation of Time-Triggered Mixed Criticality Systems. In *Proc. WMC, RTSS*. 79–84.

A. Thekkilakattil, R. Dobrin, and S. Punnekkat. 2014a. 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.

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

A Thekkilakattl, A. Burns, R. Dobrin, and S. Punnekkat. 2015. Mixed Criticality Systems: Beyond Transient Faults. In *Proc. 3rd Workshop on Mixed Criticality Systems (WMC), RTSS*, L. Cucu-Grosjean and R. Davis (Eds.). 18–23.

H. Thompson. 2012. *Mixed Criticality Systems*. http://cordis.europa.eu/fp7/ict/embedded-systems-engineering/documents/sra-mixed-criticality-systems.pdf. EU, ICT.

K. Tindell and A Alonso. 1996. *A very simple protocol for mode changes in priority preemptive systems*. Technical Report. Universidad Politecnica de Madrid.

K. Tindell, A. Burns, and A. J. Wellings. 1992. Mode Changes in Priority Preemptive Scheduled Systems. In *Proc. Real Time Systems Symposium*. Phoenix, Arizona, 100–109.

S. Tobuschat, P. Axer, R. Ernst, and J. Diemer. 2013. IDAMC: A NoC for Mixed Criticality Systems. In *Proc. RTCSA*.

S. Trujillo, A. Crespo, and A. Alonso. 2013. MultiPARTES: Multicore Virtualization for Mixed-Criticality Systems. In *Digital System Design (DSD), 2013 Euromicro Conference on*. 260–265.

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

S. Vestal. 2007. Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*. 239–243.

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

M. Volp, A. Lackorzynski, and H. Hartig. 2013. On the Expressiveness of Fixed Priority Scheduling Contexts for Mixed Criticality Scheduling. In *Proc. WMC, RTSS*. 13–18.

M. Völp, M. Roitzsch, and H. Härtig. 2015. Towards an Interpretation of Mixed Criticality for Optimistic Scheduling. In *21st IEEE RTAS, Work-in-Progress*. 15–16.

G. von der Brüggen, K-H. Chen, W-H. Huang, and J-J. Chen. 2016. Systems with Dynamic Real-Time Guarantees in Uncertain and Faulty Execution Environments. In *Proc. Real-Time Systems Symposium (RTSS)*. IEEE, 303–314.

P. Wagemann, T. Distler, H. Janker, P. Raffeck, and V. Sieh. 2016. A Kernel for Energy-Neutral Real-Time Systems with Mixed Criticalities. In *Proc. RTAS*. IEEE, 1–12.

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

A. Wasicek, C. El-Salloum, and H. Kopetz. 2010. 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)*. 210–216.

R. West, Y. Li, E. Missimer, and M. Danish. 2016. A Virtualized Separation Kernel for Mixed-Criticality Systems. *ACM Transactions on Computer Systems (TOCS)* 34, 3 (2016), 8.

H. Xu and A. Burns. 2015. Semi-partitioned Model for Dual-core Mixed Criticality System. In *23rd International Conference on Real-Time Networks and Systems (RTNS 2015)*. 257–266.

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

E. Yip, M.M.Y Kuo, D. Broman, and P.S Roop. 2014. Relaxing the Synchronous Approach for Mixed-Criticality Systems. In *Proc. Real-Time and Embedded Technology and Application Symposium (RTAS)*. IEEE, 89–100.

H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. 2012. Memory Access Control in Multiproccessor for real-time mixed criticality. In *Proc. of ECRTS, Pisa*. 299–308.

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

F. Zhang and A. Burns. 2007. Analysis of Hierarchical EDF Preemptive Scheduling. In *Proc. of IEEE Real-Time Systems Symposium (RTSS)*. 423–435.

F. Zhang and A. Burns. 2008. Schedulability Analysis for Real-Time Systems with EDF Scheduling. *IEEE Transaction on Computers* 58, 9 (2008), 1250–1258.

N. Zhang, C. Xu, J. Li, and M. Peng. 2015. A Sufficient Response-time Analysis for Mixed Criticality Systems with Pessimistic Period. *Journal of Computational Information Systems* 11, 6 (2015), 1955–1964.

X. Zhang, J. Zhan, W. Jiang, Y. Ma, and K. Jiang. 2013. Design Optimization of Security-Sensitive Mixed-Criticality Real-Time Embedded Systems. In *Proc. ReTiMiCS, RTCSA*, L. George and G. Lipari (Eds.). 12–17.

Q. Zhao, Z. Gu, and H. Zeng. 2013a. Integration of Resource Synchronization and Preemption-Thresholds into EDF-Based Mixed-Criticality Scheduling Algorithm. In *Proc. RTCSA*.

Q. Zhao, Z. Gu, and H. Zeng. 2013b. PT-AMC: Integrating Preemption Thresholds into Mixed-Criticality Scheduling. In *Proc. DATE*. 141–146.

Q. Zhao, Z. Gu, and H. Zeng. 2014. HLC-PCP: A Resource Synchronization Protocol for Certifiable Mixed Criticality Scheduling. *Embedded Systems Letters, IEEE* 6, 1 (2014).

Q. Zhao, Z. Gu, and H. Zeng. 2015. Resource Synchronization and Preemption Thresholds Within Mixed-Criticality Scheduling. *ACM Transactions on Embedded Computing Systems (TECS)* 14, 4 (2015), 81.

L. Ziarek and E. Blanton. 2015. The Fiji MultiVM Archiecture. In *Proc. JTRES*. ACM.

M. Zimmer, D.Broman, C. Shaver, and E.A. Lee. 2014. FlexPRET: A Processor Platform for Mixed-Criticality Systems. In *Proc. RTAS*. 101–110.