Mixed-Criticality Scheduling in Real-Time Multiprocessor Systems

Sergey Osmolovskiy, Ivan Fedorov, Vladimir Vinogradov, Ekaterina Ivanova, Daniil Shakurov
Saint-Petersburg State University of Aerospace Instrumentation
Saint-Petersburg, Russia
{sergey.osmolovskiy, ivan.fedorov, ekaterina.ivanova}@guap.ru, {v.v.vinogradov, daniil.shakurov95}@gmail.com

***Abstract*—The relevance and popularity of mixed-criticality real-time systems precipitously increase in many industrial domains. Today, mixed-criticality systems are increasingly being implemented on multicore platforms. So, one of the most actual and important problem in the field of mixed-criticality system is a scheduling tasks with different criticality levels on multiprocessors. In this paper, we will describe the main features of mixed-criticality systems. These include basic methods of its organization and main issues of mixed-criticality scheduling for multicore platforms. Moreover, we will report kinds of multiprocessors, types of multiprocessor scheduling, standards, concepts and research projects, related to mixed-criticality systems. At last, we will detail and compare several mixed-criticality scheduling approaches for multiprocessors.**

I. INTRODUCTION

An increasingly important trend in the design of real-time and embedded systems is the integration of components with different levels of criticality onto a common hardware platform. At once, these platforms are migrating from single cores to multicores architectures. Criticality is a designation of the level of assurance against failure needed for a system component. A mixed-criticality system is one that has two or more distinct levels (for example safety critical, mission critical and low-critical). Typical names for the levels are ASILs (Automotive Safety and Integrity Levels), DALs (Design Assurance Levels) and SILs (Safety Integrity Levels). These levels express the required protection against failure when designing a safety-critical system and hence, influence all steps of the specification, design, development, testing, and certification processes [1].

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

Many standards for different domains, such as electronic systems (IEC 61508), airborne civil avionics (DO-178B) [3], nuclear power plants (IEC 880), medical systems (IEC 601-4), European railway (EN 50128]), European space (ECSS), etc [4]. rely on the assignment of integrity or criticality levels to the different components of the system. These levels represent the likelihood of a safety related system for satisfactorily performing the required safety functions under all the stated conditions within a stated period of time. The integrity level determines the development methods and validation and verification techniques to be used.

In modern mixed-criticality systems, components with different criticality levels coexist on the same execution platform. In this scenario, certification authorities would require the certification of the whole system, even the less critical parts, which would likely result in the cost of certification rising to prohibitive levels. To resolve this problem can use virtualization in mixed-criticality systems. Under this approach, a hypervisor implements partitions (virtual machines) that are isolated from each other in the temporal and spatial (i.e. storage) domains. Applications with different criticality levels can be located in different partitions, so that there are no undesirable interferences. In this way, only the hypervisor and the critical partitions have to be certified to the highest levels.

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

Nowadays, mixed-criticality systems are increasingly being implemented on multiprocessor platforms. As more functionalities with different degrees of criticality are implemented on a common multiprocessor platform, mixed-criticality systems are becoming more complex, less uniform and predictable, and show greater variation in their performance [5].

In this way, one of the most actual and important problem in the field of mixed-criticality system is a scheduling tasks with different criticality levels on multicore platform. In this paper we are produced a comparison of the most popular approaches and algorithms for mixed-criticality scheduling in multicore systems.

The aim of this paper is to provide a general view of mixed-criticality systems, their technical challenges, some research results and mixed-criticality scheduling. Section II

introduces to mixed-criticality systems, the main research challenges related to them and an overview of some relevant research projects. Partitioned systems described in Section III. Section IV covers multiprocessor and mixed-criticality scheduling issues. Description and compare of the main scheduling approaches and algorithms of mixed-criticality multicore systems is the content of Section V. Conclusions are presented in Section VI.

II. MIXED-CRITICALITY SYSTEMS

A mixed-criticality system is a system in which tasks of different criticality levels (critical and non-critical) run on the same computing platform (processor).

System designers have an interest in scheduling tasks of different criticality on the same processor to reduce costs, and also to be able to better utilize the processor. But many embedded systems, especially safety-critical systems, require certification by certification authorities which usually set different standards from the system designers.

When tasks of different criticality are put on the same processor, the authorities will demand that all tasks will be certified to the level of the task with the highest criticality. This introduces pessimism which diminishes the utility of this approach for the system designer. Much of the research work on mixed-criticality systems has been done to try to reconcile these two conflicting interests.

A key aspect of mixed-criticality systems is that system parameters, such as tasks' worst-case execution times (WCETs), become dependent on the criticality level of the tasks. So the same code will have a higher WCET if it is defined to be safety-critical (as a higher level of assurance is required) than it would if it is just considered to be mission critical or indeed non-critical. This property of mixed-criticality systems significantly modifies/undermines many of the standard scheduling results [6].

The following features will be present in a number of next generation embedded systems including mixed-criticality systems:

- Coexistence of applications with different safety and security levels: the requirement for integrating a number of applications implies that they will be of different nature. It is no longer advisable to isolate applications with more demanding requirements on a processor, due to the different type of associated costs. They must coexist in the same computer with other applications, while behaving as expected.
- Requirements on size, weight and power (SWaP): Many embedded systems have this kind of requirements. A large number of them will be mobile, while others will be embedded on other equipments. Improvements on SWaP features will allow the development of devices that are easier to carry or with larger autonomy.
- Functional complexity: processor performance makes it possible to encapsulate a large number of functions within one system, in order to produce competitive devices. This complexity poses a number of challenges to consider
- High computer performance: a large number of the embedded applications will be multicore, which provides high computing power to the users. As it has been a common trend, systems developers will try to take advantage of this feature for providing the most advance applications.
- Non-functional requirements: these type of requirements are not directly associated with a specific function or component of the system. They usually apply to the system as a whole. Non-functional requirements are usually defined as constraints on the system functionality. Time, reliability, availability, safety, or security, are examples of non-functional requirements. Time requirements are of specific interest in the development of control systems. The outcomes of the application have to be produced within a given time interval. Otherwise it is considered to be faulty. System developers have to ensure that time requirements are always met for safety critical applications.

In order to fulfil these requirements a strong isolation of applications (critical and non-critical) is needed. An application is isolated from others if its execution is not influenced by the behavior of the other applications. Different kinds of isolation can be considered:

- Spatial isolation: applications must execute in independent physical memory address spaces. The system must control that applications cannot access any memory areas that have not been specifically allocated to them
- Temporal isolation: the real-time behavior of an application must be correct independently of the execution of other applications. The allocation of the system resources to an application is not influenced by others, and can be analyzed in a independent way.
- Fault isolation: a fault in an application must not propagate to other applications. Any fault must be handled either by the failing application itself or by the system.

A. *Open issues in research*

There are several open issues in research with respect to the development of mixed-criticality embedded systems. Some of them can be summarized as:

- Scheduling techniques for mixed-criticality systems: Scheduling policies and scheduling techniques. The scheduling problem is one of the most active research area, resulting in proposals of different approaches to deal with partitioned systems.
- Support for multicore platforms: Shared hardware resources in multicore systems have an impact on temporal isolation. The use of shared resources (such as L2/L3 cache, memory, bus, IO, etc.) by partitions

running on different cores in parallel introduces an interference in the overall execution.

- System modelling: The development process should start with a description or model of the system under development. There is a need to define notations that allow providing all the functional components. In addition, it is required to find ways to describe other types of information relevant for system partitioning and deployment.
- Methodology and development tools: The development of mixed-criticality includes additional activities, such as partitioning or system integration, that are not common in previous systems.

In this paper and our future researches, we are focused on a mixed-criticality scheduling in multiprocessor systems.

B. Research Projects

The great industrial interest of mixed-criticality systems has motivated the definition of a research roadmap and challenges. Scheduling of mixed-criticality applications is an emerging research field, which has been attracting increasing attention in recent years. Some of the most relevant projects with research activities of direct interest in this field are:

- IMA-SP (Integrated Modular Avionics for Space). The European Space Agency (ESA) launched in 2011 this project to study the applicability of IMA architectures to space applications. The aim of the project was to define the requirements for the use of temporal and spatial partitioning systems (TSP) in the space domain, using the specific hardware available [7].
- ACROSS (ARTEMIS CROSS-Domain Architecture) is a research project that aims to develop and implement an ARTEMIS cross-domain reference architecture exploiting (a cross-domain architecture for embedded Multi-Processor Systems-on-a-Chip (MPSoC) and implementing a first version in an FPGA.), for example, PikeOS, partitioning and time-triggered bus access protocols, for embedded systems based on the architecture blueprint developed in the EU FP7 project GENESYS [8].
- ARAMiS: (ARAMiS: Automotive, Railway and Avionics Multicore Systems) Its objective is to develop support for the appropriate deployment of multicore systems and virtualization in the domains of automotive, avionics and railway, especially for safety related systems. The target systems will be multicore and relying on virtualization for providing safety critical applications in mobility domains [9].
- MCC (Mixed Criticality Embedded Systems on Many-Core Platforms) is a UK funded (EPSRC) project looking at the design, verification and implementation of mixed-criticality systems. It has a specific focus on developing NoC criticality-aware protocols [10].
- MultiPARTES: (Multicores Partitioning for Trusted Embedded Systems) This project is aimed at developing tools and solutions for building trusted embedded systems with mixed-criticality components on multicore platforms. The approach is based on developing an innovative open-source multicore platform virtualization layer based on the XtratuM hypervisor [11].
- vIrtical: (SW/HW extensions for heterogeneous multicore platforms) The goal of this project is to increase functionality, reliability and security of embedded devices at sustainable cost, and power consumption. The project relies on virtualization as the basis for this aim. In particular, it tries to provide to the virtualization concept in embedded devices, the same maturity level as in the general-purpose computing domain, in terms of flexibility and security.
- PROXIMA is an EU FP7 IP Project. The PROXIMA project provides industry ready software timing analysis using probabilistic analysis for many-core and multi-core critical real-time embedded systems and will enable cost-effective verification of software timing analysis including worst case execution time. PROXIMA defines new hardware and software architectural paradigms based on the concept of randomization [12].
- DREAMS (Distributed REal-time Architecture for Mixed Criticality Systems) aims to produce a European reference architecture for mixed-criticality systems. The objective of DREAMS is to develop a cross-domain architecture and design tools for networked complex systems where application subsystems of different criticality, executing on networked multi-core chips, are supported [13].
- parMERASA: (Multicore Execution of Parallelised Hard Real-Time Applications Supporting Analyzability) The goal of this project is to make it easier to use multi-core processors in the development of real-time systems. This project will provide technical innovations for dealing with aspects such as parallelization techniques for safety-critical applications, timing analysable parallel design patterns, operating system virtualization, and efficient synchronisation mechanisms, or timing analysable multi-core architecture with up to 64 cores [14].
- The RECOMP (Reduced certification cost for trusted multi-core platforms) research project aims to establish methods, tools and platforms for enabling cost-efficient certification and re-certification of safety-critical systems and mixed-criticality systems, i.e. systems containing safety-critical and nonsafety-critical components [15].
- CERTAINTY: (CErtification of Real Time Applications desIgNed for mixed criticaliTY) It addresses the certification process for mixed-critical embedded systems featuring functions dependent on information of varying confidence levels. The main challenge is to increase the complexity of the systems,

where time and safety critical solutions becomes even more complex on multi-core platforms as time disturbances, uncertainties, and unreliability are emerging side effects that need to be efficiently handled. Application domains include avionic, automotive and automation, where real-time and safety-critical requirements are of primary importance [16].

- EMC^2 (Embedded multi-core systems for mixed-criticality applications in dynamic and changeable real-time environments) is an ARTEMIS project with 100 (approx) partners is looking at open and adaptive systems [17].
- CONTREX (Design of embedded mixed-criticality CONTRol systems under consideration of EXtra-functional properties) focusses on platforms and addresses properties such as real-time, power, temperature and reliability. It aims to develop meta-models for design and analysis [18].

C. Mixed-criticality scheduling

The scheduling algorithms in real-time systems must predictably assure a priori that all tasks are completed by their deadlines, assuming that the tasks follow the specifications in the workload model. Current hard real-time scheduling and analysis techniques are unable to efficiently utilize the computational bandwidth provided by multicore platforms. This is due to the large gap between WCET predictions used in schedulability analysis and actual execution times seen in practice. Naturally, because tasks of higher criticality may cause more severe damage when late, their analysis is taken more seriously and results in more pessimistic WCET estimates. In many papers this gap considers as “slack” that can be accounted for during schedulability analysis and reclaimed for less critical work. This technique was used to develop an architecture for scheduling mixed-criticality real-time workloads on multiprocessor platforms. This architecture provides temporal isolation among tasks of different criticalities while allowing slack to be redistributed across criticality levels [19].

The mixed-criticality scheduling problem meets two separate goals: certification of the high criticality tasks under more pessimistic assumptions and feasibility of all the tasks (including the low criticality ones) under the designer’s, less pessimistic, assumptions [5]. Therefore, while the traditional real-time system scheduling favors only urgent jobs, mixed-criticality systems must also prioritize high-criticality jobs so that they are prepared for potentially long execution [20].

Many papers described two criticality levels; high (*HI*) and low (*LO*) with $HI > LO$. These are referred to as dual-criticality systems. Where modes are used, the system is either in a LO-criticality mode or a HI-criticality mode [2].

D. Real-time task model

In most cases, real-time system is modelled as a number of tasks which need to be scheduled on one or more processors according to their timing properties and constraints. A task τi has the following timing properties:

- Relative deadline *Di*: The time by which this task needs to be done.
- Period *Ti*: The (minimum) inter-arrival time between releases.
- WCET *Ci*: The uninterrupted/undisturbed execution time of this task in the worst case.

These can also be written as a tuple (Ci, Ti, Di). A job of a task is an instance of the task. A task can spawn an infinite number of jobs.

The periodicity of a task depends on the relation of the period and the release time. A strictly periodic task is released exactly every T time units, i.e. T specifies the exact inter-arrival time. A sporadic task has an inter-arrival time of at least T time units, but the release may happen later. Finally, aperiodic tasks show no periodicity, they are released randomly. All task systems in this thesis consist of sporadic tasks.

If a task has a deadline which is lower or equal to its period ($D \leq T$), we call it a constrained deadline task system. If the deadline is equal to the period ($D = T$), we call it an implicit deadline task system. In any other case, we call it an arbitrary deadline task system [6].

III. PARTITIONED SYSTEMS

Partitioned software architectures were defined to create trusted systems. They have evolved to fulfil security and avionics requirements, where predictability is extremely important. A partition is an execution environment integrated by an operating system and a set of applications. Partitions are executed on top of a hardware platform, possibly virtual, in an independent way.

In this way, the coexistence of mixed-criticality tasks with different criticality level relies on hypervisors that provide virtual machines or partitions, where tasks can run with space and time isolation. Therefore it is possible to ensure that tasks on different partitions can run independently.

The MILS (Multiple Independent Levels of Security and Safety) initiative is a joint research effort between academia, industry, and government to develop and implement a high-assurance, real-time architecture for embedded systems. The technical foundation adopted for the so-called MILS architecture is a separation kernel. In addition, the ARINC-653 (AEEC, 1996) standard uses these principles to define a baseline operating environment for application software used within Integrated Modular Avionics (IMA), based on a partitioned architecture [21].

Integrated Modular Avionics (IMA) was the solution that allowed the aeronautic industry to integrate new functionalities, while maintaining the level of complexity and efficiency. Its main goal was to define an architecture that captures and handles faults at the different levels and permits the parallel application development. One of the main aspects to cover fault management is the temporal and spatial isolation of partitions.

The major benefit of using a partitioning approach for mixed-criticality systems is to reduce certification costs in complex systems. In order to achieve this goal, it must be possible to analyze each application in an independent way. This requirement has some implications on the overall system:

- The hypervisor has to be certified at the same criticality level as the most critical application.
- System resources (CPU time, memory areas, I/O ports etc.) have to be allocated to partitions in a predefined and static way. Static allocation is necessary to enable independent analysis and certification of partitions. This principle applies both to the applications running in a partition and to the underlying operating system that controls their execution within the partition.
- Non-critical applications running in separate partitions do not have to be certified, as long as it can be guaranteed that they do not affect to the execution of critical partitions.
- Re-certification of a partition should not affect the certification status of other certified partitions.
- Incremental certification is a goal in order to achieve independent certification. Static allocation of resources is a major requirement for achieving the previous described aims.

IV. Multicore scheduling

The multiprocessor scheduling problem consists therefore in finding a feasible schedule for n tasks running on m processors. In the following we assume that $n \geq m$. Multiprocessor real-time scheduling is intrinsically a much more difficult problem than monoprocessor scheduling. The main reason is that few of the results obtained for monoprocessors can be directly applied to the multiprocessor case [22].

From the perspective of scheduling, multiprocessor systems can be classified into three categories:

- Identical: processors are identical, i.e. all processors have equal speed and capabilities.
- Uniform heterogeneous: each processor in a uniform (or related) multiprocessor system is characterized by its own computing capacity (speed), i.e. all processors have equal capabilities, but different speeds.
- Unrelated heterogeneous: processors are different.

Multiprocessor scheduling has to solve two problems: the allocation problem, that requires deciding on which processor a task should execute, and the local scheduling problem, that is, when a task should execute. Regarding allocation, there are scheduling algorithms that permit migration (at task or job level) and algorithms that do not permit it.

The most multiprocessor scheduling algorithms can be classified as either partitioned or global. Scheduling algorithms where no migration is permitted are referred to as partitioned, whereas those where migration is permitted (either at task or at job level) are referred to as global. In addition there is a hybrid approach to the scheduling task is to generalization of global and partitioned scheduling. For example cluster and semi-partitioned scheduling.

In this paper we will look at global and partitioned scheduling.

A. Partitioned scheduling

Under partitioned scheduling, each task is assigned to a single processor. Partitioned scheduling is illustrated in Fig.1, where τ is a task. This has the following advantages:

- Task overruns have only consequences in the same processor.
- There is no penalty in terms of migration cost.
- The implementation uses a separate run queue per processor, rather than a single global queue in the global approach. This reduces overheads due to queue management.

On the contrary, its main disadvantages are:

- Finding an optimal allocation of tasks to processors is a bin-packing problem, that is NP-hard in the strong sense.
- There are task sets that are only schedulable if migration is allowed.
- Partitioned scheduling algorithms are not work conserving, as a processor may become idle, but cannot be used by ready tasks allocated to a different processor.

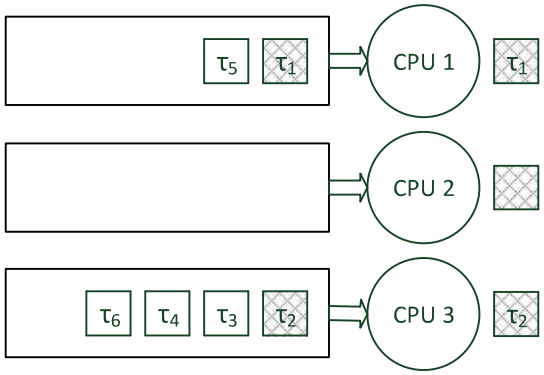

Fig. 1. Partitioned scheduling

Still, partitioning is widely used by system designers. Typical memory allocators such as First-Fit (FF), Worst-Fit (WF), and Best-Fit (BF) have been used to solve the problem of finding good sub-optimal static allocation of tasks to processors. These heuristics use the task period parameter as the key for allocation. Others, such as FFDU (First Fit Decreasing Utilization) [23], use the task utilization as a key for choosing the next task to allocate. These allocators combined with classical scheduling algorithms (FPPS or EDF) gives rise to the most popular partitioned scheduling algorithms, such as Rate Monotonic-First-Fit (RMFF), Earliest Deadline First Best-Fit (EDFBF), Rate Monotonic-First-Fit-

Decreasing-Utilization (RMFFDU), etc. A comparison of these allocation schemes can be found in [24].

B. Global scheduling

Global scheduling cannot be used to reduce the multiprocessor scheduling problem to many monoprocessor scheduling problems, contrary to partitioned scheduling. The fact that tasks are allowed to migrate in the global approach gives rise to many unexpected effects and disadvantages, which complicate the design of scheduling and allocation algorithms for the global scheme. Global scheduling is illustrated in Fig.2, where τ is a task. The most significant problems are:

- Migration of tasks to processors introduce a high overhead in the system.
- Migration increases the information flow between processors. This kind of communication may require the use of shared memory or communication channels.
- Predictability is much lower than that associated to the partitioned scheme.
- Some scheduling anomalies may occur, for example the Dhall effect: tasks sets with very small utilization may be unschedulable [25].

On the contrary, the main advantages of this approach are:

- There are typically fewer context switches/preemptions. This is because the scheduler will only preempt a task when there are no idle processors.
- An advantage of the global scheme is its generality. Since tasks can migrate from one processor to another, the processor system "could be" better utilized.
- Global scheduling is more appropriate for open systems, as there is no need to run load balancing/task allocation algorithms when the set of tasks changes.

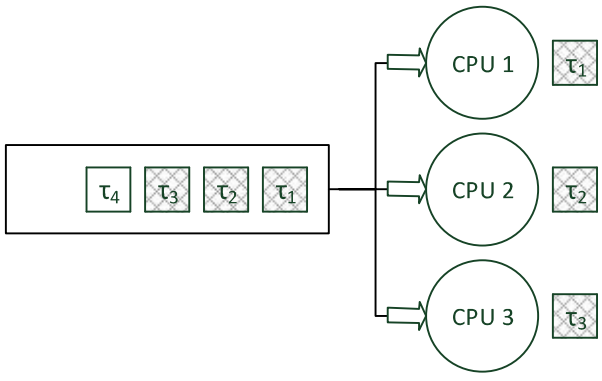

Fig. 2. Global scheduling

To classify global scheduling algorithms we use the concepts of fixed job priorities or fixed task priorities. In the former approach, the priority of a task can only change at job boundaries, while in the latter all jobs generated by the same task have identical priorities. Some well-known global scheduling methods are [26]:

- In fixed task priority (FTP) scheduling, each sporadic task is assigned a unique priority, and each job inherits the priority of the task that generated it. The rate-monotonic scheduling algorithm, which assigns priorities to tasks according to their period parameters — tasks with smaller period are assigned greater priority (ties broken arbitrarily)—is an example of an FTP scheduling algorithm.
- In fixed job priority (FJP) scheduling, different jobs of the same task may be assigned different priorities. However, the priority of a job, once assigned, may not change. The earliest deadline first (EDF) scheduling algorithm in which the priority of a job depends upon its deadline parameter—jobs with earlier deadlines are assigned greater priority (ties broken arbitrarily)—is an example of a FJP scheduling algorithm.
- In dynamic priority (DP) algorithms, there are no restrictions placed upon the manner in which priorities are assigned to jobs—the priority of a job may change arbitrarily often between its release time and its completion. For example:
  - The PFair scheduling algorithms;
  - Earliest Deadline Zero Laxity (EDZL);
  - Least Laxity algorithm.

V. MIXED-CRITICALITY MULTICORE SCHEDULING

Scheduling of mixed-criticality applications is an emerging research field, which has been attracting increasing attention in recent years. In this section we are reviewing different approaches to mixed-criticality scheduling for multicore systems. We consider three well known mixed-criticality scheduling techniques: FTTS and multicore EDF-VD algorithms, and a mixed-criticality scheduling framework for multicore platforms, called MC^2.

A. Flexible Time Triggered Scheduler

A global (flexible) time-triggered scheduling approach with barrier synchronization (FTTS) considers periodic mixed-criticality task sets executed on resource-sharing multicores. FTTS enabled only tasks of the same criticality level to be executed concurrently in order to guarantee their timing properties at a particular level of assurance, a necessary property for the certification of mixed-criticality systems.

Like a standard time triggered scheduler, the FTTS has a cycle with a duration of the least common multiple of all task periods. This cycle is divided into frames, which can be of different lengths, but are also fixed during runtime.

FTTS tries to bridge the space between strict partitioning mechanisms for timing isolation (industrial practice for safety-critical applications) and efficient mixed-criticality scheduling algorithms. Moreover, in multi-core environments there can be access contentions for shared resources (e.g. synchronous access to a shared memory), which means that an access from one core can block tasks on other cores from continuing their execution. The FTTS takes into account concurrent access to

shared memory in its analysis and assures that no low critical task can preempt a higher critical one [27].

As the experimental results show, FTTS can schedule task sets without over-provisioning resources (increased schedulability compared to static partitioning approaches). At the same time it does not affect its efficacy when compared to more dynamic mixed-criticality scheduling strategies. Applicability has been validated with an industrial avionics application. This confirms that FTTS is a potential solution to the problem of mixed-criticality scheduling on multicores, where resource (e. g., memory) sharing among several cores cannot be eliminated. FTTS is currently being implemented in an industrial setup for evaluating its runtime overhead [2].

B. Multicore EDF with Virtual Deadlines

Earliest Deadline First with Virtual Deadlines (EDF-VD) is a singlecore mixed-criticality scheduling algorithm. Nowadays, EDF-VD is extended to both global and partitioned multicore scheduling.

Global mixed-criticality scheduling approach extends the EDF-VD singlecore mixed-criticality scheduling algorithm to multicore, by applying the multicore global scheduling algorithm fpEDF for scheduling systems of non mixed-criticality implicit-deadline sporadic tasks to mixed-criticality systems [28], [29].

The partitioned EDF-VD scheduler (pEDF-VD) is an extension of the singlecore EDF-VD scheduler to multicore systems. It consists of one EDF-VD scheduler per processor core and a partitioning algorithm to partition the tasks to processing cores. The partitioning is done offline and assigns each task to one of the singlecore schedulers on the different processor cores. At runtime each of the singlecore scheduler schedules its own task set independent of the other cores and without any information exchange with them. The details of the used singlecore EDF-VD scheduler and the partitioning algorithm are discussed in the next two sections.

1) Singlecore EDF-VD

Singlecore EDF-VD scheduler is a central part of the partitioned EDF-VD. This singlecore scheduler is already implemented in the mixed-criticality extension for the HSF. It supports two criticality levels. This scheduler starts by scheduling the tasks in low criticality mode according to the EDF scheduling algorithm, namely depending solely on their deadlines and independent of the criticality level of the different tasks. The only change that needs to be done for scheduling in the low criticality mode is to adjust the deadlines of the high critical tasks to guarantee their original deadline when they need more time to execute. This is done by multiplying their original relative deadline with a factor $x \leq 1$ to obtain their virtual deadline. The factor x depends only on the task set and is calculated offline before the task set is handled by the scheduler.

The scheduler switches to high criticality mode as soon as one of the high critical tasks has overrun its low critical WCET. At this point the scheduler cancels all low critical tasks that have not finished their execution. The EDF-VD scheduler then continues scheduling only the high critical tasks according to EDF, this time according to their original, unmodified deadlines. The high criticality mode is used until no pending high critical tasks need to be scheduled. Then the scheduler switches back to low criticality mode and continues scheduling the low criticality tasks according to their original deadlines and the high criticality tasks according to their virtual deadlines [12].

2) Task Set Partitioning

Even if each core will have a fixed task set and its own EDF-VD scheduler during runtime, the task set for the pEDF-VD scheduler is given globally. This task set first needs to be partitioned before starting the independent schedulers.

In a first step all the high critical tasks are distributed across the cores such that the high critical utilization U_{HI}^{HI} of the task sets on each core does not exceed ¾. This is done using a first fit bin packing algorithm and starts with highest utilization task first. If one of the high critical task can not be mapped to any core, the partitioning algorithm stops and returns failure. After this first step the same is done using the low critical utilization of each core (including low critical utilizations of already partitioned high critical tasks). Each low critical task is mapped onto the first core that matches the criteria of having a low critical utilization of lower than or equal ¾. Here the partitioning algorithm stops as well and returns failure when a task could not be mapped to any core.

If both steps succeed, the partitioning algorithm returns success.

C. MC^2 framework

One of the main challenges in mixed-criticality system is to devise mixed-criticality scheduling (and ultimately synchronization) approaches for multicore platforms that are amenable to certification. As a step towards addressing this challenge, the researchers at University of North Carolina at Chapel Hill, in collaboration with colleagues at Northrop Grumman Corp. (NGC) proposed a mixed-criticality scheduling approach for multicore platforms that uses a two-level hierarchical scheduling framework in which containers provide isolation for tasks of different criticality levels. Partitioned EDF is used as the intra-container scheduler. This approach named MC^2. This framework provided corresponding schedulability analysis results [19]. MC^2 also proposed the use of slack re-allocation techniques to redistribute unused processing capacity at higher criticality levels to lower criticality levels.

MC^2 framework supports five criticality levels, denoted A (highest) through E (lowest). The choice of five levels was motivated by the five criticality levels found in the DO-178B/C standard for avionics, which is used by the U.S. Federal Aviation Administration (FAA) for the certification of commercial airplanes. As explained in greater detail later, level-A and -B tasks in MC^2 are subject to hard deadlines and are scheduled via partitioning, while level-C and -D tasks are subject to bounded deadline tardiness and are scheduled globally. Level-A tasks are statically prioritized above all other tasks in the system. They are scheduled according to a precomputed dispatching table, following the cyclic executive

(CE) scheduling model [19]. Level-E tasks are scheduled as best-effort tasks because DO-178B merely specifies that a failure at this level must not affect the operation of the aircraft. The schedulability analysis provided for MC^2 can be applied to validate the schedulability. of the level-l system, where l ranges over A–D (as required in a mixed-criticality setting; note that level E is best effort, so it requires no schedulability analysis) [30]. In MC^2 different intracontainer schedulers are used for tasks of different criticalities. MC^2 framework is illustrated in Fig.3.

D. Comparison

We compare three mixed-criticality scheduling techniques discussed above by several capabilities and results of implementation.

EDF-VD is more flexible with scheduling task jobs as they arrive and can preempt them any time. FTTS can schedule task sets without over-provisioning resources (increased schedulability compared to static partitioning approaches) [1].

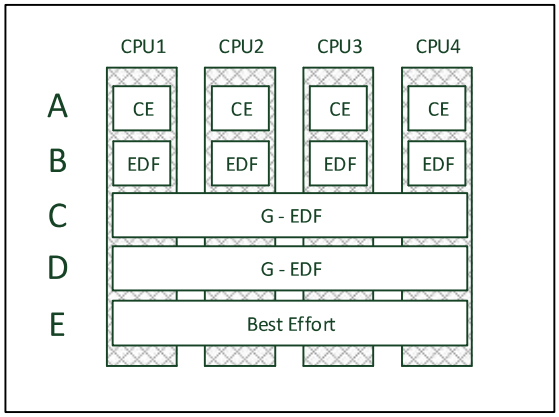

Fig. 3. MC^2 framework

When comparing FTTS to pEDF-VD, we can state that the pEDF-VD scheduler has a much lower overhead than the FTTS. However, the FTTS also has its advantages, like isolation among tasks of different criticality levels. The overhead of the FTTS highly depends on the number of subframes that need to be executed. The number of subframes depends on the dimensioning of the FTTS cycle and frames, which in turn depends on the task periods. pEDF-VD scheduler has low overhead for longer task periods. On the other hand the analysis of the scheduler is complicated, because of its partitioned manner [27].

Looking at the task model, FTTS considers periodic mixed-criticality task sets executed on resource-sharing multicores. While the EDF-VD considers mixed-criticality arbitrary-deadline sporadic task sets. MC^2 supports mixed-criticality implicit-deadline sporadic tasks on multicore platforms with shared resources.

Looking at the shared resources, FTTS supports concurrent access to shared memory and assures that no low critical task can preempt a higher critical one. In MC^2 are developed by Ward et al. two cache-management techniques, called cache locking and cache scheduling and presented experimental results on a multicore Tegra3 ARM machine. The usage of such techniques can reduce WCETs in higher-criticality tasks [31].

Summary of comparing results of approaches mixed-criticality scheduling by shared resource, limit on the number of criticality levels, type of multicore scheduling is shown in Table I.

Looking at the multiprocessor, FTTS offered for identical cores, but it can be easily generalized to heterogeneous platforms. Each core has access to a private local memory and also to a shared (global) memory. Data and instructions are fetched from the shared memory to the local during the access phases of a task, and after each computation phase, the modified data are written back to the shared memory during subsequent access phases [1].

pEDF-VD scheduling offers only identical multiprocessors. The issue of expansion of the algorithm to heterogeneous multiprocessors remains open.

MC^2 supports identical, uniform heterogeneous and unrelated heterogeneous multiprocessors.

Summary of comparing results of approaches mixed-criticality scheduling by kinds of multiprocessors is shown in Table II.

TABLE I. COMPARISON OF SCHEDULING APPROACHES BY SHARED RESOURCES, CRITICALITY LEVELS AND MULTICORE SCHEDULING

Scheduling approach	Shared resources	Criticality levels	Multicore scheduling
FTTS	Concurrent access to shared memory	Two and more	Global
pEDF-VD	No	Two and more	Partitioned
MC^2	Two cache-management techniques: cache locking and cache scheduling	Five (in first version), no limits	Global and partitioned

TABLE II. COMPARISON OF SCHEDULING APPROACHES BY KINDS OF MULTIPROCESSORS

Scheduling approach	Kinds of multiprocessors		
	Identical	Uniform heterogeneous	Unrelated heterogeneous
FTTS	Yes	Underway	Underway
pEDF-VD	Yes	No	No
MC^2	Yes	Yes	Yes

VI. Conclusion

Mixed-criticality systems are the result of the evolution of embedded systems, which is characterized by more complex functionality, more powerful processors, requirements on size, weight and power, and non-functional requirements.

In this paper, we described the main properties and capabilities of mixed-criticality systems, methods of its organization and features of mixed-criticality scheduling algorithms for multicore systems. Also, we described kinds of multiprocessors (identical, uniform heterogeneous and unrelated heterogeneous), standards and concepts (ARINC-653, DO178-B, MILS, IMA etc.), research projects (IMA-SP, EMC^2 in ARTEMIS, ACROSS) related to mixed-criticality systems.

Moreover, we considered types of multiprocessor scheduling. Upon multiprocessor systems, current engineering practice to a greater extent using partitioned scheduling (rather than global and hybrid (clustered) scheduling). One of the reasons for the better efficiency of partitioned algorithms in mixed-criticality scheduling is that properties, often pessimistic, that just characterize the behavior of a partitioning algorithm may constitute the actual schedulability tests for global scheduling. Also, the partitioning algorithm is better, from the perspective of speedup bounds (metric for quantifying deviation from optimal behavior of scheduling algorithm), when compared to the global algorithm.

We detailed and compared several mixed-criticality scheduling approaches (MC^2, FTTS, EDF-VD) for multiprocessors. In MC^2, tasks at each criticality level are scheduled by different intra-container schedulers, so, they according to different scheduling policies. This allows the tasks of each criticality level to be scheduled in a way that is appropriate for that level. FTTS approach with barrier synchronization considers periodic mixed-criticality task sets executed on resource-sharing multicores. FTTS enabled only tasks of the same criticality level to be executed at the same time in order to guarantee their timing properties at a particular level of assurance. EDF-VD algorithm is good scheduling algorithm for dual-criticality implicit-deadline sporadic tasks.

This paper has the goal of serving as an initial starting point for researchers interested on a topic with a great potential in the near future. We formulate the expected range of problems and tasks for personal solution in the future. Our further research is required for the boundary of the performance of multiprocessor mixed-criticality scheduling algorithms for various task types (implicit-/arbitrary-deadline) on various processor types (identical/heterogeneous/unrelated) with various migration rules (unrestricted/job level/task level).

References

[1] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele. "*Scheduling of mixed-criticality applications on resource-sharing multicore systems*". In EMSOFT. IEEE, 2013, pp. 1–15.

[2] A. Burns and R. Davis. *Mixed criticality systems - a review*. Technical report, Department of Computer Science, Unviersity of York, 2013.

[3] *RTCA/DO-178C*, Software Considerations in Airborne Systems and Equipment Certification, December 13, 2011.

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

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

[6] A. Lautenbach, *"Partitioned Fixed-Priority Multiprocessor Scheduling for Mixed-Criticality Real-Time Systems"* Master of Science Thesis in Programme Computer Systems and Networks, Sweden, September 2013.

[7] IMA-SP. *IMA-SP Integrated Modular Avionics for Space*, 2011

[8] ACROSS project website, Web: http://www.across-project.eu/

[9] ARAMiS project website, Web: http://www.projekt-aramis.de/

[10] MCC project website, Web: https://www.cs.york.ac.uk/research/research-groups/rts/mcc/

[11] MultiPARTES project website, Web: http://www.multipartes.eu/

[12] PROXIMA project website, Web: http://www.proxima-project.eu/

[13] DREAMS project website, Web: http://www.dreams-project.eu/

[14] parMERASA project website, Web: http://www.parmerasa.eu/

[15] RECOMP project website, Web: http://atcproyectos.ugr.es/recomp/

[16] CERTAINTY project website, Web: http://www.certainty-project.eu/

[17] EMC^2 project website, Web: http://www.artemis-emc2.eu/

[18] CONTREX project website, Web: https://contrex.offis.de /home/

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

[20] M. Volp, A. Lackorzynski, and H. Hartig. "*On the expressiveness of fixed priority scheduling contexts for mixed criticality scheduling"*. In Proc. WMC, RTSS, 2013, pp. 13–18.

[21] ARINC 653-1 (2006). *Avionics Application Software Standard Interface (ARINC-653)*. PART 1 – REQUIRED SERVICES. Airlines Electronic Eng. Committee.

[22] R. Davis, A. Burns *"A survey of hard real-time scheduling for multiprocessor systems"*, ACM Computing Surveys, 43(4), 35:1, 2011, pp. 35-44.

[23] Y. Oh, S.H. Son *"Fixed-priority scheduling of periodic tasks on multiprocessor systems"*, Technical report, Department of Computer Science, University of Virginia, 1995.

[24] O.U.P. Zapata, P. Mejia-Alvarez "Analysis of real-time multiprocessors scheduling algorithms", In the 24nd IEEE Real-Time Systems Symposium (RTSS'03), 2003.

[25] S.K. Dhall, C.L.Liu "On a real-time scheduling problem", Operations Research, 26(1), 1978, pp. 127–140.

[26] S. Baruah, M. Bertogna, G. Buttazzo. *Multiprocessor Scheduling for Real-Time Systems.*Cham: Springer, 2015.

[27] L. Sigrist. *"Implementation and Evaluation of Mixed-Criticality Scheduling Algorithms for Multi-core Systems"*. Semester Thesis. January 2014.

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

[29] H. Li, *"Scheduling mixed-criticality Real-Time Systems"*, Ph.D. dissertation, University of North Carolina at Chapel Hill, 2013.

[30] J. Herman, C. Kenna, M. Mollison, J. Anderson, and D. Johnson. "*RTOS support for multicore mixed-criticality systems"*. In RTAS, 2012, pp. 197–208.

[31] Kim, Namhoon, Jeremy P. Erickson, and James H. Anderson. "*Mixed-Criticality on Multicore (MC^2): A Status Report"*, OSPERT 2014, 2014, pp. 45–50.