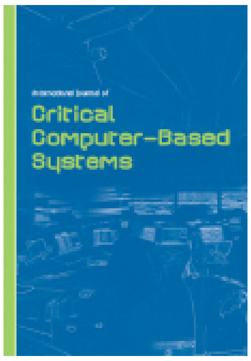


International Journal of Critical Computer-Based Systems

ISSN online: 1757-8787 - ISSN print: 1757-8779
https://www.inderscience.com/ijccbs

Task models for mixed criticality systems - a review

Louella Colaco, Arun S. Nair, Biju K. Raveendran, Sasikumar Punnekkat

DOI: 10.1504/IJCCBS.2022.10053341

Article History:

Received:	29 July 2022
Accepted:	21 October 2022
Published online:	30 January 2024

Task models for mixed criticality systems – a review

Louella Colaco*, Arun S. Nair and
Biju K. Raveendran

Department of Computer Science and Information Systems,
BITS Pilani K.K. Birla Goa Campus,
Sancoale, Goa, India
Email: p20150009@goa.bits-pilani.ac.in
Email: p20150024@goa.bits-pilani.ac.in
Email: biju@goa.bits-pilani.ac.in
*Corresponding author

Sasikumar Punnekkat

Dependable Software Engineering,
Mälardalen University,
Västerås, Sweden
Email: sasikumar.punnekkat@mdu.se

Abstract: The past decade has seen tremendous interest in mixed criticality systems research due to its exponential growth with inherent challenges of effective resource utilisation and isolation. The pervasiveness of these systems along with their certification needs, prompt for suitable task models to perform the required analysis. Extensive usage scenarios and strict certification requirements have spawned a broad spectrum of research and evolved into several task models. In this work, a thematic survey of task models for both uni-core and multi-core mixed criticality systems is carried out. The work categorises task models based on attributes such as resources, quality of service, operating system overheads, energy, fault tolerance and parallel processing. After synthesising the state-of-the-art, the work summarises task models by providing a visual aid and a ready reckoner with traceability to mixed criticality challenges. This work serves as a quintessential reference manual for researchers and academicians in the mixed criticality domain.

Keywords: uni-core mixed criticality systems; multi-core mixed criticality systems; task models for mixed criticality systems.

Reference to this paper should be made as follows: Colaco, L., Nair, A.S., Raveendran, B.K. and Punnekkat, S. (2023) 'Task models for mixed criticality systems – a review', *Int. J. Critical Computer-Based Systems*, Vol. 10, No. 4, pp.287–329.

Biographical notes: Louella Colaco is currently pursuing her PhD at the BITS Pilani K.K. Birla Goa Campus. She is also an Assistant Professor in the Department of Computer Engineering at Padre Conceicao College of

Engineering, Verna, Goa. She has 17 years of teaching experience at UG and PG level and has guided several graduate and post graduate dissertations. Her research interest include operating systems, real-time systems and mixed criticality systems. Her main areas of research are resource synchronisation and energy efficiency in mixed criticality systems.

Arun S. Nair is working as a divisional manager (Architect at Nexteer Automotive India Software Center) and also pursuing his PhD at the BITS Pilani K.K. Birla Goa Campus. He is a post-graduate in Electronics and has 25 years of industry experience in space and automotive electronics. He is an author of research studies published in national/international journals and conference proceedings. His research interest includes mixed-criticality systems, multi-core controller, automotive functional safety ISO26262 and automotive applications.

Biju K. Raveendran is currently serving as an Associate Professor in the Department of Computer Science and Information Systems, BITS Pilani K.K. Birla Goa Campus, Goa, India. He received his PhD from the BITS Pilani, Pilani Campus, Rajasthan in 2009. His research area includes energy efficient multi-core/many-core real-time scheduling, energy efficient memory architecture for multi-core/many-core embedded systems, predictable and dependable real-time/embedded system design, big data systems, etc. He was one of the five recipients of Microsoft Research India Fellowship in 2005 for his PhD work. He is a recipient of Microsoft Young Faculty Award in 2009. He is actively involved in collaborative projects with industries like Microsoft, Aditya Birla Group, etc.

Sasikumar Punnekkat is a Professor of Dependable Software Engineering at Mälardalen University, Sweden and a leader of the Dependable Software Engineering Research Group. He has more than 15 years industrial experience as a Scientist at the Indian Space research Organization, and was the Head of the Software Test and Reliability Engineering. He was recipient of the Prestigious Commonwealth Scholarship and was awarded Doctor of Philosophy in Computer Science by the University of York, UK in 1997 for his research on schedulability analysis of fault-tolerant systems. He was the Director of BITS Pilani K.K. Birla Goa Campus during March 2015–August 2016. His research interests include multiple aspects of real-time systems, dependability, and software engineering. He has over 140 research publications in international conferences and journals. He is a senior member of IEEE and had a lead role in many EU funded research and collaboration projects.

1 Introduction

The induction of mixed criticality systems (MCS) to industry is dawdling due to deep-rooted certifiability issues, which in turn triggers inefficient usage of resources and under utilisation of computing power. In MCS, a common platform is assigned to functionalities with different degrees of significance. Industry wide initiatives such as Automotive Open System Architecture (AUTOSAR) and Integrated Modular Avionics

(IMA) focus on the co-existence of MCS (Guo and Baruah, 2016). The introduction of multi-core systems aid in increasing performance and lowering energy consumption. It also promises isolation and fault tolerance of safety critical and mission critical systems. However, the phenomenal increase in MCS and their certification/implementation needs, comes with its own challenges. Design of new technologies, tools and models are required to meet the domain needs of modern MCS, which has to take care of size, weight, power and certification of these complex systems. The mixed criticality task models should accommodate complex scheduling schemes, effective usage of resources and system mode changes along with certification needs of safety critical systems.

The Burns and Davis (2017) survey provides an elaborate description of MCS that includes uniprocessor/multi-processor scheduling, shared resources, task allocation, realistic models, formal treatments, system issues, industrial practices and related topics. However, this paper delves into details on the current state-of-the-art task models for uni-core and multi-core MCS with emphasis on attributes such as resources, quality of service (QoS), operating system (OS) overheads, energy, fault tolerance and parallel processing. It provides a detailed survey of the existing task models and serves as a quintessential reference manual for the research and industry fraternity in the mixed criticality domain.

This work is ordered along the following lines: Section 2 provides an overview of MCS and includes MCS challenges. Section 3 analyses mixed criticality models for uni-core systems. Section 4 extends to include multi-core systems. Section 5 provides an unified view of the various task models surveyed in this work, and Section 6 concludes the work with future directions.

2 Mixed criticality systems

Real-time systems (RTS) are time-constrained systems having jobs with deadlines to meet (Liu and Layland, 1973). They can be classified as hard, weakly hard, soft, firm or hybrid systems based on requirements and timeliness guarantees. In RTS, a job is an active entity and a set of related jobs form a task. Tasks can be periodic, aperiodic or sporadic in nature. A periodic task is defined by four-tuple with elements $\{\phi, T, D, C\}$, denoted by phase, period, relative deadline and worst case execution time (WCET). Instead of period, a minimum inter arrival time between adjacent jobs of a task is used in sporadic tasks. Aperiodic jobs are usually soft deadline or no deadline jobs with arbitrary arrival times.

There exist RTS with varying significance among tasks. Significance or criticality is not the same as priority. Priority refers to precedence or ordering among tasks/jobs whereas criticality refers to importance among tasks/jobs. Burns and Davis (2013a) defined criticality as "the designation of the level of assurance against failure needed for a system component." For example, in avionics, the flight cockpit system is more critical than an in-flight entertainment system. MCS execute tasks of various criticality levels on a single computing platform. These tasks are characterised as safety, mission and non-critical according to the criticality impact on the system. The basic task model of MCS is the same as RTS with addition of criticality as a parameter. A task τ in MCS is usually defined by five-tuple with elements $\{\phi, T, D, L, \overrightarrow{C}\}$ denoted by phase, period, relative deadline, criticality level and WCET. The WCET is a vector whose size is defined by the number of criticality levels. A task's computational requirements are mapped to the differing behaviour of criticalities. This is achieved either by increasing the WCET or by decreasing the period of the task with increase in criticality levels. The variations in computation times or/and periodicity of tasks trigger the concept of mode change in MCS (Burns, 2014). A system begins at a defined criticality level and stays in the same level until all the jobs meet their current mode retention criteria, else a criticality mode change occurs. In low criticality (LC) mode, the completion of jobs take place by their low WCET. If any job executes beyond the WCET profiled for LC mode, the system changes mode and executes in high mode (Baruah, 2009, 2014; Baruah et al., 2010a,b,c, 2015a; Burns and Baruah, 2011; De Niz et al., 2009; Guan et al., 2011; Li and Baruah, 2010a,b). In dual criticality systems, the consequence of mode change is the suspension or delayed execution of LC jobs. For any criticality level, if a job executes beyond the WCET of that level, all the jobs of that level and lower are suspended or executed in degraded mode and the system changes its mode to high for execution. The WCET parameter of task is a vector that contributes to mode change due to the execution time. A mode change can also be triggered by change in task periodicity (Baruah and Chattopadhyay, 2013). Here, the frequency of job arrivals gives rise to mode change. The period parameter of task is a vector that contributes to mode change due to frequency of job arrivals.

Mode change triggers either delayed execution/suspension of LC jobs (Burns, 2013), retaining LC jobs based on importance (Fleming and Burns, 2014) or replacing HC jobs that triggered mode change with a new set of high importance jobs (Bletsas et al., 2018). The practical aspects of criticality mode switching like QoS improvement, industry relevant modes like no-fail/fail-safe/fail recovery and mode-specific schedulability analysis mechanisms require specific adaptations to task models.

2.1 Challenges in MCS

MCS have their own unique set of challenges to resolve. Some of the most vital ones are listed below. These challenges can be categorised as MCS principles/framework, industrial requirements and implementation details. These challenges are mapped to features like resources, QoS, OS overheads, energy, fault tolerance, virtualisation and parallel processing.

2.1.1 MCS framework

This section details the challenges with respect to criticality definition, mode switching, WCET estimation guidelines and practical representations.

C1 Criticality definition (Esper et al., 2015) – In academic publications, criticality refers to the modes of execution. Switching between these modes denotes the increase or decrease in system criticality level. In industry, system criticality refers to the level of assurance DAL/ASIL/SIL applied in software development related to safety functions. Academic and industrial definitions of criticality require convergence to have uniformity in task modelling, scheduling and design and development of applications.

C2 Mode switching – Mixed criticality introduces multiple modes of operations based on dropping or delaying of LC jobs (Burns, 2013). The modes such as no-fail/fail-safe/fail-recovery and the switching between these modes require complex measurement and validation mechanisms.

C3 Critically-based WCET estimation – Theoretical model of MCS recommends multiple values of WCETs – one per criticality level. The general notion is to have larger WCET at higher criticality levels. But having an implementation with varying degree of execution time at different confidence levels is not practical. With existing WCET estimation techniques, the value of WCET for a given criticality level cannot be known with complete certainty (Burns and Davis, 2013a). The probabilistic approach of timing measurement is not matured enough to deploy it into practical use. Moreover, in multi-core MCS, WCET becomes a function of heterogeneous tasks, heterogeneous cores, shared resources and task-to-core mapping.

C4 Practical representations – System designers require efficient practical notations and design languages to represent industrial applications. In MCS, modes of operations, mode relapse, resource reclaim, multiple deadline values and many-core/multi-core deployment trigger additional complexities for system representation. Graph-based notations are one of best suited tools for design. Hence, there is need to incorporate graph-based task models in MCS. Another research area is the probabilistic task models for timing estimation to incorporate complex and stochastic system scenarios.

2.1.2 Industrial requirements

This section presents industry driven challenges on certifications, QoS, energy efficiency and fault tolerance. The introduction of multi-core systems also gives rise to additional challenges.

C5 Certification requirements – MCS are exposed to certification mandates due to their safety critical needs. Certification related scheduling problems raise many interesting challenges like – pessimistic WCET, resource over-reservation, computing power under-utilisation, low QoS, etc. These challenges are not addressed by conventional mixed criticality schedulers. To meet certification needs, a certification authority makes conservative assumptions regarding the WCET of safety critical tasks (Baruah et al., 2010a,b, 2011a). This results in over allocation of resources to these tasks and a possible impact on QoS, as lower mission critical tasks may not get sufficient resources or computing power. In multi-core MCS, this results in under utilisation of cores and may increase the power per productive work factor. Also, in MCS, deadline misses of high criticality (HC) tasks are highly undesirable. A clear-cut definition that determines the relationship between deadline miss and criticality failure ceases to exist (Guan et al., 2017).

C6 QoS – In MCS, QoS improvement depends on the maximum possible completions of LC jobs. QoS improvement mechanisms include dropping or delaying of LC jobs, retaining LC jobs based on importance (Fleming and Burns, 2014) and replacing existing HC jobs with a new set of high importance jobs (Bletsas et al., 2018) during mode change. To have an enhanced QoS, it is required to analyse industry relevant usage scenarios, customer needs and system configuration scalability. Extensive usage scenarios and complex features trigger the need for QoS coherent task models, subject to system configuration capability.

C7 Energy efficiency – MCS have inherent challenges of size weight and power (SWAP). The introduction of multi-core systems presented novel ideas on energy saving by using procrastination and core level shutdown and other dynamic energy optimisation techniques. Energy optimisation techniques like dynamic-voltage-and-frequency-scaling (DVFS) and dynamic-power-management (DPM) are applied to improve energy performance in RTS. There is a need of mixed criticality energy models to perform energy aware schedulability analysis.

C8 Fault tolerance – The acceptable failure rate of a system varies based on the type of application/usage scenarios. To handle each of these usage scenarios, varying failure in time (FIT) measurements are required. This triggers the need for different mechanisms to improve the correctness of the system. For example, Axer et al. (2011) provided reliability analysis for task allocation in a system on chip (SoC) where only higher criticality tasks were duplicated for fault tolerant functioning. Safety analyses such as failure mode effects and criticality analysis, fault tree analysis, critical path analysis and freedom from interference analysis (Pintard et al., 2015) create additional challenges.

C9 Multi-core systems – Multi-core MCS were introduced by Anderson et al. (2009) and Mollison et al. (2010) for providing temporal isolation of mixed criticality jobs. In MCS industry, there is tremendous push to integrate multi-core systems due to their huge computing needs. This triggers the induction of fully/semi/no partitioning of tasks and multi-core mixed criticality scheduling with task migration capabilities. Beyond migration and load balancing capabilities, there are challenges with respect to task allocation/partitioning, resource allocation and management, time synchronisation, task-to-core mapping, computing/resource utilisation and job scheduling. Designing efficient algorithms, task models and simulation studies to suit multi-core MCS is an important research direction.

2.1.3 Implementation details

This section provides an overview on the implementation challenges in-terms of resource handling, OS overheads and parallel processing.

C10 Resources – Pellizzoni et al. (2010) measured the effect of resource interference in a eight-core system and observed that a task shows 300% increase in its execution time. Achieving predictive timing is one of the fascinating research areas in MCS and it includes proper allocation of shared resources and effective resource synchronisation. Though there is no scarcity of research (Lakshmanan et al., 2011; Zhao et al., 2014, 2015; Burns and Davis, 2013b; Zhao et al., 2018; Giannopoulou et al., 2013; Li and Wang, 2016; Hassan and Patel, 2016; Nair et al., 2019) in this domain, more specific and in depth research is still required to cope with changing MCS usage scenarios.

C11 OS overheads – A proper timing model considering OS overheads like context switching, page fault, queue waiting, etc. is mandatory for appropriate MCS schedulability analysis. There is a requirement for an accurate WCET measurement. Static WCET analysis techniques have inherent challenges (Abella et al., 2015; Hassan, 2017) such as lack of trustworthiness of timing models, incomplete documentation of register transfer level and fragility of program flow facts. Analysing hierarchical scheduling either with or without flattening has known limitations of over-provisioning. Hence, there is need for elaborate task models that consider OS overheads with normal and virtualised scenarios.

C12 Parallel processing – Increase in computational power requirements because of the growing software complexities lead to parallel processing systems in MCS. This brings the challenge of having task models related to parallel processing to perform schedulability analysis of parallel systems.

There are many task models and techniques in literature based on the need and type of analysis to overcome many of the listed challenges. The state-of-the-art indicates that challenges on industrial requirements and implementation are partially addressed by certain task models. It is also indicated that strengthening of research is required to overcome challenges especially related to MCS frameworks and principles. Burns postulated in Dagstuhl report (Baruah et al., 2015b) the need of an MCS framework with formal languages, models, techniques, protocols and analysis to allow trade-off for effective utilisation and isolation. These frameworks should depend on usage scenarios and application domains.

After analysing these challenges, it is decided to categorise task models based on specific attributes. Although diversity in task models is required, the research fraternity and industrial designers require a ready reckoner of apt task models that consider parameters in terms of attributes like resources, mode change, QoS, OS overheads, energy, fault tolerance and parallel processing. There is also need to deliberate on industry proven task representation schemes like parametric, graph-based and probabilistic task models. This survey is an attempt to provide the same.

2.2 *Outline notes*

The notations with uniform nomenclature are summarised in Table 1. Task models in Tables 2 to 3 are coded as UTx and MTx. Uni-core task models are coded with UTx and multi-core task models are coded with MTx, where x denotes a positive integer. The arrow over a parameter indicates that it is a vector. The number of elements in the vector is two for a dual criticality system indicating its value in low and high modes. The elements in the vector will have values in each mode of operation in a multi-criticality system. Sections 3 and 4 provide a thematic study on uni-core and multi-core mixed criticality task models with emphasis on attributes like resources, QoS, OS overheads, energy, fault tolerance virtualisation and parallel processing. Each subsections also detail the inadequacies and requirements with respect to each task model.

Table 1 Notation

Symbol used in this work	*Interpretation*	*Symbols used in other works*
	Basic parameters	
i	Task/job index	i, j
τ	Task	τ
J	Job	J
$T/\overrightarrow{T}$	Period	T, $T[L]$, Π, π, p, P, $p^{\max}$, W
ϕ	Job release time/phase	r, R, O, A
$D/\overrightarrow{D}$	Relative deadline	D, $D[L]$, d
$\mathbb{D}$	Absolute deadline	D
L	Criticality level	L, χ, ζ, l, κ, $Crit$, β, Z
$C/\overrightarrow{C}$	WCET	$\overrightarrow{C}$, C_l, $C[L]$, $e\overrightarrow{\epsilon}$, E, c^L, C^l, C^u, $\overrightarrow{c}$
$\overrightarrow{C}_{deg}$	WCET in degraded mode	C_{deg}
$\overrightarrow{C}^L$	WCET in low mode	$\overrightarrow{C}^L$, $\overrightarrow{C}(L)$
$\overrightarrow{C}^H$	WCET in high mode	$\overrightarrow{C}^H$, $\overrightarrow{C}(H)$
$\overrightarrow{\c{c}}$	Computation time	C, C^e
	Resource parameters	
$b/\overrightarrow{b}$	Intra-core blocking time	$B[L]$, B, b
$P/\overrightarrow{P}$	Fixed priority	P, p, $\overrightarrow{\eta}$, γ, pr
p	Active priority	p
l	Active criticality	l
σ	Set of semaphores (vector)	σ
λ	Preemption level (vector)	λ, y
$\oint$	Worst case stack space usage	ϕ
$\overrightarrow{\mathcal{J}}$	Release time jitter	J
$\wedge^{\min}$	Minimum number of memory accesses	$\mu^{\min}$
$\wedge^{\max}$	Maximum number of memory accesses	$\mu^{\max}$, $\wedge$
$exec^L$	Lower bound on computation time	$exec^L$
$exec^U$	Upper bound on computation time	$exec^U$
CM	Worst case number of cache misses	CM
Θ	Message size	NA
$\overrightarrow{M}$	Worst case memory access time	M, C^m
$\overrightarrow{\mathbb{I}}$	Worst case latency due to task interference	I
$\overrightarrow{M1}$	Worst case number of L1 cache misses	$M1$
$\overrightarrow{M2}$	Worst case number of shared LLC misses	$M2$
$\overrightarrow{B}$	Inter-core blocking time	B
$\overrightarrow{\varpi}$	Number of frequently used pages by a task in LLC	σ^L
$Q/\overrightarrow{Q}$	Number of memory accesses	$Q^{L/H}$
$\overrightarrow{\mathscr{C}}$	WCCT	$C[M]$

Table 1 Notation (continued)

Symbol used in this work	*Interpretation*	*Symbols used in other works*
	Context switching parameters	
C^{cs}	Context switching time (scalar/vector)	$\overrightarrow{C}^{cs}$, C^C, C^S
A	Address space	A
	Energy parameters	
$E/\overrightarrow{E}$	Energy/power consumption	E, Pow, $\overrightarrow{E}$
$\overrightarrow{f}$	Processor frequency level	f^L
ρ	Voltage and frequency scaling factor	ρ
α	Success ratio constraint	α
	Fault tolerance parameters	
f	Failure probability	f
$\mathbb{B}$	WCET of backups	B
$\overrightarrow{n}$	Number of executions of a task	$n[L]$
R	Reliability constraint	R
d	Dropping factor	d
O^d	Fault detection overhead	O^d
O^r	Roll-back overhead	O^r
O^v	Voting overhead	O^v
u	Relative task utility	u
m	Replication requirement	m
r	Distribution requirement	r
	Probabilistic task parameters	
$\overrightarrow{fe}$	Execution time probability mass function	fe
Ψ	Execution demand random variable	ζ
Ω	Execution scenarios (vector)	Ω
$\mathcal{M}$	Set of events (vector)	$\mathcal{M}$
$\mathbb{P}$	Probability measure of Ω (vector)	$\mathbb{P}$
	Graph-based parameters	
β	Block in a synchronous program	B
G	Graph/DAG	G, g, Dep
V	Vertices in a graph (vector)	V, $\mathcal{V}$
ϵ	Edges in a graph (vector)	E, $\mathcal{E}$
ϵ_{cf}	Directed edge that defines task control flow	E_{cf}
ϵ_{ms}	Directed edge that defines mode switch	E_{ms}
μ	Mode of the job	μ
ς	Function of allowed interference between tasks	σ
TFE	Timing failure event	TFE

Table 1 Notation (continued)

Symbol used in this work	*Interpretation*	*Symbols used in other works*
	QoS parameters	
x^L	Non-uniform deadline scaling factor at criticality level L	x^l
P^{ER}	Early release points (vector)	P^{ER}
I	Importance	I, λ, V, S
$\mathbb{C}$	Component	$\mathbb{C}$
$\mathcal{W}$	Tasks assigned to a component (vector)	$\mathcal{W}$
TL	Tolerance limit	TL
TM	Task mode	M, B, Ω
S	Stretching factor	S, E
$V/\overrightarrow{V}$	QoS Values	V, qos
$\overline{\alpha}^u$ (Δ)	Upper limit on task activations over a time interval Δ	$\overline{\alpha}^u$ (Δ)
δ (q)	Minimum time interval of q + 1 activations	δ_i (q)
	Virtualisation parameters	
$\mathbb{R}^{\min}$	Minimum resource requirements	$\phi^{\min}$
$\mathbb{R}^{\max}$	Maximum resource requirements	$\phi^{\max}$
$\mathbb{Q}$	Quality	Q
$\mathbb{F}$	Set of functions	enter, main, leave
$\mathbb{S}$	Subset of task/virtual machine profiles	NA
	Parallel task parameters	
$\approx$	Number of threads	m
$\mathbb{M}/\overrightarrow{\mathbb{M}}$	Number of cores	m
$\overrightarrow{work}$	Work for parallel tasks	$work_{O/N}$, C
$\overrightarrow{span}$	Span for parallel tasks	$span_{O/N}$, L

3 Uni-core MCS task models

Vestal's (2007) model is the basis of many significant works published in mixed criticality research. He used a periodic task model with four criticality levels. A task in Vestal's model is characterised by four-tuple with elements: period (T), deadline (D), criticality (L) and WCET vector ($\overrightarrow{C}$), where the WCETs are non-decreasing with increase in criticality. Vestal's seminal work influenced the evolution of many task models in the mixed criticality domain. This section provides a thematic survey on uni-core mixed criticality task models with emphasis on resources, mode change, energy, fault tolerance and OS overheads.

Table 2 Task models for uni-core MCS

Code	*Task* τ_i	*Usage scenario*	*Ref*
UT1	$(T_i; D_i; L_i; \overrightarrow{C}_i)$	$\overrightarrow{C}_i$ is vector that has minimum of two levels and can extend to N levels	Baruah et al. (2010b, 2010a, 2010c, 2011b, 2011a), Baruah and Burns (2011), Burns and Baruah (2011), Dorin et al. (2010), Ekberg and Yi (2014), Gettings et al. (2015), Guan et al. (2011), Li and Baruah (2010a, 2010b), Santy et al. (2012), Vestal (2007), Chwa et al. (2018), Gu and Easwaran (2017), Guo et al. (2018), Gupta et al. (2018), Kahil et al. (2018), Mahdiani and Masrur (2018), Agrawal et al. (2019) and Baruah and Burns (2020)
UT2	$(T_i; \overrightarrow{D}_i; L_i; \overrightarrow{C}_i)$	Virtual deadline is estimated and is used for improving schedulability of HC jobs in MCS. $\overrightarrow{D}_i$ is a vector with virtual and actual deadlines	Baruah et al. (2015a)
UT3	$(\overrightarrow{T}_i; D_i; L_i; C_i)$	Criticality is proportional to frequency. Mode change results in increase in frequency. $\overrightarrow{T}_i$ is a vector with two to N elements	Baruah and Chattopadhyay (2013)
UT4	$\tau_i = (T_i; D_i; L_i; \overrightarrow{C}_i; \overrightarrow{b}_i)$	Incorporates criticality specific blocking times for resource synchronisation and deadlock avoidance	Burns (2013)
UT5	$\tau_i = (T_i; D_i; L_i; \overrightarrow{C}_i)$	Considers WCET under non-overloaded (C) and overloaded (Co) conditions. C and Co are equivalent to low and high values of WCET in a dual MCS. Incorporates resource synchronisation by extending zero slack scheduler	Lakshmanan et al. (2011)
UT6	$\tau_i = (T_i; D_i; L_i; \overrightarrow{C}_i; P_i, p_i, l_i, \sigma_i)$	Provides resource synchronisation with nominal/active priority, nominal/active criticality and semaphores	Zhao et al. (2014)
UT7	$\tau_i = (T_i; \overrightarrow{D}_i; L_i; \overrightarrow{C}_i; \overrightarrow{\lambda}_i)$	Provides resource synchronisation with low and high values of preemption thresholds	Zhao et al. (2015)
UT8	$\tau_i = (T_i; D_i; L_i; \overrightarrow{C}_i; P_i; \lambda_i; \oint_i)$	Improves schedulability by using preemption thresholds and reducing stack space	Zhao et al. (2018)

Table 2 Task models for uni-core MCS (continued)

Code	*Task* τ_i	*Usage scenario*	*Ref*
UT9	$\tau_i = (\overrightarrow{T}_i;\ D_i;\ L_i;\ \overrightarrow{C}_i;\ b_i;\ \overrightarrow{\mathcal{J}}_i)$	Message passing by considering blocking times and jitters	Burns and Davis (2013b)
UT10	$\tau_i = (\overrightarrow{T}_i;\ L_i;\ \overrightarrow{C}_i;\ \overrightarrow{P}_i)$	A vector-based model to improve QoS and safeguard scheduling of LC jobs	Su et al. (2016) and Boudjadar et al. (2019)
UT11	$\tau_i = (\overrightarrow{T}_i;\ L_i,\ \overrightarrow{C}_i;\ P_i^{ER})$	An elastic task model with variable periods to improve QoS of LC jobs	Su and Zhu (2013)
UT12	$\tau_i = (T_i;\ D_i;\ L_i;\ \overrightarrow{C}_i;\ x_i^{L_i})$	Non-uniform deadline scaling to improve QoS	Chen et al. (2018)
UT13	$\tau_i = (\overrightarrow{T}_i;\ \overrightarrow{D}_i;\ L_i;\ \overrightarrow{C}_i)$	Minimises service degradation for LC jobs by using adaptive task profile	Huang et al. (2014c)
UT14	$\tau_i = (T_i;\ L_i;\ \overrightarrow{C}_i;\ TM_i)$	Adaptive task-based mode change to improve schedulability of LC jobs	Lee et al. (2017) and Sundar and Easwaran (2019)
UT15	$\tau_i = (T_i;\ D_i;\ L_i;\ \overrightarrow{C}_i;\ P_i;\ I_i)$	Job dropping based on predefined importance factor applied to LC jobs	Fleming and Burns (2014)
UT16	$\tau_i = (T_i;\ D_i;\ L_i;\ \overrightarrow{C}_i;\ I_i)$	Job dropping based on importance factor applied to all jobs	Bletsas et al. (2018)
UT17	$\tau_i = (T_i;\ D_i;\ L_i;\ \overrightarrow{C}_i),\ \mathbb{C} = (\mathcal{W},\ TL)$	Job dropping based on predefined tolerance limit	Gu et al. (2015)
UT18	$\tau_i = (\overrightarrow{T}_i,\ D_i;\ L_i;\ \overrightarrow{C}_i;\ \overrightarrow{P}_i)$	Adaptive task profile to improve QoS of LC jobs	Burns and Baruah (2013)
UT19	$\tau_i = (\overrightarrow{T}_i;\ L_i;\ \overrightarrow{C}_i;\ \overrightarrow{P}_i;\ \overrightarrow{C}_i^{cs1};\ \overrightarrow{C}_i^{cs2})$	Minimises latency of event-triggered tasks, while ensuring schedulability of time-triggered tasks. The model incorporates virtualised systems and the impact of context switching	Evripidou (2016)
UT20	$\tau_i = (T_i;\ D_i;\ L_i;\ \overrightarrow{C}_i;\ C_i^{cs1},\ C_i^{cs2};\ A_i)$	Considers context switching overhead between tasks of the same criticality that share the address space (C_i^{cs1}) and those of different criticalities that do not (C_i^{cs2})	Davis et al. (2018)
UT21	$\tau_i = (T_i;\ L_i;\ \overrightarrow{C}_i;\ \overrightarrow{E}_i)$	Incorporates energy consumption vector to improve schedulability	Völp et al. (2014)
UT22	$\tau_i = (T_i;\ D_i;\ L_i;\ \overrightarrow{C}_i;\ \overrightarrow{f}_i)$	Includes processor frequency vector to reduce energy consumption and improve schedulability	Huang et al. (2014a)

Table 2 Task models for uni-core MCS (continued)

Code	*Task* τ_i	*Usage scenario*	*Ref*
UT23	$\tau_i = (T_i; L_i; \overrightarrow{C}_i; E_i; \alpha_i)$	Probabilistic framework for addressing the uncertainty in power management	Asyaban et al. (2016)
UT24	$\tau_i = (T_i; D_i; L_i; \overrightarrow{C}_i; f_i)$	A probabilistic framework to avoid over-provisioning of resources and improve schedulability of the system	Guo et al. (2015) and Huang et al. (2014b)
UT25	$\tau_i = (T_i; D_i; L_i; \overrightarrow{C}_i^L; \overrightarrow{C}_i^H)$ where $\overrightarrow{C}_i^L = <C_{i,p}, \mathbb{B}_{i,1}, ..., \mathbb{B}_{i,f}>$ and $\overrightarrow{C}_i^H = <\widehat{C}_{i,p}, \hat{\mathbb{B}}_{i,1}, ..., \hat{\mathbb{B}}_{i,f}, ..., \hat{\mathbb{B}}_{i,F}>$	Incorporates back up tasks to achieve fault tolerance and guaranteed schedulability	Pathan (2014)
UT26	$\tau_i = (T_i; D_i; L_i; \overrightarrow{C}_i; P_i)$	Fail operational or fail robust, redundancy system using priority	Burns et al. (2018) and Abdeddaïm (2020)
UT27	$\beta_i = (T_i; \overrightarrow{C}_i)$	Represents synchronous reactive execution of tasks	Baruah (2014)
UT28	$\tau_i = (V(\tau); \epsilon_{cf}(\tau); \epsilon_{ms}(\tau))$, $V(\tau) = (C(v); D(v); \mu(v))$	Represents dependencies such a job arrival, mode switching and control flow	Ekberg et al. (2013)
UT29	$G(V; \epsilon; \varsigma)$, $V = (T_i; D_i; L_i; \overrightarrow{C}_i)$	Represents interference among tasks	Huang et al. (2013)
UT30	$J_i = (\phi_i; L_i; D_i; \overrightarrow{fe_i})$	Uses probability mass function	Alahmad et al. (2011)
UT31	$J_i = (D_i; L_i; \overrightarrow{C}_i; \Psi_i(\Omega_i, \mathcal{M}_i, \mathbb{P}_i))$	Performs risk analysis	Alahmad and Gopalakrishnan (2018)

3.1 Basic task model

The works in Baruah et al. (2010c, 2010b, 2010a), Dorin et al. (2010), Li and Baruah (2010a, 2010b), Baruah et al. (2011a), Baruah and Burns (2011), Burns and Baruah (2011), Guan et al. (2011), Santy et al. (2012), Ekberg and Yi (2014), Baruah et al. (2015a), Gu and Easwaran (2017), Kahil et al. (2018), Mahdiani and Masrur (2018), Agrawal et al. (2019) and Baruah and Burns (2020) use the same task model as Vestal. Although Vestal's (2007) model considered multiple levels of criticalities, most of the works that followed Baruah and Chattopadhyay (2013), Baruah et al. (2010a, 2010b, 2011a, 2010c), Burns and Baruah (2011), De Niz et al. (2009), Ekberg and Yi (2014), Guan et al. (2011), Li and Baruah (2010a, 2010b), Agrawal et al. (2019) and Baruah and Burns (2020) considered only two criticality levels. This dual criticality task model was used to explore the schedulability and complexity of mixed criticality workloads consisting of periodic and sporadic tasks.

Table 2 presents the basic task models (UT1, UT2 and UT3) and their usage scenarios. Using UT1, scheduling schemes like own criticality-based priority (OCBP) (Baruah et al., 2010a), extended OCBP (Li and Baruah, 2010a), priority list reuse scheduling (PLRS) (Guan et al., 2011), partitioned criticality (PC), static mixed criticality (SMC), adaptive mixed criticality (AMC) (Baruah et al., 2011a) and AMC with criticality aware assignment of priorities (CAAP) (Burns and Baruah, 2011) were implemented along with their schedulability analyses. Li and Baruah (2010b) performed load-based schedulability analysis and Dorin et al. (2010) furnished sensitivity analysis using UT1.

Baruah et al. (2015a) (UT2) proposed EDF with virtual deadlines (EDF-VD) and evaluated its performance with the help of the processor speedup metric. The analysis was extended upto thirteen criticality levels. In this task model, deadline is interpreted as a vector having two values – virtual and actual deadlines. Gu and Easwaran (2017) proposed a demand bound-based schedulability test for EDF-VD and presented a new method to calculate virtual deadlines. The work in Agrawal et al. (2019) presented the superiority of semi-clairvoyant scheduling over mixed criticality scheduling in terms of speed up factor (1.5 v/s 1.618). Baruah et al. extended EDF-VD and fluid algorithms to incorporate robustness and resilience in Baruah and Burns (2020).

Baruah and Chattopadhyay (2013) proposed a variant of the basic task model (UT3) for MCS where instead of varying computation times, period of the task is varied with criticality. In this model, period of the task decreases with increase in criticality. Schedulability analysis shows that UT3 is more suitable for AMC than PC and SMC.

Most of these works (Baruah et al., 2011a; Dorin et al., 2010; Ekberg and Yi, 2014; Gettings et al., 2015; Santy et al., 2012; Chwa et al., 2018; Gu and Easwaran, 2017; Guo et al., 2018; Mahdiani and Masrur, 2018; Baruah and Chattopadhyay, 2013) are evaluated using synthetic task sets, though Vestal (2007) in his seminal work used task sets from the avionic industry.

Though the basic task model serves as a stepping stone to carry out research and analysis, it lacks consideration of practical factors. The factors like resources, I/O, fault tolerance, energy consumption, context switching and OS related overheads need to be considered for obtaining a real world timing scenario from design phase to deployment phase. Also, there is a need to map the criticality aspect of task model to the industry standard. There exist works that extended the basic models with factors like overload execution time (De Niz et al., 2009), blocking time (Burns, 2013), failure probability (Guo et al., 2015), power (Asyaban et al., 2016), context switching time (Evripidou, 2016), etc. The next subsections survey extensions to the basic task model.

3.2 Resource related task models

The impact on resource sharing among critical tasks is one of the fascinating research problems (Lakshmanan et al., 2011; Zhao et al., 2014). Modelling of MCS requires careful selection of task parameters related to shared resource usage, resource synchronisation and communication.

Table 2 presents resource-based task models (UT4–UT9) with their usage scenarios. Burns (2013) (UT4) proposed priority ceiling protocol (PCP) for dual MCS where only tasks of the same criticality could access shared resources. The work extended the basic task model to accommodate blocking times (b) and WCET is computed by considering the resource access time. Resource synchronisation techniques like priority and criticality inheritance protocol (PCIP) and priority and criticality ceiling protocol (PCCP) (Lakshmanan et al., 2011), highest-locker criticality priority ceiling protocol (HLCPCP) (Zhao et al., 2014) and mixed-criticality stack resource policy (Zhao et al., 2015) are proposed in literature. Lakshmanan et al. (2011) (UT5) considered the issues of criticality and priority inversion due to shared resources in dual MCS. In this model, the low and high values of WCET are labelled as WCET under non-overloaded and WCET under overloaded conditions respectively. Blocking time experienced by a job due to a low priority job under non-overload condition and LC jobs under overloaded condition is computed. Zhao et al. (2014) (UT6) considers additional parameters like active/nominal priority, active criticality and semaphores. The nominal values of priority (P) and criticality (L) are assigned initially at design time while the active values (p and l) are assigned dynamically according to PCP and highest-locker criticality (HLC) protocol. These nominal/active values are used to dispatch or drop jobs during execution. Zhao et al. (2015, 2018) combined preemption threshold scheduling and MCS for optimising stack usage. Zhao et al. (2015) (UT7) considered the additional parameter of preemption levels in high and low mode. These values are inversely proportional to their relative deadlines. The preemption levels are used to ensure that a task does not block more than once in each mode of execution. In UT8 (Zhao et al., 2018), the basic task model is augmented with priority, preemption threshold and worst case stack space usage as parameters. Preemption thresholds and priority are used to control the number of preemptions per job and the worst case stack space usage per task. The work aimed at enhancing schedulability and reducing stack space usage.

Message passing is used extensively for data transfer in MCS. Burns and Davis (2013b) (UT9) characterised the task model for message passing with additional parameters of blocking time and jitter. The work also states that a message passing task model can be represented with vectors for period, jitter and WCET. A mixed criticality protocol for message passing over controller area network (CAN) is presented and a schedulability analysis is derived. Brandenburg (2014) also proposed a mixed criticality IPC (MC-IPC) scheme with shared resources stationed on resource servers. The works in Lakshmanan et al. (2011), Zhao et al. (2014, 2015), Burns and Davis (2013b) and Zhao et al. (2018) are evaluated using synthetic task sets.

Certification requirements in MCS compel resource over-provisioning for complete isolation. But, for efficient utilisation of resources, a tighter upper bound on WCET including blocking due to resources need to be estimated. The impact of resource types like memory/displays/pipes/signals, synchronisation resources like sequence numbers/mutex/database locks/monitors and communication protocols need to be considered for achieving a realistic upper bound. Task models that contemplate on these resources and communication aspects need to be consolidated.

3.3 QoS related task models

Burns (2014) surveyed various mode switching models and outlined different aspects of mode switching. This section surveys mechanisms for improving QoS along with their task models.

The acceptance of more LC jobs improve the QoS of the system. Guo et al. (2018) used EDF-VD and improved the schedulability of LC jobs by limiting the number of

HC jobs that could exceed their low WCET values. The limiting factor was based on offline computations. Santy et al. (2012) presented latest completion time to improve the QoS. They achieved 24% to 32% reduction in LC task suspensions. Instead of dropping LC jobs during a mode change, Baruah and Burns (2011) proposed to reduce their priorities, thus ensuring schedulability of HC jobs. The work in Gettings et al. (2015) introduced a method that allowed graceful degradation of the system during a mode change. They proposed AMC-weakly hard to guarantee HC job completions by skipping limited number of successive LC jobs. The task models for the above QoS mechanisms use UT1.

Tables 2 (UT4, UT10–UT19, UT23, UT26, UT29, UT31) presents task models related to improving QoS. Burns (2013) (UT4) allowed LC jobs to continue execution in best effort as long as the HC jobs did not miss their deadlines. Su et al. (2016) (UT10) proposed a mode switched fixed priority scheduler to provide guaranteed service for LC jobs. Here, scheduling of jobs is carried out using fixed priority, where priorities are calculated by employing nonlinear optimisation and branch and bound search tree techniques. In this work, LC jobs are not suspended but continue to perform in high mode. To facilitate this, the task model is modified to have low and high values of periods and priorities in both modes. The values of periods vary in low and high modes only for LC tasks. When mode change occurs, the LC tasks execute using their high value of periods. The low and high values of priorities of LC jobs are used to improve schedulability of HC jobs.

Another method based on elastic task model was proposed by Su and Zhu (2013) (UT11). The work also assigns low and high values of periods to LC jobs. If mode change takes place, LC jobs are executed to use their low WCET and high value of period. A set of early release points (P^{ER}) are also associated with LC jobs to ensure their early release in order to utilise the slack of HC jobs and ensure the schedulability of more number of LC jobs. This work proposed early-release EDF (ER-EDF) and showed it performs better than EDF-VD with respect to accommodation of more LC jobs.

Chen et al. (2018) (UT12) extended EDF-VD with non-uniform deadlines in each mode thus allowing fine grained transitions and improving QoS for LC jobs. Huang et al. (2014c) (UT13) extended EDF-VD to guarantee service to LC jobs whenever HC jobs executed beyond their low WCETs. A minimum service degradation factor for LC jobs and a deadline tuning factor for HC jobs is computed such that schedulability is guaranteed in all modes. Lee et al. (2017) (UT14) also extended EDF-VD and presented a mode change mechanism at the task level rather than system level. In this work, HC tasks are augmented with a task mode (TM) parameter that defines the criticality mode at each level. An adaptive online mechanism is presented to minimally drop LC jobs based on their decreasing order of utilisations. More recently, Boudjadar et al. (2019) proposed a dynamic online mode change mechanism at both the task level and system level. The work uses the same task model as UT10 with the addition of TM parameter updating dynamically based on the current task execution time. To improve QoS for LC jobs, a context-aware flexible degradation task model is proposed in Sundar and Easwaran (2019) where $\overrightarrow{C}$ represents low WCET, high WCET and n – 1 degraded execution times for a system with n tasks. The TM parameter is used used to define three modes, namely normal, safe and degraded. The work uses UT14. Fleming and Burns (2014) (UT15) added an importance parameter to all except the highest criticality jobs. This importance information is used to drop jobs during a mode change. The jobs are dropped according to their increasing order of importance. Audsley's priority assignment mechanism was modified to allocate low priority to lower importance tasks. A similar model was proposed by Bletsas et al. (2018) (UT16). Here, the importance parameter is added to all jobs and job dropping is based on importance not criticality. Gu et al. (2015) (UT17) used a tolerance parameter to decide whether to drop LC jobs or not. Here, tasks are divided into components ($\mathbb{C}$) and each component has a defined tolerance threshold (TL). Two types of mode switches are defined – internal and external. In an internal mode switch, if a HC job within a component exceeds its low WCET, all the LC jobs within that component are dropped. All jobs in other components are not affected. But, if the number of HC jobs overrunning the low WCET exceed TL, then all the LC jobs in other components are also dropped. Burns and Baruah (2013) (UT18) presented three mechanisms that allowed non-interfering LC jobs to continue execution during a mode change. The first method aimed at reducing the priorities of LC jobs, the second focused on reducing the computation time and the third considered increasing the job periods. The work also defines the circumstances under which a system can return to LC mode. Evripidou (2016) (UT19) considered a three criticality level system and recommended two degraded modes. When a job executes beyond the low WCET, the system enters the first degraded mode where no jobs are dropped. An entry to the next degraded mode results in LC jobs being dropped. Asyaban et al. (2016) (UT23) put forth a scheduling mechanism for a battery less system that restarted all pending LC jobs when it switched from high to low mode to improve QoS. A fault mode mechanism was presented by Abdeddaïm (2020) (UT26) where the task model adopted a priority parameter to decide on the number and type of LC jobs to be suspended. Huang et al. (2013) (UT29) presented an interference constraint graph (ICG) mechanism and declared that it may not be necessary to drop all LC jobs, as not all these jobs interfere with HC jobs. The interference edges from the graph were used for improving schedulability of LC jobs. Alahmad and Gopalakrishnan (2018) (UT31) examined the probability of a job's deadline miss and accommodated LC jobs based on this probabilistic measure.

Most of these works (Guo et al., 2018; Santy et al., 2012; Gettings et al., 2015; Su and Zhu, 2013; Chen et al., 2018; Lee et al., 2017; Fleming and Burns, 2014; Gu et al., 2015; Asyaban et al., 2016; Huang et al., 2013; Alahmad and Gopalakrishnan, 2018) are evaluated using synthetic task sets, however Huang et al. (2014c), Evripidou (2016), Sundar and Easwaran (2019) and Boudjadar et al. (2019) used practical task sets from the avionic industry and automotive industry respectively.

It is observed that most works exist on mode change from low to high, whereas returning back to low mode – mode relapse, also needs attention. There are conditions under which the system can return back to low mode, the most common being when the system is idle (Burns and Baruah, 2013; Baruah et al., 2015a). Future auto-correcting MCS mandates mode relapse in a staged manner to have seamless connect and (re)functioning of the system.

3.4 OS overheads related task models

Often, OS overheads are ignored during schedulability analysis. These overheads can have substantial impact on the overall schedulability of tasks. Table 2 (UT19, UT20) provides task models related to OS overheads. The work by Evripidou (2016) (UT19) considered three criticality levels and added parameters of context switching vectors and a priority vector to their task model. $\overrightarrow{C}_i^{cs1}$ and $\overrightarrow{C}_i^{cs2}$ account for context switching overheads before and after execution of the task in each mode of operation. The work aims at scheduling event-driven tasks with minimum latency while ensuring the overall schedulability of the system. They use virtualisation with periodic and deferrable servers to schedule time-triggered and event-triggered tasks respectively. Davis et al. (2018) (UT20) considered the impact of spatial isolation on WCET estimation. They use parameters $\overrightarrow{C}_i^{cs1}$ and $\overrightarrow{C}_i^{cs2}$ for context switching between processes/tasks having distinct criticalities and switching between threads within the same process/tasks of the same criticality respectively. The work proposes a heuristic approach for allocating priorities with an aim to decrease large context switching costs. Davis et al. (2018) used synthetic task sets to evaluate their work while Evripidou (2016) used practical task sets from the automotive industry.

A realistic timing model in MCS is estimated using WCET, recovery time and OS overheads. OS overheads such as queuing time, sleep timer, interrupt/task suppression time, interrupt latency, blocking, page fault, etc. need to be considered. These factors will provide a practical upper bound on WCET suitable for industrial applications.

3.5 *Energy related task models*

In MCS, high computing volume with increasing energy demands elicit the need for efficient energy optimisation techniques. This necessitates amendments in task modelling related to energy parameters. Table 2 (UT21–UT23) provides task models that consider energy related parameters. Völp et al. (2014) (UT21) proposed an energy constrained task model. The work focused on the importance of dynamic energy consideration in MCS and showed an improvement in schedulability by 17%. The basic task model is enhanced with an dynamic energy consumption vector ($\overrightarrow{E}$). Huang et al. (2014a) (UT22) considered energy consumption and its impact on MCS. The objective was to reduce dynamic power by decreasing processor frequency. To facilitate this, the task model is extended with a processor frequency parameter ($\overrightarrow{f}$) in high and low modes of execution. The work combines DVFS and EDF-VD. Asyaban et al. (2016) (UT23) put forward a scheduling mechanism for a battery less dual criticality uni-core system. Besides the basic parameters, the task model considers two additional parameters – one for power consumption (E) and the other for success ratio constraint (α). Power consumption denotes the power consumed by a task independent of the mode. The success ratio constraint conveys the probability with which the system must guarantee enough energy for a job of a task to complete execution. Though the traditional MCS model focuses on adhering to timeliness guarantees, the MCS model with energy constraints focus on schedulability of LC and HC jobs by considering their success ratio constraints. The works (Asyaban et al., 2016; Völp et al., 2014) were evaluated using synthetic task sets and Huang et al. (2014a) was evaluated using both synthetic and practical task sets from the avionic domain.

Energy efficiency and deadline honoring are conflicting requirements. The usage of procrastination and energy optimisation techniques in practical applications without hampering certification is an emerging domain of MCS research and requires an elaborate framework with apt task parameters.

3.6 Fault tolerance related task models

There is a distinct relationship between fault tolerance and MCS. By providing spatial/temporal/fault isolation in MCS, a high degree of fault tolerance can be achieved. Table 2 (UT24–UT26) presents task models with fault tolerance. Guo et al. (2015) (UT24) considered a sporadic dual criticality task model where a HC task is augmented with parameter f – the failure probability where $1 - f$ indicates the probability of a mode change in an hour. The work assumes $C[High]$ to be deterministic but the combination of $(C[Low], f)$ to be probabilistic for HC tasks. The work proposed EDF with clustering and they claimed through experimental evaluation a schedulability improvement of 21% over traditional MCS with UT1 and EDF-VD. Guo et al. (2021) extended Guo et al. (2015) with a system failure probability threshold and presented a schedulability analysis for both uni-core and multi-core systems. Huang et al. (2014b) (UT24) extended the task model with a failure probability parameter for all tasks. The work defines failure probability as the likeliness that a job may not complete by its deadline. A criticality-based re-execution profile using failure probability is computed so as to ensure safety and schedulability guarantees.

A distinct model for fault tolerant scheduling in MCS was presented by Pathan (2014) (UT25). The work proposes backup tasks to compensate for faults in primary tasks. The low/high WCET values of the task are expanded as WCET of the primary task and WCET of the backups. The cumulative values of primary and all the backups account for the total WCET in each mode. A fault tolerant scheduling algorithm was designed to guarantee schedulability of the system by considering backup tasks in case of fault in the primary task. To make the system more robust, Burns et al. (2018) (UT26) proposed a robust MCS. Robustness is attained by performing fail-operational and fail-robust analysis. The task model is augmented with priority and a robust/non-robust boolean parameter. Audsley's algorithm is used to assign unique priorities and AMC is used for scheduling. A task was termed robust if its job could be abandoned. A nominal/verification friendly fault tolerant approach was presented by Schmidt and García-Ortiz (2022). UT1 was used as the task model and an enhanced EDF-VD with non-uniform deadlines was proposed to handle errors and QoS. The works (Guo et al., 2015; Pathan, 2014; Burns et al., 2018; Guo et al., 2021) are evaluated using synthetic task sets and Huang et al. (2014b) is evaluated using both synthetic and practical task sets from the avionic domain.

Schedulability analysis with task parameters for temporal/spatial/fault isolation and the FIT requirements are mandated in industrial applications. An overall framework with these additional parameters needs to be consolidated.

3.7 Graph-based task models

Graph-based task models articulate dependencies and constraints in a system. There exist MCS task models where tasks are represented using graphs. Table 2 (UT27–UT29) presents graph-based task models. In Baruah (2014) (UT27), resource-efficient synchronous execution of tasks is shown using a dependency graph, where each vertex represents a block and each edge represents inter-block dependency. A block (β) is a fundamental unit of a synchronous program and is characterised by period, low WCET and high WCET. Ekberg et al. (2013) (UT28) presented a directed acyclic graph (DAG)-based model to support complex job arrivals, mode switching and synchronisation patterns. A task in DAG is defined by vertices and directed edges. Edges are further categorised as mode switching edges (ϵ_{ms}) and control-flow edges (ϵ_{cf}). Edges are labelled with period and vertices are defined with WCET, relative deadline and mode of the job. Huang et al. (2013) (UT29) put forth an ICG mechanism to capture the interference among tasks. ICG is a directed graph where vertices specify tasks and an edge exists form task τ_i to task τ_j if and only if τ_i interferes with τ_j. A parameter ς is used to define the allowable interference between tasks. A task is defined by parameters of UT1. The interference information from the graph is used to improve schedulability of the system. The work was evaluated using synthetic task sets.

Graph-based models provide clarity for designing systems and are most suitable for practical implementations. MCS research needs to evolve towards constructing realistic graph models that suit industrial MCS applications.

3.8 Probabilistic task models

The task models discussed so far are deterministic in nature and make pessimistic assumptions regarding the WCET of the system. Table 2 (UT24, UT30, UT31) presents probabilistic task models. To reduce the gap between the actual and pessimistic execution time, Guo et al. (2015) (UT24) considered a sporadic dual criticality task model for uni-core systems. LC jobs were scheduled using failure probability parameter of HC jobs to improve their schedulability. Alahmad et al. (2011) (UT30) presented a probabilistic model and provided a probability analysis for mixed criticality uni-core systems. The work identified deficiencies with regards to pessimistic timing behaviour of the deterministic model and proposed a probabilistic model to solve them. In a probabilistic task model, a job is defined by four-tuple with elements: release time, criticality, deadline and execution time probability mass function (fe) at each criticality level. A similar model that considered a probabilistic execution time matrix was proposed by Bhuiyan et al. (2020) for energy optimisation. The work reported energy savings of upto 46%. Subsequently, Alahmad and Gopalakrishnan (2018) (UT31) devised an online scheduling mechanisms and proposed a probabilistic task model framework. In this work, the job is extended with parameters of execution scenarios (Ω_i), events ($\mathcal{M}_i$), probability measure ($\mathbb{P}_i$) on Ω_i and demand random variable (ζ_i). These parameters were added for providing probabilistic schedulability guarantees. The works (Alahmad and Gopalakrishnan, 2018; Guo et al., 2015; Bhuiyan et al., 2020) were evaluated using synthetic task sets.

4 Multi-core MCS models

Over the last few decades, massive computing and certification requirements motivated the need for multi-core architectures in MCS. Multi-core systems are classified as homogeneous and heterogeneous architectures based on types of cores in use (Davis and Burns, 2011). The important factors for consideration in multi-core MCS are task assignment/task-to-core mapping, scheduling and migration. Efficient task allocation/task assignment/task-to-core mapping mechanisms consider utilisation, isolation, migration, energy efficiency and fault tolerance features.

Table 3 Task models for multi-core MCS

Code	*Task* τ_i	*Usage scenario*	*Ref*
MT1	$\tau_i = (T_i;\ D_i;\ L_i;\ \vec{C}_i)$	$\vec{C}$ is a vector that has a minimum of two levels and can extend to N levels	Li and Baruah (2012), Baruah et al. (2014), Pathan (2012), Lee et al. (2014), Baruah et al. (2015c), Gratia et al. (2014), Ramanathan and Easwaran (2015), Ren and Phan (2015), Xu and Burns (2015), Behera and Bhaduri (2018), Huang et al. (2018), Xu and Burns (2019), Lee et al. (2017), Ramanathan et al. (2018), Nagalakshmi and Gomathi (2018), Han et al. (2017), Kim et al. (2018), Baek and Lee (2020)
MT2	$Superblock = (\wedge_{i,j}^{\min};\ \wedge_{i,j}^{\max};\ exec_{i,j}^{L};\ exec_{i,j}^{U})$	Includes shared memory and cache profiles	Pellizzoni et al. (2010)
MT3	$\tau_i = (T_i;\ D_i;\ L_i;\ \vec{C}_i;\ CM_i)$	Includes throttling to prevent memory interference	Yun et al. (2012)
MT4	$\tau_i = (T_i;\ L_i;\ \vec{C}_i;\ \vec{C}_{i,deg};\ Dep)$	Incorporates dependency to offer temporal isolation	Giannopoulou et al. (2013) and Trüb et al. (2017)
MT5	$\tau_i = (T_i;\ D_i;\ L_i;\ \vec{C}_i;\ \vec{¢}_i;\ \vec{M}_i)$	Includes shared memory with memory access profile of computation phase and execution phase	Li and Wang (2016), Awan et al. (2018a) and Li and He (2017)
MT6	$\tau_i = (T_i;\ D_i;\ L_i;\ \vec{C}_i;\ \vec{¢}_i;\ \vec{\mathbb{I}}_i;\ \wedge_i^{\max})$	Controls interference because of shared bus with hardware-based hierarchical arbiter	Hassan and Patel (2016)
MT7	$\tau_i = (\phi_i;\ T_i;\ D_i;\ L_i;\ \vec{C}_i;\ \overrightarrow{M1}_i;\ \overrightarrow{M2}_i;\ \vec{b}_i;\ \vec{B}_i)$	Controls interference because of shared bus with hardware-based arbiter	Nair et al. (2019)
MT8	$\tau_i = (T_i;\ D_i;\ L_i;\ \vec{C}_i;\ \vec{\varpi}_i$	Includes page locking and reassignment of pages during mode change	Awan et al. (2017)
MT9	$\tau_i = (T_i;\ D_i;\ L_i;\ \vec{C}_i;\ \vec{Q}_i)$	Regulates core-wise access using dynamic memory access budgets	Awan et al. (2018b)
MT10	$MessageFlow_i = (\phi_i;\ T_i;\ L_i;\ \vec{\mathscr{C}}_i)$	Message passing on NoC	Dridi et al. (2019)
MT11	$\tau_{i\_critical} = (T_i;\ D_i;\ C_i)$, $\tau_{i\_non-critical} = (T_i;\ D_i;\ C_i;\ I_i;\ S_i)$	Elastic model to improve schedulability of non-critical tasks	Jan et al. (2013)

Table 3 Task models for multi-core MCS (continued)

Code	*Task τ_i*	*Usage scenario*	*Ref*
MT12	τ_{i_LO} = ($\overrightarrow{T}_i$; L_i; $\overrightarrow{C}_i$ P_i^{ER}), τ_{i_HI} = (T_i; L_i; $\overrightarrow{C}_i$)	Elastic model to provide minimum guarantee service for LC tasks	Su et al. (2013)
MT13	τ_i = (T_i; D_i; L_i; $\overrightarrow{C}_i$)	Improves QoS with the help of zero slack scheduler	De Niz and Phan (2014)
MT14	τ_i = ($\overrightarrow{T}_i$; L_i; $\overrightarrow{C}_i$)	Incorporates multiple execution times and periods to facilitate job migration during mode change	Al-Bayati et al. (2015)
MT15	τ_i = (T_i; L_i; $\overrightarrow{C}_i$; $\overrightarrow{V}_i$)	Incorporates QoS values for each LC tasks to improve their schedulability	Pathan (2017)
MT16	τ_i = ($\overline{\alpha}^u(\Delta)$ or $\delta_i(q)$; $\overrightarrow{C}_i$; L_i)	Employs slack to improve QoS	Hu et al. (2018)
MT17	τ_i = (T_i; D_i; $\overrightarrow{C}_i$)	Hierarchical scheduling approach with five containers for five criticality levels on each core	Anderson et al. (2009), Herman et al. (2012) and Chisholm et al. (2016)
MT18	$Profile_{\tau_i}$ = ($\mathbb{R}^{\min}$; $\mathbb{R}^{\max}$; $\mathbb{Q}$; $\mathbb{F}$; $\mathbb{S}$), $Virtual\_Machine\_Profile$ = (L; $\mathbb{R}^{\min}$; $\mathbb{R}^{\max}$; $\mathbb{Q}$; $\mathbb{F}$; $\mathbb{S}$)	Hierarchical scheduling with dynamic resource management	Groesbrink et al. (2013)
MT19	τ_i = (T_i; D_i; L_i; $\overrightarrow{C}_i^L$; $\overrightarrow{C}_i^H$; $\overrightarrow{E}_i$)	Incorporates parameters for heterogeneity and energy factors for task allocation	Awan et al. (2015, 2016a, 2016b)
MT20	τ_i = ($\overrightarrow{T}_i$; L_i; $\overrightarrow{C}_i$; P_i^{ER}; ρ_i)	Incorporates voltage and frequency scaling factor to provide energy management	Taherin et al. (2018)
MT21	τ_i = (T_i; D_i; L_i; $\overrightarrow{C}_i$; m_i; r_i)	MCS model with fault tolerance	Thekkilakattil et al. (2014)
MT22	τ_i = (T_i; D_i; L_i; $\overrightarrow{C}_i$; P_i; $\overrightarrow{n}_i$)	Incorporates multi-modes to provide fault tolerance and QoS	Caplan et al. (2017)
MT23	G = (V_G; ϵ_G; T_G; D_G; L_G; R_G; I_G; d_G; V_G) τ_i = ($\overrightarrow{C}_i$; map_i; P_i; O_i^d; O_i^r; O_i^v; G)	DAG model with fault tolerance and QoS	Choi et al. (2018)

Table 3 Task models for multi-core MCS (continued)

Code	*Task* τ_i	*Usage scenario*	*Ref*
MT24	$\tau_i = (T_i;\ D_i;\ L_i;\ \overrightarrow{C}_i;\ u_i)$	Criticality and utility aware adaptation to provide fault tolerance and QoS	Iacovelli et al. (2018)
MT25	$G_i = (V_i;\ \epsilon_i)$, $\tau_j \in V_i$, $\tau_j = (\overrightarrow{C}_j)$, $\epsilon_j = (\Theta_j)$ and $(T_{G_i};\ D_{G_i};\ L_{G_i}) \in G_i$	DAG for cost aware partitioning	Tamas-Selicean and Pop (2011)
MT26	$G = (V;\ \epsilon)$ $J_j \in V$, $J_j = (\overrightarrow{C}_j)$ and $(D_G) \in G$	Represents synchronous reactive execution of tasks	Baruah (2013)
MT27	$G_i = (V_i;\ \epsilon_i)$ $\tau_j \in V_i$, $\tau_j = (L_j;\ \overrightarrow{C}_j;\ TFE_j)$	DAG for increasing LC task availability	Medina et al. (2018)
MT28	$J_i = (\phi_i;\ \mathbb{D}_i;\ L_i;\ (J_i^1,\ J_i^2,\ ...,\ J_i^{Si}))$, $J_i^k = (\overrightarrow{C}_i^k;\ \approx_i^k)$	Accommodates parallel jobs with varying execution times	Liu et al. (2014)
MT29	$\tau_i = (T_i;\ D_i;\ L_i;\ \overrightarrow{C}_i;\ \mathbb{M}_i)$	Accommodates parallel tasks with varying execution times	Bhuiyan et al. (2019)
MT30	$\tau_i = (D_i;\ \overrightarrow{work}_i;\ \overrightarrow{span}_i)$	Execution times are represented as work and span in overloaded and non-overload conditions	Agrawal and Baruah (2018)
MT31	$\tau_i = (T_i;\ D_i;\ L_i;\ \overrightarrow{work}_i;\ \overrightarrow{span}_i)$	Parallel tasks with slack-based QoS	Pathan (2018)
MT32	$\tau_i = (D_i;\ L_i;\ \overrightarrow{work}_i;\ \overrightarrow{span}_i;\ \overrightarrow{\mathbb{M}}_i;\ S_i)$	Parallel tasks with elasticity to improve QoS	Gill et al. (2018)

The following subsections review multi-core MCS by considering various task models with emphasis on resources and communications, mode change, virtualisation, energy, fault tolerance and parallel processing.

4.1 Basic task model

The basic task model (MT1) for multi-core MCS is the same as uni-core MCS. The task model is defined by four-tuple elements namely, period (T), deadline (D), criticality (L) and WCET vector ($\overrightarrow{C}$). Table 3 presents MT1 for multi-core MCS with its usage scenario. Some of the works that use MT1 are (Li and Baruah, 2012; Baruah et al., 2014; Pathan, 2012; Lee et al., 2014; Baruah et al., 2015c; Gratia et al., 2014; Ramanathan and Easwaran, 2015; Ren and Phan, 2015; Xu and Burns, 2015; Behera and Bhaduri, 2018; Huang et al., 2018; Xu and Burns, 2019; Lee et al., 2017; Ramanathan et al., 2018; Nagalakshmi and Gomathi, 2018; Baek and Lee, 2020). Li and Baruah (2012) proposed global EDF-VD for multi-processor MCS and derived a schedulability analysis and speed up factor. Subsequently, Baruah et al. (2014) compared global EDF-VD with partitioned EDF-VD and found through simulation analysis that in the context of MCS, partitioned EDF-VD outperforms global EDF-VD. Partitioned EDF-VD is also superior in terms of speed up factor. Pathan (2012) proposed a global fixed priority scheduling algorithm and furnished schedulability analysis for a multi-processor MCS with multiple levels. A fluid-based approach along with schedulability analysis and speed up factor was presented by Lee et al. (2014). Baruah et al. (2015c) extended the work with an improved run time complexity and speed up factor. Ramanathan and Easwaran (2015) further extended the work and proposed two fluid mechanisms – sort and slope, and showed that they provide better schedulability than Baruah et al. (2015c) at the cost of a slight increase in time complexity. Lee et al. (2017) extended (Lee et al., 2014; Ramanathan and Easwaran, 2015) with a new optimal rate assignment and scheduling mechanism. Ramanathan et al. (2018) extended Lee et al. (2014) to incorporate multiple execution rates for HC carry over jobs to guarantee their completions. Gratia et al. (2014) proposed modification of the RUN scheduling algorithm for multi-core MCS. A different approach using partitioned scheduling and task grouping was proposed by Ren and Phan (2015). The work presented mixed-integer-nonlinear-programming (MINLP) for creating task groups. To schedule jobs, they used a server-based strategy within a group and EDF between groups. A similar task grouping using MINLP was proposed by Nagalakshmi and Gomathi (2018). In this work, scheduling within a group is implemented using an enhanced version of EDF. Xu and Burns (2015) proposed a semi-partitioned scheduling approach for a dual-core MCS and presented definite response time analysis. Behera and Bhaduri (2018) proposed a time-triggered scheduling mechanism – low-criticality-based-priority (LoCBP) for dependent and independent jobs and proved its dominance in terms of run time complexity over time-triggered mixed-criticality-priority-improvement (MCPI) algorithm. Baek and Lee (2020) presented a contention free scheduling mechanism for dual MCS. The works (Li and Baruah, 2012; Baruah et al., 2015c, 2014; Pathan, 2012; Lee et al., 2014; Baruah et al., 2015c; Ramanathan and Easwaran, 2015; Ren and Phan, 2015; Xu and Burns, 2015; Behera and Bhaduri, 2018; Lee et al., 2017; Nagalakshmi and Gomathi, 2018; Ramanathan et al., 2018; Baek and Lee, 2020) used synthetic task sets for evaluation.

Ward et al. (2013) perceived the importance of predictive computation for safety critical certification needs. The verification complexity/certification efforts increase aggressively with the number of applications in a multi-core system. To simplify the analysis, apt task models with parameters relating to resources and communication, OS overheads and parallel processing factors are required. The next subsections survey extensions to the multi-core MCS basic task model.

4.2 Resource related task models

In multi-core MCS, access to shared memory by concurrently running jobs lead to contention, which may result in increase of WCET. Table 3 (MT2–MT10) provides the list of existing resource related task models in literature. Pellizzoni et al. (2010) (MT2) observed that WCET of a task inflates by 2.96 times due to shared write request. The work computed arrival curves and performed delay analysis for a periodic task model. To facilitate this, a task is split into a series of sequential superblocks and each superblock is defined by a cache profile, which comprises of four elements:

minimum/maximum main memory requests and minimum/maximum computation time. Yun et al. (2012) (MT3) devised a regulation mechanism to control access to shared memory and prevent interference in a multi-core dual criticality system. To achieve this, the basic task model is augmented with worst-case number of cache misses (CM). Giannopoulou et al. (2013) (MT4) implemented scheduling using time-triggered criticality monotonic approach wherein only tasks of the same criticality could interfere on shared resources. The WCET in this work considers the min/max number of memory accesses and min/max computation times. Besides, two additional parameters are added to the task model – WCET vector in degraded mode ($\overrightarrow{C}_{deg}$) and DAG (G) representing dependencies among tasks. Degraded mode is the minimum execution profile for lower criticality tasks during a mode change. Trüb et al. (2017) (MT4) used the same task model as Giannopoulou et al. (2013) but with two differences:

1 WCET considers only number of memory accesses and computation times in low and high modes

2 DAG is omitted.

The work confers an exhaustive timing analysis by considering run-time overheads and interference due to shared memory. Li and Wang (2016) (MT5) examined both memory access time and computation time of a task. The WCET of this model is computed as the sum of memory access time and computation time. Li and He (2017) (MT5) extended Li and Wang (2016) where to enhance schedulability, priorities were assigned in two stages. First, based on memory access and next based on computation needs. To control access from various cores and prevent interference on shared bus, Hassan and Patel (2016) (MT6) used a bus arbitration mechanism. The WCET of tasks were supervised to dynamically control access to shared bus. In this work, tasks are grouped based on criticality and each task is augmented with task interference delay ($\mathbb{I}$) and maximum number of memory access requests ($\wedge^{\max}$). Another arbitration mechanism to control access to shared last level cache (LLC) was proposed by Nair et al. (2019) (MT7). The task model is enhanced with additional parameters of L1 cache misses ($\overrightarrow{M1}$), LLC misses ($\overrightarrow{M2}$), intra-core (b) and inter-core blocking times (B). To have an upper bound on the number of LLC misses, Awan et al. (2017) (MT8) partitioned LLC and assigned to each task. Each task maintains the number of lock down pages (ϖ) in their partition. The WCET is a function of ϖ. Awan et al. (2018a) (MT5) used the same task model as Li and Wang (2016) to regulate memory bandwidth in order to prevent interference due to main memory access. The work was extended in Awan et al. (2018b) (MT9) to allow dynamic memory access control.

Dridi et al. (2019) (MT10) proposed improving worst case communication time (WCCT) of HC messages using network on chip (NoC) router. The task model is adapted to suit message transfers. A message flow is defined with parameters of ϕ – for the first message released, T – between consecutive messages, $\overrightarrow{\mathscr{C}}$ – of a message in a flow in each mode of operation and L – for each flow. Resource synchronisation in multi-core systems is more challenging than uni-core systems as both tasks and resources are distributed. Priority/criticality inversion in multi-core systems further increases the blocking time of high priority/critical jobs. In view of this, Han et al. (2017) proposed a resource synchronisation mechanism for multi-core systems using partitioned EDF and multi-processor-stack-resource-policy (MSRP) to have a tighter upper bound on utilisations and reduced blocking times. The work used MT1 and considered factors such as number of critical sections and their sizes, blocking times, utilisation and synchronisation overheads. OS supports functionalities like IPC and device I/O management. This results in OS becoming a central point of conflict. Kim et al. (2018) presented a hardware isolation framework and examined conflicts due to OS sharing. The work also used MT1. Most of the works (Pellizzoni et al., 2010; Yun et al., 2012; Giannopoulou et al., 2013; Li and Wang, 2016; Hassan and Patel, 2016; Nair et al., 2019; Awan et al., 2017, 2018b; Dridi et al., 2019; Han et al., 2017; Li and He, 2017; Kim et al., 2018; Trüb et al., 2017) are evaluated using synthetic task sets, however Giannopoulou et al. (2013), Hassan and Patel (2016) and Trüb et al. (2017) used practical task sets from the avionic industry. Dridi et al. (2019), Kim et al. (2018) and Trüb et al. (2017) evaluated the work using benchmark programs like Rosace, Matrix, Framecopy, etc.

One of the widely addressed topics in multi-core MCS is resource provisioning. However, effective and efficient utilisation of resources without hampering certification requirement continues to be a fascinating challenge in MCS. Therefore, future works on resource and communication related parameters for different usage scenarios such as cache/resource synchronisation/message passing are expected to thrive.

4.3 *QoS related task models*

In MCS, mode switching takes place either by observing execution time of the currently running job or by observing the rate of arrival of the newly arriving job. This section surveys multi-core MCS task models which support various mode switching mechanisms. It also surveys mechanisms to improve schedulability of LC tasks in multi-core MCS along with their task models.

The effectiveness of a system depends on how many LC tasks are getting accommodated along with HC tasks. Various methods have been proposed to improve the QoS of the system. The four commonly used techniques are elastic scheduling which stretches the periods/execution times of LC jobs in high mode, priority reduction of LC jobs, imprecise computing which allows LC jobs to execute with lower computing times and variable precision scheduling which uses slack for LC job completions in high mode. Table 3 (MT4, MT6, MT11–MT16, MT21–MT24, MT27, MT31, MT32) presents task models with QoS parameters and their usage scenarios. To provide an enhanced service to LC tasks in high mode, Xu and Burns (2015) explored the possibility of migrating LC jobs to other cores that have not undergone a mode change in a dual-core system. Xu and Burns (2019) extended the migration capability to $\log_2 n$ cores, where n is the number of cores. Ren and Phan (2015) grouped one HC task with many LC tasks to provide best effort service to LC tasks during mode change. They also proposed mode relapse if all HC jobs displayed LC behaviour. Huang et al. (2018) compared global and partitioned scheduling and found that partitioned scheduling with variable precision accommodates more LC jobs thus improved QoS. Xu and Burns (2015), Ren and Phan (2015), Huang et al. (2018) and Xu and Burns (2019) used the basic task model (MT1).

Another approach to improve schedulability of LC jobs was proposed by Giannopoulou et al. (2013) and Trüb et al. (2017) (MT4). A degraded WCET parameter is appended to the task model to allow lower criticality jobs to continue execution in degraded mode, hence improving their schedulability. Hassan and Patel (2016) (MT6) used dynamic run time control for delaying the switching to degraded mode. This resulted in accepting more LC jobs into the system. Jan et al. (2013) (MT11) used an elastic model for stretching periods of LC jobs when they were about to miss their deadlines. Two parameters namely, importance level (I) and stretching factor (S) are added to the basic task model of LC tasks. A similar approach for handling LC tasks during a mode change was proposed in Su et al. (2013) (MT12). Besides stretching the period, a set of early release points are associated with LC jobs to improve their schedulability. Gill et al. (2018) (MT32) also used the elastic task model for parallel tasks to improve LC job schedulability. The work used VD to detect task overruns rather than computation times. In this model, utilisations are regulated to achieve elasticity. De Niz and Phan (2014) (MT13) computed ZERO slack instant of tasks in each mode and during mode change. During a mode change, LC jobs continue executing until they reach the zero slack instant of HC jobs. In this task model WCET is considered under non-overloaded and overloaded conditions. Al-Bayati et al. (2015) (MT14) proposed a twin partitioning algorithm. One task assignment is provided for low mode and another for high mode. Before a mode change, tasks are fully partitioned based on the assignment in low mode. During a mode change, LC tasks are allowed to have restricted migration based on the assignment in high mode. Pathan (2017) (MT15) used QoS vector ($\overrightarrow{V}$) during mode change to determine whether LC jobs will execute in normal manner or degraded manner. Hu et al. (2018) (MT16) suggested a semi-slack approach to guarantee service to LC jobs during HC overruns. Thekkilakattil et al. (2014) (MT21) proposed a slack-based approach to improve the feasibility of LC tasks. Caplan et al. (2017) (MT22) introduced low/overrun/transient-fault/high modes to handle faults and pre-configured a subset of LC tasks which are allowed to run in each mode. Choi et al. (2018) (MT23) represented graph-based task model and categorised droppable LC tasks based on importance (I), dropping factor (d) and QoS (V) parameters. This task model considered the number of task activations per interval ($\overline{\alpha}^u(\Delta)$) or time interval ($\delta(q)$) for slack computation. Iacovelli et al. (2018) (MT24) proposed utility (u) and tolerance aware LC jobs dropping based on core failure. On recovery, dropped tasks were reassigned based on core utility. In Medina et al. (2018) (MT27), DAG representation of task model for a schedule table was presented. The work proposes the following QoS improvements: successor-only dropping due to LC job overruns, LC mode relapse at the beginning of each schedule table execution and allow few HC overruns, while keeping the system intact. Pathan (2018) (MT31) proposed improvement of QoS by reassigning HC jobs to unused processors. LC jobs are classified as droppable and non-droppable using integer linear programming approach. The works (Jan et al., 2013; Su et al., 2013; De Niz and Phan, 2014; Pathan, 2017; Al-Bayati et al., 2015; Giannopoulou et al., 2013; Hassan and Patel, 2016; Xu and Burns, 2015; Caplan et al., 2017; Hu et al., 2018; Choi et al., 2018; Pathan, 2018; Medina et al., 2018) were evaluated using synthetic task sets while Giannopoulou et al. (2013), Hassan and Patel (2016) and Choi et al. (2018) were evaluated using practical task sets from the automotive and avionic industries.

Multi-core systems provide a clear advantage of implementing QoS improvement mechanisms by utilising high computing power with isolation. Industry relevant QoS mechanisms like symmetric clustering, LC mode relapses by importance, etc. are not clearly known to academia due to its intellectual property constraints. There exist a dire need to have domain relevant studies to overcome this gap.

4.4 Virtualisation related task models

One of the initial works for limiting interference and providing isolation in multi-core MCS was proposed by Anderson et al. (2009) (MT17). They put forth a hierarchical scheduling mechanism to carry out temporal isolation on multi-core MCS. They used a periodic task model with harmonic periods and five criticality levels. Each core uses five containers with varying criticality levels and higher criticality tasks are statically assigned higher priority. Each task is represented by parameters: period, relative deadline and WCET vector. An allocation window and a budget characterises each container. Table 3 (MT17, MT18) presents virtualisation related task models with their usage scenarios. Herman et al. (2012) extended the work using MT17 to consider OS overheads related to scheduling and release of jobs. Chisholm et al. (2016) proposed data sharing among criticality levels. The work also used MT17. An overhead cognisant schedulability study was presented. Lackorzyński et al. (2012) proposed improving performance in a hierarchical system by dynamically adjusting virtual machine (VM) budgets based on needs. Völp et al. (2013) proposed integration of existing mixed criticality scheduling algorithms like PC, SMC and variants, AMC and variants, OCBP and EDF-VD to schedule OS contexts in the hypervisor. To guarantee safety and security in MCS, Woolley et al. (2020) proposed combining hierarchical scheduling and genetic algorithms to uniformly distribute tasks in a multi-core scenario. The works (Lackorzyński et al., 2012; Völp et al., 2013; Woolley et al., 2020) used MT1 as the task model. Groesbrink et al. (2013) (MT18) combined virtualisation, mixed criticality and heterogeneous systems to provide an adaptive resource management approach. In this work, the system is partitioned into criticality-based VMs. Each VM is defined by a VM profile consisting of criticality, min/max number of resources, quality, functions activate/main/leave and a set of switching profiles. Each task is also characterised by a set of profiles. Each task profile is defined by same set of VM profile parameters excluding criticality, as criticality is assigned to each VM. The works (Chisholm et al., 2016; Herman et al., 2012; Woolley et al., 2020) were evaluated using synthetic task sets. Lackorzyński et al. (2012) provided two case studies using FreeRTOS and Linux-RT.

Multi-core with hypervisor provides an integrated system that utilises computing resources effectively and additionally provides isolation capabilities required by MCS. They need extension with practical implementations of OS level virtualisation or real-time container techniques like Docker or Linux containers. Task models/profiles that suit real-time container techniques is a prime area of research.

4.5 Energy related task models

To take energy consumption into consideration Awan et al. (2015, 2016a) (MT19) augmented the basic task model with an energy consumption vector, with core-wise information of all tasks in low mode. The WCET in low and high modes are also represented as vectors with core-wise information. The work proposed partitioning tasks such that energy consumption is minimised. Table 3 (MT19, MT20) presents energy related task models with their usage scenarios. Awan et al. (2016b) (MT19) adopted a two step approach to reduce energy consumption. The first step proposed partitioning tasks to minimise dynamic energy consumption. The second step proposed re-assigning tasks so as to attain improved sleep states. Digalwar et al. (2017) presented an energy efficient scheduler for periodic and aperiodic tasks. The work aims at reducing static and dynamic energy consumption by using techniques like procrastination and DVFS. The analysis used MT1 for periodic tasks and arrival time and WCET for aperiodic tasks. A scheduler for reducing static energy consumption was presented by Legout et al. (2013) using MT1. Taherin et al. (2018) (MT20) enhanced the elastic task model (Su and Zhu, 2013) with a voltage and frequency scaling factor (ρ). The work extended early release EDF and evenly distributed slack time to provide energy management. The works (Awan et al., 2016a; Digalwar et al., 2017; Taherin et al., 2018; Legout et al., 2013; Awan et al., 2016b) were evaluated using synthetic task sets.

There is a huge potential to have energy optimisation/procrastination solutions with multi-core systems. The optimised leakage/standby/static/dynamic power is of prime importance in energy starving battery-less MCS. The positive steps in this direction require apt task parameters to carry out offline energy-aware schedulability analysis.

4.6 Fault tolerance related task models

For multi-core dual criticality systems, Thekkilakattil et al. (2014) (MT21) provided a fault tolerant task scheduling mechanism by extending MT1 with replication (r) and distribution (m) parameters. r and m are used to specify the amount of replication required and the distribution of replicas on different nodes respectively. The work provided fault tolerance and certification for HC tasks with graceful degradation of LC tasks. Table 3 (MT21–MT24) presents fault tolerance related task models with their usage scenarios. Caplan et al. (2017) (MT22) presented a mechanism for handling transient faults using a reliability parameter (n) that defined the number re-executions of a HC task. They extended AMC-rtb to accommodate low/overrun/transient/high modes with apt LC task configurations. Choi et al. (2018) (MT23) proposed an optimistic fault tolerant multi-core MCS and presented a worst-case response time analysis. The goal was to provide a tighter upper bound on response times by keeping track of the number of faults that can be tolerated. In this work, reliability constraint (R), fault detection (O^d), roll-back (O^r) and voting (O^v) parameters contribute to achieve fault tolerance in critical applications. Iacovelli et al. (2018) (MT24) proposed an adaptive task model to achieve fault tolerance. The basic task model is augmented with utility (u) parameter. In case of core failures, tasks are re-assigned to available cores using u and their frequencies are adjusted in order to ensure schedulability. Naghavi et al. (2021) addressed both fault tolerance and energy savings. They proposed semi-partitioned/partitioned mechanisms with/without job migrations, HC job replications and QOS for LC jobs. The work used MT1. The works in Caplan et al. (2017), Choi et al. (2018) and Naghavi et al. (2021) were evaluated using synthetic task sets.

The increase in autonomy of MCS results in fault tolerance being an important tenet. This triggers enhanced research and certification requirements. In turn, it requires additional validation and offline analysis of autonomous systems such as autonomous driving systems. To perform these domain specific schedulability analysis, additional task parameters related to fault tolerance are required.

4.7 Graph-based task models

Graph-based task models are well suited to illustrate inter/intra dependencies between tasks on different cores. Table 3 (MT23, MT25–MT27) presents DAG-based task models with their usage scenarios. Tamas-Selicean and Pop (2011) (MT25) presented a mechanism to optimally combine and schedule tasks of different criticalities on a heterogeneous system such that development costs were reduced. Each application is designed as a DAG where vertices represent tasks and edges represent precedence among tasks. Each task is characterised by core level WCET, each edge is denoted by the size of messages transmitted and each application DAG is denoted by a period, deadline and criticality. Baruah (2013) (MT26) represented a resource- efficient synchronous execution of tasks using a DAG, where vertices represented jobs and edges represented inter-job dependencies. Each job was characterised by a WCET vector and it had deadline associated with it. Medina et al. (2018) (MT27) used a DAG-based model to depict task dependencies and limit fault propagation due to task overruns. In this model, vertices represented tasks with criticality, WCET and overrun probability (TFE) as parameters. The same task model was used in Medina et al. (2020) where a safe transition meta heuristic was defined to improve QoS in a multi-level criticality scenario. The task model in Choi et al. (2018) (MT23) is also represented by a DAG. Each graph is associated with a period, relative deadline, criticality, reliability constraint (R), importance (I), dropping factor (d) and QoS (V) as parameters. Each task in the graph has priority (P), WCET vector, fault detection overhead (O^d), roll-back overhead (O^r), voting overhead (O^v), the core (map) the task is assigned and the graph (G) it belongs as parameters.

The works in Medina et al. (2018), Choi et al. (2018), Tamas-Selicean and Pop (2011) and Medina et al. (2020) were evaluated using synthetic task sets. Additionally, Tamas-Selicean and Pop (2011) evaluated their model using consumer/networking/telecom-cords practical benchmarks. The representational graph-based models existing in literature are fork-join model (Lakshmanan et al., 2010) and sporadic/multi/conditional (Baruah et al., 2012; Fonseca et al., 2015; Melani et al., 2015) DAG tasks models. Graph-based representations depict module interactions between tasks within and between cores. They are also suitable for representing shared resources, migration mechanisms, task overruns, OS overheads, mode changes, and fault tolerance. The availability of such graph-based task models that consider real world applications ease the job of MCS designers.

4.8 Parallel task models

The introduction of multi-core architecture brings ample opportunities to implement parallel processing. Table 3 (MT28–MT32) presents parallel task models with their usage scenarios. Liu et al. (2014) (MT28) put forward an approach to schedule jobs of same task across cores simultaneously. In this work, jobs were characterised by release time, criticality, absolute deadline and a sequence of segments (J^1, J^2, ..., J^{Si}). Each job segment is defined by the number of threads and WCET vector. Bhuiyan et al. (2019) (MT29) proposed global EDF-VD to schedule parallel tasks. The basic task model is augmented with number of cores (Ⲙ) required for task execution. Subsequently, Bhuiyan et al. (2021) extended this task model for global EDF-VD where the number of cores considered is a vector $\overrightarrow{\text{Ⲙ}}$ that defines the core requirement at each criticality level. Agrawal and Baruah (2018) and Pathan (2018) (MT30, MT31) proposed work and span parameters to represent parallel tasks. Work parameter is a vector with overload/non-overload values that indicate the total WCET of parallel threads executing on all cores. Span vector is the critical path length of execution times in overload/non-overload conditions. Agrawal and Baruah (2018) considered an implicit deadline system whereas Pathan (2018) also considered criticality parameter. Gill et al. (2018) (MT32) proposed a task model for parallel tasks by considering a super set of parameters, namely criticality, work, span and number of cores ($\overrightarrow{\mathcal{M}}$). Unlike Bhuiyan et al. (2019) and Gill et al. (2018) considered $\overrightarrow{\mathcal{M}}$ and extended the task model with a stretch factor (S) to facilitate elasticity to improve QoS of LC tasks.

The works (Liu et al., 2014; Bhuiyan et al., 2019; Agrawal and Baruah, 2018; Pathan, 2018; Bhuiyan et al., 2021) were evaluated using synthetic task sets. To achieve high performance computing in MCS, as much parallelism as possible needs to be facilitated. The analysis and validation of parallel systems is a complex endeavour and requires apt task models to make it feasible.

5 Grouping of task models

Based on the study of numerous task models, we present a ready reckoner and graphical representation of task models for both uni-core and multi-core MCS.

5.1 Grouping task models – attributes wise

Tables 4 and 5 present a ready reckoner on uni-core and multi-core task models. Each column in the table indicates attributes, application domains and challenges addressed. Table 4 has uni-core task models coded from UT1 to UT31 and Table 5 has multi-core task models coded from MT1 to MT32. The attributes in Table 4 include resources, QoS, OS overheads, energy and fault tolerance. Table 5 has additional features of parallel processing and virtualisation for multi-core models. Application domains like avionics (avi), automotives (auto), cyber physical systems (CPS), wireless sensor networks (WSN), internet of things (IoT), components of the shelf (COTS), railways (rail), medical and robotics are the listed domains in literature for which the task models were designed. These application domains are tabulated as part of both uni-core and multi-core task models. Even though certification requirements vary among these application domains, there are many similarities in terms of their functional attributes. Hence, these task models are applicable across all domains.

Tables 4 and 5 also provide challenges (Section 2.1) that are partially addressed with respect to specific task models. Though attempts have been made by Esper et al. (2015) and Ekberg et al. (2013), challenge [C1] on criticality definition continues to remain majorly unresolved. The tabular ready reckoner is an useful asset for researchers and designers in MCS domain. It provides a feasible mechanism to search appropriate task models that suit given usage scenarios like resources, QoS, OS overheads, energy, fault tolerance, virtualisation and parallel systems. A designer/academician can identify task parameters or task models that are required for a given specifications from the extensive task models presented in Tables 4 and 5.

Table 4 Grouping of uni-core task models

Code	*Attributes*					*Domains addressed*	*Challenges*
	Resources	*QoS*	*OS overheads*	*Energy*	*Fault tolerance*		
UT1	-	√	-	-	-	Avi, auto	C3, C5, C6
UT2	√	√	-	-	-	Avi	C3, C5
UT3	-	√	-	-	-	Avi	C3, C5
UT4	√	√	-	-	-	Avi	C3, C5, C6, C10
UT5	√	-	-	-	-	General	C3, C5, C10
UT6	√	-	-	-	-	Avi, auto	C3, C5, C10
UT7	√	-	-	-	-	Avi, auto	C3, C5, C10
UT8	√	-	-	-	-	Avi, auto	C3, C5, C10
UT9	√	-	-	-	-	Auto	C3, C5, C10
UT10	-	√	-	-	-	CPS, avi	C3, C5, C6
UT11	-	√	-	-	-	CPS	C3, C5, C6
UT12	-	√	-	-	-	Avi, auto	C3, C5, C6
UT13	-	√	-	-	-	Avi, auto	C3, C5, C6
UT14	-	√	-	-	-	Avi, auto	C3, C5, C6
UT15	-	√	-	-	-	Avi, auto	C3, C5, C6
UT16	-	√	-	-	-	Avi, auto	C3, C5, C6
UT17	-	√	-		-	Avi	C3, C5, C6
UT18	-	√	-	-	-	Avi	C3, C5, C6
UT19	-	√	√	-	-	Auto	C3, C5, C6, C11
UT20	-	-	√	-	-	Avi, auto	C3, C5, C11
UT21	-	-	-	√	-	Avi	C3, C5, C7
UT22	-	-	-	√	-	Avi	C3, C5, C7
UT23	-	√	-	√	-	WSN, IoT	C3, C5, C6, C9
UT24	-	-	-	-	√	Avi	C3, C4, C5, C8
UT25	-	-	-	-	√	Avi	C3, C5, C8
UT26	-	-	-	-	√	Avi, auto	C3, C5, C8
UT27	√	-	-	-	-	Avi, auto	C3, C4, C5
UT28	-	√	-	-	-	General	C3, C4, C5
UT29	-	√	-	-	-	Avi	C3, C4, C5
UT30	-	-	√	-	-	Auto	C3, C4, C5
UT31	-	√	-	-	-	Avi	C3, C4, C5

5.2 Grouping of task models – graphical view

The graphical representation of all task models in uni-core and multi-core systems are shown in Figures 1 and 2 respectively. The vertex in the graph represents the task model and directed edges indicate parameters that are included in the subsequent model. Each vertex is colour coded depending on the functional features and representation. This graphical representation provides a pictorial view and displays the relationship between available task models. For example, in Figure 1, UT1 is the basic task model with parameters T; D; L; $\overrightarrow{C}$. It is extended to UT2 by modifying the deadline parameter to

$\overrightarrow{D}$. It is further extended with parameter $\overrightarrow{\lambda}$ to form UT7. Similarly in Figure 2, MT1 is extended to MT14 with $\overrightarrow{T}$. It is further extended to MT12 by adding parameter P^{ER} and finally to MT20 by adding ρ.

Table 5 Grouping of multi-core task models

Code	*Attributes*						*Domain addressed*	*Challenges*
	Resource	*QoS*	*Virtualisation*	*Energy*	*Fault tolerance*	*Parallel*		
MT1	√	√	√	√	-	-	Avi, auto	C5, C6, C9, C10, C11
MT2	√	-	-	-	-	-	COTS	C3, C5, C9, C10
MT3	√	-	-	-	-	-	Avi	C3, C5, C9, C10
MT4	√	√	-	-	-	-	Avi, auto	C3, C5, C6, C9, C10
MT5	√	-	-	-	-	-	Avi, auto	C3, C5, C9, C10
MT6	√	√	-	-	-	-	Avi	C3, C5, C6, C9, C10
MT7	√	-	-	-	-	-	Auto	C3, C5, C9, C10
MT8	√	-	-	-	-	-	Avi, auto	C3, C5, C9, C10
MT9	√	-	-	-	-	-	Avi, auto, rail	C3, C5, C9, C10
MT10	√	-	-	-	-	-	Auto	C3, C5, C9, C10
MT11	-	√	-	-	-	-	Auto	C3, C5, C6, C9
MT12	-	√	-	-	-	-	Avi	C3, C5, C6, C9
MT13	-	√	-	-	-	-	Avi, auto, CPS	C3, C5, C6, C9
MT14	-	√	-	-	-	-	Avi, auto	C3, C5, C6, C9
MT15	-	√	-	-	-	-	Auto	C3, C5, C6, C9
MT16	-	√	-	-	-	-	Auto, robotics	C3, C5, C6, C9
MT17	-	-	√	-	-	-	Avi	C3, C5, C9, C11
MT18	-	-	√	-	-	-	CPS	C3, C5, C9, C11
MT19	-	-	-	√	-	-	Avi, auto	C3, C5, C7, C9
MT20	-	-	-	√	-	-	Avi, auto	C3, C5, C7, C9
MT21	-	√	-	-	√	-	General	C3, C5, C6, C8, C9
MT22	-	√	-	-	√	-	Avi, auto	C3, C5, C6, C8, C9
MT23	-	√	-	-	√	-	Auto	C3, C4, C5, C6, C7, C8
MT24	-	√	-	-	√	-	Rail, medical	C3, C5, C6, C8, C9
MT25	√	-	-	-	-	√	Avi	C3, C4, C5, C9
MT26	√	-	-	-	-	√	Avi, auto	C3, C4, C5, C9
MT27	-	√	-	-	-	√	Avi	C3, C4, C5, C6, C9
MT28	-	-	-	-	-	√	Avi, auto, robotics	C3, C5, C9, C12
MT29	-	-	-	-	-	√	Avi	C3, C5, C9, C12
MT30	-	-	-	-	-	√	Avi, auto	C3, C5, C9, C12
MT31	-	√	-	-	-	√	Avi, auto	C3, C5, C6, C9, C12
MT32	-	√	-	-	-	√	CPS	C3, C5, C6, C9, C12

6 Conclusions and future directions

This work consolidates existing task models in literature and provides insights and future trends based on attributes such as resources, QoS, OS overheads, energy, fault tolerance and parallel processing. It also presents a ready reckoner and a graphical representation an easy visual aid for future researchers and industrial designers in the MCS domain.

By archiving and analysing MCS task models, it is observed that there is need of a collaboration and research dissemination framework in MCS research. The framework should consist of tools for archiving, generating and comparing task models/task sets. It should also have a mechanism to generate schedules and perform analysis, thereby aiding as a decision support system.

Figure 1 Uni-core task model (see online version for colours)

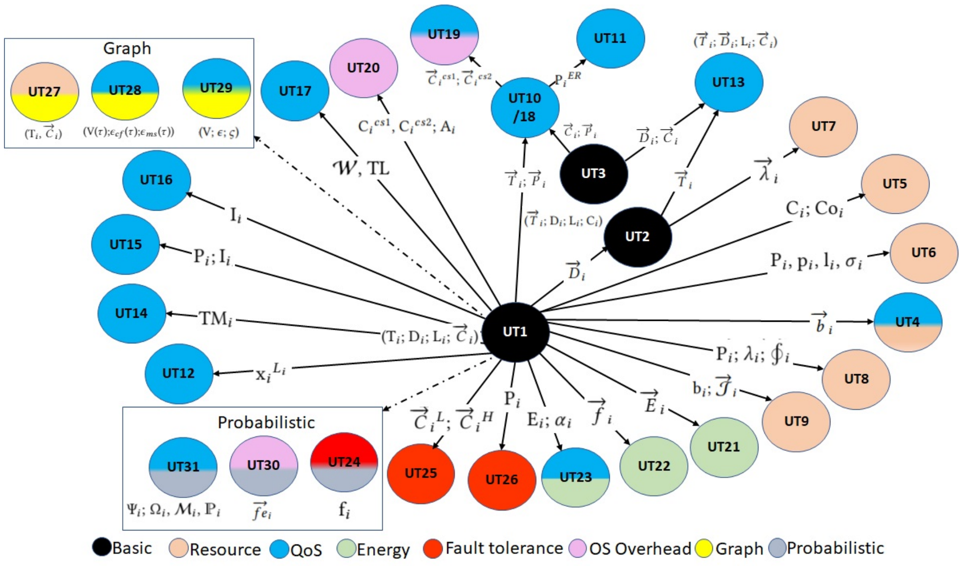
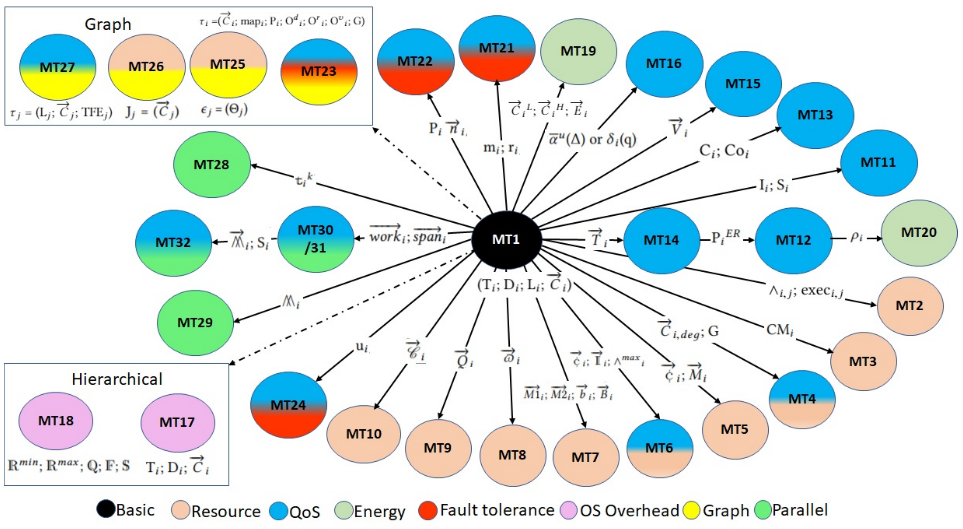

Graph
UT27
UT28
UT29
$(T_i, \vec{C}_i)$
$(V(\tau); \epsilon_{cf}(\tau); \epsilon_{ms}(\tau))$
$(V; \epsilon; \varsigma)$
UT1
UT2
UT3
UT4
UT5
UT6
UT7
UT8
UT9
UT10 /18
UT11
UT12
UT13
UT14
UT15
UT16
UT17
UT19
UT20
UT21
UT22
UT23
UT25
UT26
$\vec{C}_i^{cs1}; \vec{C}_i^{cs2}$
P_i^{ER}
$(\vec{T}_i; \vec{D}_i; L_i; \vec{C}_i)$
$C_i^{cs1}, C_i^{cs2}; A_i$
$\vec{C}_i; \vec{P}_i$
$\vec{D}_i; \vec{C}_i$
$\mathcal{W}$, TL
$\vec{T}_i; \vec{P}_i$
$\vec{T}_i$
$\vec{\lambda}_i$
$(\vec{T}_i; D_i; L_i; C_i)$
$C_i; Co_i$
I_i
$P_i; I_i$
P_i, p_i, l_i, σ_i
$\vec{D}_i$
TM_i
$(T_i; D_i; L_i; \vec{C}_i)$
$\vec{b}_i$
$P_i; \lambda_i; \oint_i$
$x_i^{L_i}$
$b_i; \mathcal{J}_i$
$\vec{E}_i$
P_i
$\vec{C}_i^L; \vec{C}_i^H$
$E_i; \alpha_i$
$\vec{f}_i$
Probabilistic
UT31
UT30
UT24
$\Psi_i; \Omega_i, \mathcal{M}_i, P_i$
$\vec{fe}_i$
f_i
●Basic ●Resource ●QoS ●Energy ●Fault tolerance ●OS Overhead ●Graph ●Probabilistic

Figure 2 Multi-core task model (see online version for colours)


Graph
$\tau_i = (\vec{C}_i; map_i; P_i; O^d_i; O^r_i; O^v_i; G)$
MT27
MT26
MT25
MT23
$\tau_j = (L_j; \vec{C}_j; TFE_j)$
$J_j = (\vec{C}_j)$
$\epsilon_j = (\Theta_j)$
MT1
MT2
MT3
MT4
MT5
MT6
MT7
MT8
MT9
MT10
MT11
MT12
MT13
MT14
MT15
MT16
MT19
MT20
MT21
MT22
MT24
MT28
MT29
MT30 /31
MT32
$P_i \vec{n}_i$
$m_i; r_i$
$\vec{C}_i^L; \vec{C}_i^H; \vec{E}_i$
$\overline{\alpha}^u(\Delta)$ or $\delta_i(q)$
$\vec{V}_i$
$C_i; Co_i$
$I_i; S_i$
t_i^k
$\vec{T}_i$
P_i^{ER}
ρ_i
$work_i; span_i$
$\vec{\mathbb{M}}_i; S_i$
$(T_i; D_i; L_i; \vec{C}_i)$
$\wedge_{i,j}; exec_{i,j}$
CM_i
$\mathbb{M}_i$
$\vec{C}_{i,deg}; G$
u_i
$\vec{\mathscr{C}}_i$
$\vec{Q}_i$
$\vec{\varpi}_i$
$\vec{\varsigma}_i; \vec{\mathbb{T}}_i; \wedge^{max}_i$
$\vec{M1}_i; \vec{M2}_i; \vec{b}_i; \vec{B}_i$
$\vec{\varsigma}_i; \vec{M}_i$
Hierarchical
MT18
MT17
$\mathbb{R}^{min}; \mathbb{R}^{max}; Q; F; S$
$T_i; D_i; \vec{C}_i$
●Basic ●Resource ●QoS ●Energy ●Fault tolerance ●OS Overhead ●Graph ●Parallel

A careful scrutiny of existing task models also indicates lack of task models with certain combinations. Few missing entries include combination of fault tolerance and parallel task models, combination of probabilistic and graph tasks models in tandem with functional specific attributes like resources, energy and QoS. It is also observed that certification of MCS for pioneering features like parallel computing, fault tolerance and energy efficiency are complex endeavours. Likewise, the analysis of OS-level-virtualisation/real-time-container techniques and real world task models that consider realistic factors found in industrial applications add another dimension to the complexity of MCS.

References

Abdeddaïm, Y. (2020) 'Accurate strategy for mixed criticality scheduling', in *International Conference on Verification and Evaluation of Computer and Communication Systems*, Springer, pp.131–146.

Abella, J., Hernandez, C., Quiñones, E., Cazorla, F.J., Conmy, P.R., Azkarate-Askasua, M., Perez, J., Mezzetti, E. and Vardanega, T. (2015) 'WCET analysis methods: pitfalls and challenges on their trustworthiness', in *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, IEEE, pp.1–10.

Agrawal, K. and Baruah, S. (2018) 'A measurement-based model for parallel real-time tasks', in *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Agrawal, K., Baruah, S. and Burns, A. (2019) 'Semi-clairvoyance in mixed-criticality scheduling', in *2019 IEEE Real-Time Systems Symposium (RTSS)*, IEEE, pp.458–468.

Alahmad, B.N. and Gopalakrishnan, S. (2018) 'Risk-aware scheduling of dual criticality job systems using demand distributions', *Leibniz Transactions on Embedded Systems*, Vol. 5, No. 1, p.1.

Alahmad, B., Gopalakrishnan, S., Santinelli, L. and Cucu-Grosjean, L. (2011) 'Probabilities for mixed-criticality problems: bridging the uncertainty gap', *WiP, RTSS*, pp.1–4.

Al-Bayati, Z., Zhao, Q., Youssef, A., Zeng, H. and Gu, Z. (2015) 'Enhanced partitioned scheduling of mixed-criticality systems on multicore platforms', in *The 20th Asia and South Pacific Design Automation Conference*, IEEE, pp.630–635.

Anderson, J.H., Baruah, S.K. and Brandenburg, B.B. (2009) 'Multicore operating-system support for mixed criticality', in *Proceedings of the Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification*, Citeseer, Vol. 4, p.7.

Asyaban, S., Kargahi, M., Thiele, L. and Mohaqeqi, M. (2016) 'Analysis and scheduling of a battery-less mixed-criticality system with energy uncertainty', *ACM Transactions on Embedded Computing Systems (TECS)*, Vol. 16, No. 1, p.23.

Awan, M.A., Masson, D. and Tovar, E. (2015) 'Energy-aware task allocation onto unrelated heterogeneous multicore platform for mixed criticality systems', in *2015 IEEE Real-Time Systems Symposium*, IEEE, pp.377–377.

Awan, M.A., Masson, D. and Tovar, E. (2016a) 'Energy efficient mapping of mixed criticality applications on unrelated heterogeneous multicore platforms', in *2016 11th IEEE Symposium on Industrial Embedded Systems (SIES)*, IEEE, pp.1–10.

Awan, M.A., Yomsi, P.M., Nelissen, G. and Petters, S.M. (2016b) 'Energy-aware task mapping onto heterogeneous platforms using DVFS and sleep states', *Real-Time Systems*, Vol. 52, No. 4, pp.450–485.

Awan, M.A., Bletsas, K., Souto, P.F., Akesson, B. and Tovar, E. (2017) *Mixed-Criticality Scheduling with Dynamic Redistribution of Shared Cache*, arXiv preprint arXiv:1704.08876.

Awan, M.A., Souto, P.F., Bletsas, K., Akesson, B. and Tovar, E. (2018a) 'Mixed-criticality scheduling with memory bandwidth regulation', in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, pp.1277–1282.

Awan, M.A., Bletsas, K., Souto, P.F., Akesson, B. and Tovar, E. (2018b) 'Mixed-criticality scheduling with dynamic memory bandwidth regulation', in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, IEEE, pp.111–117.

Axer, P., Sebastian, M. and Ernst, R. (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*, ACM, pp.149–158.

Baek, H. and Lee, K. (2020) 'Contention-free scheduling for mixed-criticality multiprocessor real-time system', *Symmetry*, Vol. 12, No. 9, p.1515.

Baruah, S. (2009) *Mixed Criticality Schedulability Analysis is Highly Intractable*, Technical Report, University of North Carolina at Chapel Hill.

Baruah, S. (2013) *Implementing Mixed Criticality Synchronous Reactive Systems upon Multiprocessor Platforms*, Tech. Rep., The University of North Carolina at Chapel Hill.

Baruah, S. (2014) 'Implementing mixed-criticality synchronous reactive programs upon uniprocessor platforms', *Real-Time Systems*, Vol. 50, No. 3, pp.317–341.

Baruah, S. and Burns, A. (2011) 'Implementing mixed criticality systems in ADA', in *International Conference on Reliable Software Technologies*, Springer, pp.174–188.

Baruah, S. and Burns, A. (2020) 'Expressing survivability considerations in mixed-criticality scheduling theory', *2019 IEEE Symposium on Real-time Computing ISORC*, Vol. 109, p.101755.

Baruah, S. and Chattopadhyay, B. (2013) 'Response-time analysis of mixed criticality systems with pessimistic frequency specification', in *2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications*, IEEE, pp.237–246.

Baruah, S., Li, H. and Stougie, L. (2010a) 'Towards the design of certifiable mixed-criticality systems', in *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, IEEE, pp.13–22.

Baruah, S.K., Bonifaci, V., D'Angelo, G., Li, H., Marchetti-Spaccamela, A., Megow, N. and Stougie, L. (2010b) 'Scheduling real-time mixed-criticality jobs', in *International Symposium on Mathematical Foundations of Computer Science*, Springer, pp.90–101.

Baruah, S.K., Li, H. and Stougie, L. (2010c) 'Mixed-criticality scheduling: improved resource-augmentation results', in *25th International Conference on Computers and their Applications Conference Proceedings*, pp.217–223.

Baruah, S.K., Burns, A. and Davis, R.I. (2011a) 'Response-time analysis for mixed criticality systems', in *2011 IEEE 32nd Real-Time Systems Symposium*, IEEE, pp.34–43.

Baruah, S.K., Bonifaci, V., D'Angelo, G., Marchetti-Spaccamela, A., Van Der Ster, S. and Stougie, L. (2011b) 'Mixed-criticality scheduling of sporadic task systems', in *European Symposium on Algorithms*, Springer, pp.555–566.

Baruah, S., Bonifaci, V., Marchetti-Spaccamela, A., Stougie, L. and Wiese, A. (2012) 'A generalized parallel task model for recurrent real-time processes', in *2012 IEEE 33rd Real-Time Systems Symposium*, IEEE, pp.63–72.

Baruah, S., Chattopadhyay, B., Li, H. and Shin, I. (2014) 'Mixed-criticality scheduling on multiprocessors', *Real-Time Systems*, Vol. 50, No. 1, pp.142–177.

Baruah, S., Bonifaci, V., D'Angelo, G., Li, H., Marchetti-Spaccamela, A., Van Der Ster, S. and Stougie, L. (2015a) 'Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems', *Journal of the ACM*, Vol. 62, No. 2, pp.1–33.

Baruah, S.K., Cucu-Grosjean, L., Davis, R.I. and Maiza, C. (2015b) 'Mixed criticality on multicore/manycore platforms (Dagstuhl Seminar 15121)', in *Dagstuhl Reports*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Vol. 5.

Baruah, S., Easwaran, A. and Guo, Z. (2015c) 'MC-fluid: simplified and optimally quantified', in *2015 IEEE Real-Time Systems Symposium*, IEEE, pp.327–337.

Behera, L. and Bhaduri, P. (2018) 'Time-triggered scheduling for multiprocessor mixed-criticality systems', in *International Conference on Distributed Computing and Internet Technology*, Springer, pp.135–151.

Bhuiyan, A.A., Yang, K., Arefin, S., Saifullah, A., Guan, N. and Guo, Z. (2019) 'Mixed-criticality multicore scheduling of real-time gang task systems', in *2019 IEEE Real-Time Systems Symposium (RTSS)*, IEEE, pp.469–480.

Bhuiyan, A., Reghenzani, F., Fornaciari, W. and Guo, Z. (2020) 'Optimizing energy in non-preemptive mixed-criticality scheduling by exploiting probabilistic information', *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 39, No. 11, pp.3906–3917.

Bhuiyan, A., Yang, K., Arefin, S., Saifullah, A., Guan, N. and Guo, Z. (2021) 'Mixed-criticality real-time scheduling of gang task systems', *Real-Time Systems*, Vol. 57, No. 3, pp.268–301.

Bletsas, K., Awan, M.A., Souto, P., Åkesson, B., Burns, A. and Tovar, E. (2018) 'Decoupling criticality and importance in mixed-criticality scheduling', in *6th International Workshop on Mixed Criticality Systems (WMC 2018)*, pp.25–30.

Boudjadar, J., Ramanathan, S., Easwaran, A. and Nyman, U. (2019) 'Combining task-level and system-level scheduling modes for mixed criticality systems', in *2019 IEEE/ACM 23rd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, IEEE, pp.1–10.

Brandenburg, B.B. (2014) 'A synchronous IPC protocol for predictable access to shared resources in mixed-criticality systems', in *2014 IEEE Real-Time Systems Symposium*, IEEE, pp.196–206.

Burns, A. (2013) 'The application of the original priority ceiling protocol to mixed criticality systems', *Proc. ReTiMiCS, RTCSA*, pp.7–11.

Burns, A. (2014) 'System mode changes-general and criticality-based', in *Proc. of 2nd Workshop on Mixed Criticality Systems (WMC)*, pp.3–8.

Burns, A. and Baruah, S. (2011) 'Timing faults and mixed criticality systems', in Jones, C.B. and Lloyd, J.L. (Eds.): *Dependable and Historic Computing. Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, Vol. 6875.

Burns, A. and Baruah, S. (2013) 'Towards a more practical model for mixed criticality systems', in *Workshop on Mixed-Criticality Systems (Colocated with RTSS).*

Burns, A. and Davis, R. (2013a) *Mixed Criticality Systems – A Review*, Tech. Rep., pp.1–69, Department of Computer Science, University of York.

Burns, A. and Davis, R.I. (2013b) 'Mixed criticality on controller area network', in *2013 25th Euromicro Conference on Real-Time Systems*, IEEE, pp.125–134.

Burns, A. and Davis, R.I. (2017) 'A survey of research into mixed criticality systems', *ACM Computing Surveys (CSUR)*, Vol. 50, No. 6, pp.1–37.

Burns, A., Davis, R.I., Baruah, S. and Bate, I. (2018) 'Robust mixed-criticality systems', *IEEE Transactions on Computers*, Vol. 67, No. 10, pp.1478–1491.

Caplan, J., Al-Bayati, Z., Zeng, H. and Meyer, B.H. (2017) 'Mapping and scheduling mixed-criticality systems with on-demand redundancy', *IEEE Transactions on Computers*, Vol. 67, No. 4, pp.582–588.

Chen, G., Guan, N., Hu, B. and Yi, W. (2018) 'EDF-VD scheduling of flexible mixed-criticality system with multiple-shot transitions', *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 37, No. 11, pp.2393–2403.

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

Choi, J., Yang, H. and Ha, S. (2018) 'Optimization of fault-tolerant mixed-criticality multi-core systems with enhanced WCRT analysis', *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, Vol. 24, No. 1, pp.1–26.

Chwa, H.S., Shin, K.G., Baek, H. and Lee, J. (2018) 'Physical-state-aware dynamic slack management for mixed-criticality systems', in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, IEEE, pp.129–139.

Davis, R.I. and Burns, A. (2011) 'A survey of hard real-time scheduling for multiprocessor systems', *ACM Computing Surveys (CSUR)*, Vol. 43, No. 4, p.35.

Davis, R.I., Altmeyer, S. and Burns, A. (2018) 'Mixed criticality systems with varying context switch costs', in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, IEEE, pp.140–151.

De Niz, D. and Phan, L.T.X. (2014) 'Partitioned scheduling of multi-modal mixed-criticality real-time systems on multiprocessor platforms', in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, IEEE, pp.111–122.

De Niz, D., Lakshmanan, K. and Rajkumar, R. (2009) 'On the scheduling of mixed-criticality real-time task sets', in *2009 30th IEEE Real-Time Systems Symposium*, IEEE, pp.291–300.

Digalwar, M., Raveendran, B.K. and Mohan, S. (2017) 'LAMCS: a leakage aware DVFS based mixed task set scheduler for multi-core processors', *Sustainable Computing: Informatics and Systems*, Vol. 15, No. Supplement C, pp.63–81, DOI: 10.1016/j.suscom.2017.06.001.

Dorin, F., Richard, P., Richard, M. and Goossens, J. (2010) 'Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities', *Real-Time Systems*, Vol. 46, No. 3, pp.305–331.

Dridi, M., Rubini, S., Lallali, M., Flórez, M.J.S., Singhoff, F. and Diguet, J-P. (2019) 'Design and multi-abstraction-level evaluation of a NOC router for mixed-criticality real-time systems', *ACM Journal on Emerging Technologies in Computing Systems*, Vol. 15, No. 1, pp.1–37.

Ekberg, P. and Yi, W. (2014) 'Bounding and shaping the demand of generalized mixed-criticality sporadic task systems', *Real-Time Systems*, Vol. 50, No. 1, pp.48–86.

Ekberg, P., Stigge, M., Guan, N. and Yi, W. (2013) 'State-based mode switching with applications to mixed criticality systems', *Proc. WMC, RTSS*, pp.61–66.

Esper, A., Nelissen, G., Nélis, V. and Tovar, E. (2015) 'How realistic is the mixed-criticality real-time system model?', in *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, pp.139–148.

Evripidou, C. (2016) *Scheduling for Mixed-Criticality Hypervisor Systems in the Automotive Domain*, PhD thesis, University of York.

Fleming, T. and Burns, A. (2014) 'Incorporating the notion of importance into mixed criticality systems', in *Proc. 2nd Workshop on Mixed Criticality Systems (WMC), RTSS*, pp.33–38.

Fonseca, J.C., Nélis, V., Raravi, G. and Pinho, L.M. (2015) 'A multi-dag model for real-time parallel applications with conditional execution', in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pp.1925–1932.

Gettings, O., Quinton, S. and Davis, R.I. (2015) 'Mixed criticality systems with weakly-hard constraints', in *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, ACM, pp.237–246.

Giannopoulou, G., Stoimenov, N., Huang, P. and Thiele, L. (2013) 'Scheduling of mixed-criticality applications on resource-sharing multicore systems', in *Proceedings of the Eleventh ACM International Conference on Embedded Software*, IEEE Press, p.17.

Gill, C., Orr, J. and Harris, S. (2018) 'Supporting graceful degradation through elasticity in mixed-criticality federated scheduling', in *Proc. 6th Workshop on Mixed Criticality Systems (WMC), RTSS*, pp.19–24.

Gratia, R., Robert, T. and Pautet, L. (2014) 'Adaptation of run to mixed-criticality systems', *JRWRTC '2014*.

Groesbrink, S., Oberthür, S. and Baldin, D. (2013) 'Architecture for adaptive resource assignment to virtualized mixed-criticality real-time systems', *ACM SIGBED Review*, Vol. 10, No. 1, pp.18–23.

Gu, X. and Easwaran, A. (2017) 'Efficient schedulability test for dynamic-priority scheduling of mixed-criticality real-time systems', *ACM Transactions on Embedded Computing Systems (TECS)*, Vol. 17, No. 1, pp.1–24.

Gu, X., Easwaran, A., Phan, K-M. and Shin, I. (2015) 'Resource efficient isolation mechanisms in mixed-criticality scheduling', in *2015 27th Euromicro Conference on Real-Time Systems*, IEEE, pp.13–24.

Guan, F., Peng, L., Perneel, L., Fayyad-Kazan, H. and Timmerman, M. (2017) 'A design that incorporates adaptive reservation into mixed-criticality systems', *Scientific Programming*, Vol. 2017, Article ID 3403685, pp.1–20, Hindawi Scientific Programming [online] https://doi.org/10.1155/2017/3403685.

Guan, N., Ekberg, P., Stigge, M. and Yi, W. (2011) 'Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems', in *2011 IEEE 32nd Real-Time Systems Symposium*, IEEE, pp.13–23.

Guo, Z. and Baruah, S.K. (2016) 'A neurodynamic approach for real-time scheduling via maximizing piecewise linear utility', *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 27, No. 2, pp.238–248.

Guo, Z., Santinelli, L. and Yang, K. (2015) 'EDF schedulability analysis on mixed-criticality systems with permitted failure probability', in *2015 IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications*, IEEE, pp.187–196.

Guo, Z., Santinelli, L. and Yang, K. (2018) 'Mixed-criticality scheduling with limited HI-criticality behaviors', in *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, Springer, pp.187–199.

Guo, Z., Vaidhun, S., Satinelli, L., Arefin, S., Wang, J. and Yang, K. (2021) 'Mixed-criticality scheduling upon permitted failure probability and dynamic priority', *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 41, No. 1, pp.62–75.

Gupta, T., Luit, E.J., van den Heuvel, M.M.H.P. and Bril, R.J. (2018) 'Experience report: towards extending an osek-compliant RTOs with mixed criticality support', *E-Informatica Software Engineering Journal*, Vol. 12, No. 1, pp.305–320, DOI: 10.5277/e-Inf180112.

Han, J-J., Tao, X., Zhu, D. and Yang, L.T. (2017) 'Resource sharing in multicore mixed-criticality systems: utilization bound and blocking overhead', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 28, No. 12, pp.3626–3641.

Hassan, M. (2017) *Heterogeneous MPSoCs for Mixed Criticality Systems: Challenges and Opportunities*, arXiv preprint arXiv:1706.07429.

Hassan, M. and Patel, H. (2016) 'Criticality-and requirement-aware bus arbitration for multi-core mixed criticality systems', in *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, IEEE, pp.1–11.

Herman, J.L., Kenna, C.J., Mollison, M.S., Anderson, J.H. and Johnson, D.M. (2012) 'RTOs support for multicore mixed-criticality systems', in *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, IEEE, pp.197–208.

Hu, B., Chen, G. and Huang, K. (2018) 'Semi-slack scheduling arbitrary activation patterns in mixed-criticality systems', *IEEE Access*, Vol. 6, pp.68507–68524, DOI: 10.1109/ACCESS.2018.2879717.

Huang, P., Kumar, P., Stoimenov, N. and Thiele, L. (2013) 'Interference constraint graph – a new specification for mixed-criticality systems', in *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, IEEE, pp.1–8.

Huang, P., Kumar, P., Giannopoulou, G. and Thiele, L. (2014a) 'Energy efficient DVFS scheduling for mixed-criticality systems', in *Proceedings of the 14th International Conference on Embedded Software*, ACM, p.11.

Huang, P., Yang, H. and Thiele, L. (2014b) 'On the scheduling of fault-tolerant mixed-criticality systems', in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, IEEE, pp.1–6.

Huang, P., Giannopoulou, G., Stoimenov, N. and Thiele, L. (2014c) 'Service adaptions for mixed-criticality systems', in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, pp.125–130.

Huang, L., Hou, I-H., Sapatnekar, S.S. and Hu, J. (2018) 'Graceful degradation of low-criticality tasks in multiprocessor dual-criticality systems', in *Proceedings of the 26th International Conference on Real-Time Networks and Systems*, pp.159–169.

Iacovelli, S., Kirner, R. and Menon, C. (2018) 'ATMP: an adaptive tolerance-based mixed-criticality protocol for multi-core systems', in *2018 IEEE 13th International Symposium on Industrial Embedded Systems (SIES)*, IEEE, pp.1–9.

Jan, M., Zaourar, L. and Pitel, M. (2013) 'Maximizing the execution rate of low criticality tasks in mixed criticality system', *Proc. WMC, RTSS*, pp.43–48.

Kahil, R., Socci, D., Poplavko, P. and Bensalem, S. (2018) 'Algorithmic complexity of correctness testing in MC-scheduling', in *Proceedings of the 26th International Conference on Real-Time Networks and Systems*, pp.180–190.

Kim, N., Tang, S., Otterness, N., Anderson, J.H., Smith, F.D. and Porter, D.E. (2018) 'Supporting I/O and IPC via fine-grained OS isolation for mixed-criticality real-time tasks', in *Proceedings of the 26th International Conference on Real-Time Networks and Systems*, pp.191–201.

Lackorzyński, A., Warg, A., Völp, M. and Härtig, H. (2012) 'Flattening hierarchical scheduling', in *Proceedings of the Tenth ACM International Conference on Embedded Software*, pp.93–102.

Lakshmanan, K., de Niz, D. and Rajkumar, R. (2011) 'Mixed-criticality task synchronization in zero-slack scheduling', in *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, IEEE, pp.47–56.

Lakshmanan, K., Kato, S. and Rajkumar, R. (2010) 'Scheduling parallel real-time tasks on multi-core processors', in *2010 31st IEEE Real-Time Systems Symposium*, IEEE, pp.259–268.

Lee, J., Chwa, H.S., Phan, L.T.X., Shin, I. and Lee, I. (2017) 'MC-ADAPT: adaptive task dropping in mixed-criticality scheduling', *ACM Transactions on Embedded Computing Systems (TECS)*, Vol. 16, No. 5s, pp.1–21.

Lee, J., Ramanathan, S., Phan, K-M., Easwaran, A., Shin, I. and Lee, I. (2017) 'MC-fluid: multi-core fluid-based mixed-criticality scheduling', *IEEE Transactions on Computers*, Vol. 67, No. 4, pp.469–483.

Lee, J., Phan, K-M., Gu, X., Lee, J., Easwaran, A., Shin, I. and Lee, I. (2014) 'MC-fluid: fluid model-based mixed-criticality scheduling on multiprocessors', in *2014 IEEE Real-Time Systems Symposium*, IEEE, pp.41–52.

Legout, V., Jan, M. and Pautet, L. (2013) 'Mixed-criticality multiprocessor real-time systems: energy consumption vs. deadline misses', in *First Workshop on Real-time Mixed Criticality Systems (ReTiMiCS)*, pp.1–6.

Li, H. and Baruah, S. (2010a) 'An algorithm for scheduling certifiable mixed-criticality sporadic task systems', in *2010 31st IEEE Real-Time Systems Symposium*, IEEE, pp.183–192.

Li, H. and Baruah, S. (2010b) 'Load-based schedulability analysis of certifiable mixed-criticality systems', in *Proceedings of the Tenth ACM International Conference on Embedded Software*, ACM, pp.99–108.

Li, H. and Baruah, S. (2012) 'Outstanding paper award: global mixed-criticality scheduling on multiprocessors', in *2012 24th Euromicro Conference on Real-Time Systems*, IEEE, pp.166–175.

Li, Z. and He, S. (2017) 'Fixed-priority scheduling for two-phase mixed-criticality systems', *ACM Transactions on Embedded Computing Systems (TECS)*, Vol. 17, No. 2, pp.1–20.

Li, Z. and Wang, L. (2016) 'Memory-aware scheduling for mixed-criticality systems', in *International Conference on Computational Science and Its Applications*, Springer, pp.140–156.

Liu, C.L. and Layland, J.W. (1973) 'Scheduling algorithms for multiprogramming in a hard-real-time environment', *Journal of the ACM*, Vol. 20, No. 1, pp.46–61.

Liu, G., Lu, Y., Wang, S. and Gu, Z. (2014) 'Partitioned multiprocessor scheduling of mixed-criticality parallel jobs', in *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, IEEE, pp.1–10.

Mahdiani, M. and Masrur, A. (2018) 'On bounding execution demand under mixed-criticality EDF', in *Proceedings of the 26th International Conference on Real-Time Networks and Systems*, pp.170–179.

Medina, R., Borde, E. and Pautet, L. (2018) 'Availability enhancement and analysis for mixed-criticality systems on multi-core', in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, pp.1271–1276.

Medina, R., Borde, E. and Pautet, L. (2020) 'Generalized mixed-criticality static scheduling for periodic directed acyclic graphs on multi-core processors', *IEEE Transactions on Computers*, Vol. 70, No. 3, pp.457–470.

Melani, A., Bertogna, M., Bonifaci, V., Marchetti-Spaccamela, A. and Buttazzo, G.C. (2015) 'Response-time analysis of conditional dag tasks in multiprocessor systems', in *2015 27th Euromicro Conference on Real-Time Systems*, IEEE, pp.211–221.

Mollison, M.S., Erickson, J.P., Anderson, J.H., Baruah, S.K. and Scoredos, J.A. (2010) 'Mixed-criticality real-time scheduling for multicore systems', in *2010 10th IEEE International Conference on Computer and Information Technology*, IEEE, pp.1864–1871.

Nagalakshmi, K. and Gomathi, N. (2018) 'Criticality-cognizant clustering-based task scheduling on multicore processors in the avionics domain', *International Journal of Computational Intelligence Systems*, Vol. 11, No. 1, pp.219–237.

Naghavi, A., Safari, S. and Hessabi, S. (2021) 'Tolerating permanent faults with low-energy overhead in multicore mixed-criticality systems', *IEEE Transactions on Emerging Topics in Computing*, Vol. 10, No. 2, pp.985–996.

Nair, A.S., Colaco, L.M., Patil, G., Raveendran, B.K. and Punnekkatt, S. (2019) 'Mediator-a mixed criticality deadline honored arbiter for multi-core real-time systems', in *2019 IEEE/ACM 23rd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, IEEE, pp.1–8.

Pathan, R.M. (2012) 'Schedulability analysis of mixed-criticality systems on multiprocessors', in *2012 24th Euromicro Conference on Real-Time Systems*, IEEE, pp.309–320.

Pathan, R.M. (2014) 'Fault-tolerant and real-time scheduling for mixed-criticality systems', *Real-Time Systems*, Vol. 50, No. 4, pp.509–547.

Pathan, R.M. (2017) 'Improving the quality-of-service for scheduling mixed-criticality systems on multiprocessors', in *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Pathan, R.M. (2018) 'Improving the schedulability and quality of service for federated scheduling of parallel mixed-criticality tasks on multiprocessors', in *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Pellizzoni, R., Schranzhofer, A., Chen, J-J., Caccamo, M. and Thiele, L. (2010) 'Worst case delay analysis for memory interference in multicore systems', in *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, IEEE, pp.741–746.

Pintard, L., Leeman, M., Ymlahi-Ouazzani, A., Fabre, J-C., Kanoun, K. and Roy, M. (2015) 'Using fault injection to verify an autosar application according to the ISO 26262', in *SAE 2015 World Congress & Exhibition*, SAE International.

Ramanathan, S. and Easwaran, A. (2015) 'MC-fluid: rate assignment strategies', in *Proc. 3rd Workshop on Mixed Criticality Systems (WMC), RTSS*, pp.6–11.

Ramanathan, S., Easwaran, A. and Cho, H. (2018) 'Multi-rate fluid scheduling of mixed-criticality systems on multiprocessors', *Real-Time Systems*, Vol. 54, No. 2, pp.247–277.

Ren, J. and Phan, L.T.X. (2015) 'Mixed-criticality scheduling on multiprocessors using task grouping', in *2015 27th Euromicro Conference on Real-Time Systems*, IEEE, pp.25–34.

Santy, F., George, L., Thierry, P. and Goossens, J. (2012) 'Relaxing mixed-criticality scheduling strictness for task sets scheduled with FP', in *2012 24th Euromicro Conference on Real-Time Systems*, IEEE, pp.155–165.

Schmidt, R. and García-Ortiz, A. (2022) 'Implications of non-uniform deadline scaling to quality of service under single errors', *IEEE Access*, Vol. 10, pp.14586–14599, DOI: 10.1109/ACCESS.2022.3143714.

Su, H. and Zhu, D. (2013) 'An elastic mixed-criticality task model and its scheduling algorithm', in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, pp.147–152.

Su, H., Deng, P., Zhu, D. and Zhu, Q. (2016) 'Fixed-priority dual-rate mixed-criticality systems: schedulability analysis and performance optimization', in *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, IEEE, pp.59–68.

Su, H., Zhu, D. and Moss, D. (2013) 'Scheduling algorithms for elastic mixed-criticality tasks in multicore systems', in *2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications*, IEEE, pp.352–357.

Sundar, V.K. and Easwaran, A. (2019) 'A practical degradation model for mixed-criticality systems', in *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, IEEE, pp.171–180.

Taherin, A., Salehi, M. and Ejlali, A. (2018) 'Reliability-aware energy management in mixed-criticality systems', *IEEE Transactions on Sustainable Computing*, Vol. 3, No. 3, pp.195–208.

Tamas-Selicean, D. and Pop, P. (2011) 'Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures', in *2011 IEEE 32nd Real-Time Systems Symposium*, IEEE, pp.24–33.

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

Trüb, R., Giannopoulou, G., Tretter, A. and Thiele, L. (2017) 'Implementation of partitioned mixed-criticality scheduling on a multi-core platform', *ACM Transactions on Embedded Computing Systems (TECS)*, Vol. 16, No. 5s, pp.1–21.

Völp, M., Hähnel, M. and Lackorzynski, A. (2014) 'Has energy surpassed timeliness? Scheduling energy-constrained mixed-criticality systems', in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, IEEE, pp.275–284.

Völp, M., Lackorzynski, A. and Härtig, H. (2013) 'On the expressiveness of fixed priority scheduling contexts for mixed criticality scheduling', *Proc. WMC, RTSS*, pp.13–18.

Vestal, S. (2007) 'Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance', in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, IEEE, pp.239–243.

Ward, B.C., Herman, J.L., Kenna, C.J. and Anderson, J.H. (2013) 'Outstanding paper award: making shared caches more predictable on multicore platforms', in *25th Euromicro Conference on Real-Time Systems*, IEEE, pp.157–167.

Woolley, B., Mengel, S. and Ertas, A. (2020) 'An evolutionary approach for the hierarchical scheduling of safety-and security-critical multicore architectures', *Computers*, Vol. 9, No. 3, p.71.

Xu, H. and Burns, A. (2015) 'Semi-partitioned model for dual-core mixed criticality system', in *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, ACM, pp.257–266.

Xu, H. and Burns, A. (2019) 'A semi-partitioned model for mixed criticality systems', *Journal of Systems and Software*, Vol. 150, pp.51–63.

Yun, H., Yao, G., Pellizzoni, R., Caccamo, M. and Sha, L. (2012) 'Memory access control in multiprocessor for real-time systems with mixed criticality', in *2012 24th Euromicro Conference on Real-Time Systems*, IEEE, pp.299–308.

Zhao, Q., Gu, Z. and Zeng, H. (2014) 'HLC-PCP: a resource synchronization protocol for certifiable mixed criticality scheduling', *IEEE Embedded Systems Letters*, Vol. 6, No. 1, pp.8–11.

Zhao, Q., Gu, Z. and Zeng, H. (2015) 'Resource synchronization and preemption thresholds within mixed-criticality scheduling', *ACM Transactions on Embedded Computing Systems (TECS)*, Vol. 14, No. 4, pp.1–25.

Zhao, Q., Gu, Z., Zeng, H. and Zheng, N. (2018) 'Schedulability analysis and stack size minimization with preemption thresholds and mixed-criticality scheduling', *Journal of Systems Architecture*, Vol. 83, No. 83, pp.57–74.