Hierarchical Real-Time Scheduling for Multicore Systems

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

***Abstract*—Nowadays, real-time embedded computing systems are widely used in safety-critical environments such as avionics and space systems. Microprocessor technology is rapidly developing, therefore, multicore designs are becoming an attractive solution to fulfill increasing performance demands. Thereby, in the real-time systems community there has been a growing interest in real-time multicore scheduling theories. One of the multicore scheduling approach that provides predictable timing and temporal isolation (two properties desirable in real-time systems) is a hierarchical scheduling. In this paper, we will review many aspects of hierarchical scheduling for multicore systems. These include existing types of multiprocessor systems, as well as the types of multiprocessor scheduling. Further will be presented standards and specifications related to the time and space partitioning whose problems can be decided by hierarchical scheduling. In addition, we will conduct a description and comparison of several multicore approaches of hierarchical scheduling.**

I. Introduction

Microprocessor technology is rapidly developing, therefore, multiprocessor and multicore designs are becoming an attractive solution to fulfill increasing performance demands. In the real-time systems community, there has been a growing interest in real-time multiprocessor scheduling theories. Multicore architectures have received significant interest as thermal and power consumption problems limit further increase of speed in single-cores. In the multicore research community a considerable amount of work has been done on real-time multicore scheduling algorithms [1].

Embedded systems is an essential part of many modern products including complex safety critical real-time systems. Within the industrial domains of avionics and automotive, the safe composition of several embedded features within the same systems can be achieved through the use of hierarchical scheduling. The separation between features is secured by using time partition scheduling at the system level [2]. A trend within embedded systems is to use multicore platforms in order to increase performance and to be able to implement more functionality within one embedded system [3].

The advent of multicore processors will guarantee a steady improvement of the processors performance over time. However, multicore processors also make application development more difficult if the goal is to make use of all the extra processors performance that multicore brings. For example, shared resources [4] and cache memory [5] are two challenges that the real-time community are faced with. However, adapting to multicore is the only option if the objective is to maximize the processors performance [6].

Hierarchical scheduling is a general term for composing applications as well defined components. Software which is structured in such a way is more robust than flat system since defects will only affect a delimited part of the system. Hierarchical real-time scheduling is the core technology for realizing temporal partitioning. Hierarchical scheduling is used to allow coexistence of hard, soft and non-real-time tasks in applications. In the hierarchical scheduling, the global scheduler (partition scheduler) assigns computation resources across partitions according to their period and execution time. In the second level, the local scheduler (task scheduler) runs tasks of a partition during the time window given by the partition scheduler. The isolation that comes with hierarchical scheduling will also make software reuse more simple.

In this paper we propose consumption of hierarchical scheduling for multicore systems.

This powerful mechanism has been adopted by the avionics industry in form of the ARINC-653 [7] software specification. ARINC-653 isolates applications in terms of both the processors and the memory. Hence, hierarchical scheduling can be used in safety-critical systems to make them more safe. Hierarchical scheduling is also used in mixed-criticality systems — systems providing multiple functionalities who differ in how critical they are for the overall welfare of the system, and in the level of assurance of each one's mandatory certification.

This paper is organized as follows: in Section II we describe kinds of multiprocessors and types of multiprocessor scheduling. In Section III we present a features of real-time scheduling, strategy of hierarchical scheduling, standards, which requirements are satisfied by hierarchical scheduling. Section IV have description and compare of existing hierarchical multicore scheduling approaches. Finally, we conclude this paper in Section V.

II. Background

A. Types of real-time systems

Real-time systems are classified as hard or soft real-time systems [8]:

- Hard real-time (HRT) systems have very strict time constraints, in which missing the specified deadline is unacceptable. The system must be designed to guarantee all time constraints. Every resource management system (for example, the scheduler, communications and input–output manager) must work in the correct order to meet the specified time constraints.
- Soft real-time systems (SRT) also have time constraints; however, missing some deadline may not lead to catastrophic failure of the system. Thus, soft real-time systems are similar to hard real-time systems in their infrastructure requirements, but it is not necessary that every time constraint be met. In other words, some time constraints are not strict, but they are nonetheless important. A soft real-time system is not equivalent to non-real-time system, because the goal of the system is still to meet as many deadlines as possible.

Space missions and military applications and are typical instances of hard real-time systems. Some applications with real-time requirements include rocket and satellite control, aircraft control and navigation, industrial automation and control, telecom switching, car navigation, the medical instruments with the critical time constraints, and robotics.

Some applications with soft real-time requirements include web-services such as real-time query, call admittance in voice over internet protocol and cell phone, digital TV transmissions, cable and digital TV set-top-boxes, video conferencing, TV broadcasting, games, and gaming equipment. Multimedia systems in general are examples of soft real-time systems (e.g., dropping frames while displaying video).

B. Real-time scheduling

In real-time systems the temporal predictability which is often in the form of guaranteeing every task's response within strict deadlines is as important as the performance (how fast an individual task can complete) or the throughput (how many tasks can be completed over a long period of time).

In real-time scheduling theory research, the scheduling algorithms that switch tasks and allocate resources in real-time systems are studied. These algorithms are constructed based on real-time task models. These models extract the essential information of the temporal behaviors of the tasks in a real-time system.

The scheduling algorithms must predictably assure a priori that all tasks are completed by their deadlines, assuming that the tasks follow the specifications in the workload model. These guarantees must be analytically proved before the actual execution of the system, there are usually two types of algorithms on scheduling: scheduling policies and schedulability tests.

Scheduling policies, sometimes called schedulers, are the algorithms that control the run-time schedule. The scheduling policies will be executed along with real-time tasks, and make scheduling decisions based on the time and/or the temporal behavior of real-time tasks. Scheduling policies are generally required to be simple and fast because they compete with real-time tasks and occupy computational resources.

Schedulability tests are the algorithms that check before run-time if the deadlines are guaranteed to be met. Schedulability tests can be complicated and time-consuming if they can bring in better run-time performance and computational resource efficiency. It is non-trivial to design schedulability tests, especially for real-time tasks with a large variance of run-time behaviors because the tests must guarantee that no deadline is missed in all possible system runs [9].

A feasible schedule is one which fulfils all timing constraints of a task system, and a task system is schedulable if at least one scheduling algorithm exists which generates a feasible schedule.

C. Real-time task model

In many works, a real-time system is modelled as a number of independent, recurrent tasks which need to be scheduled on one or more processors according to their timing properties [10]. A task τ_i has the following timing properties:

- Relative deadline D_i: The time by which this task needs to be done.
- Period T_i: The (minimum) inter-arrival time between releases.
- Worst-case execution time C_i: The uninterrupted/undisturbed execution time of this task in the worst case.

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

Very important property is the task utilization, i.e. the ratio of

$$Ui = Ci / Ti \quad (1)$$

the worst-case execution time (WCET) and the period (1).

The utilization provides a very simple scheduling check for any processor. Processor utilization can be defined as the sum of the task utilizations of the tasks scheduled on that processor. If the processor utilization is bigger than 1, the tasks cannot all be scheduled successfully on that processor

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. Aperiodic tasks show no periodicity, they are released randomly.

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.

D. Priority-driven real-time scheduling

A priority-driven scheduler does not pre-compute a schedule of tasks/jobs: instead assigns priorities to jobs when released,

places them on a run queue in priority order. When preemption is allowed, a scheduling decision is made whenever a job is released or completed. At each scheduling decision time, the scheduler updates the run queues and executes the job at the head of the queue.

Fixed-priority algorithm assigns the same priority all jobs in each task. By contrast, dynamic-priority algorithm assigns different priorities to the individual jobs in each task.

Best known fixed-priority algorithm is Rate Monotonic (RM) [11]. It assigns priorities to tasks based on their periods by rule "The shorter the period, the higher the priority". The rate of job releases is the inverse of the period, so jobs with shorter period have higher priority. An example of a dynamic-priority algorithm is Earliest deadline first (EDF), which assigns priorities to jobs in the tasks according to their deadline. The rate of job releases is the inverse of the deadline, so jobs with nearer deadline have higher priority.

E. Kinds of multiprocessors

In multiprocessor computing platforms there are several processors available upon which tasks may execute. Scheduling theorists distinguish between at least three different kinds [12] of multiprocessors:

- Identical: Processors are identical, in the sense that they have the equal speed and capabilities.
- Uniform heterogeneous: Processors have equal capabilities, but different speeds.
- Unrelated heterogeneous: Processors have different capabilities and speed. Tasks may not be able to execute on all processors.

Kinds of multiprocessors are illustrated in Fig. 1.

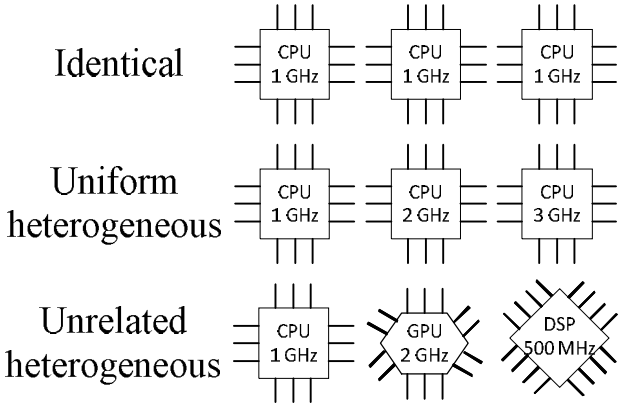

Fig. 1. Kinds of multiprocessors

F. Types of multiprocessor scheduling

Existing real-time scheduling approaches over several processors can fall into three categories: partitioned, global and hybrid scheduling.

1) Partitioned Scheduling

In partitioned scheduling, a task set allocate into multiple non-intersected sets and each of this sets is assigned to a dedicated processor. Each processor have scheduler with a separate run queue, where tasks or jobs cannot migrate. That is allowed during run time. Practically, this type of scheduling allows to apply existing real-time scheduling techniques and schedulability analyses for uniprocessor systems, when allocation of tasks to processor has been achieved. Also partitioned scheduling have advantage in context of multiprocessor systems, introduced by Dhall and Liu [13] where some task set with total utilization close to 1, that are not schedulable by global scheduling even in multiprocessor platform, where more then one processor.

However, partitioned scheduling has the disadvantage to the global scheduling in the sense that partitioned schedulers may require more processors to schedule a task system. This is clear from the fact that 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.

Early research into partitioned multiprocessor scheduling examined the use of common uniprocessor scheduling algorithms such as EDF or RM on each processor, combined with bin-packing heuristics such as First Fit (FF), Next Fit (NF), Best Fit (BF), and Worst Fit (WF), and task orderings such as Decreasing Utilization (DU) for task allocation. Later different variants of EDF and fixed priority algorithms are proposed such as EDF-Utilization Separation (EDF-US), EDF-First Fit Increasing Deadline (EDF-FFID), etc. to improve utilization bound of the partitioned scheduling [14].

2) Global Scheduling

In global scheduling, tasks are scheduled from a single priority queue and may migrate among processors. The main advantage of global scheduling is that it can overcome the algorithmic complexity inherent in partitioned approach. As all the processors use a single shared ready queue, this eliminates the need to solve the task assignment problem, which is the source of complexity under any partitioned scheduling. Another key advantage of global scheduling is that it typically requires fewer preemptions as the scheduler will only preempt a task if there is no idle processor. Global scheduling is more suitable for open systems where new tasks arrive dynamically, as a new task can be added easily to existing schedule without assigning it to a particular partition.

However, unlike partitioned scheduling, results from uniprocessor scheduling does not fit easily for global scheduling of multiprocessors. The problem of global scheduling of real-time tasks in multiprocessors was first considered by Dhall and Liu in the context of the periodic task model. Their result known as Dhall's effect shows that neither RM nor EDF retains its respective optimality property in uniprocessor when the number of processors m exceeds one. Given these early negative results and the lack of widespread availability of shared-memory multiprocessor platforms, interest in the global scheduling was quite limited in the first two decades of research into real-time systems.

3) Hybrid Scheduling

In global scheduling, the overhead of migrating tasks can be very high depending on the architecture of the multiprocessor platform. In fact delays related to cache miss and

communication loads can potentially increase the worst case execution time of a task which is undesirable in real-time domain. On the other hand, fully partitioned algorithms suffer from waste of resource capacity as fragmented resource in each partition remains unused. To overcome this problem, hybrid approaches are proposed which includes semi-partitioned and clustering algorithm.

- Semi-partitioned Scheduling

  In semi-partitioned scheduling algorithm, most of the tasks are executed on only one processor as in original partitioned approach. However, a few tasks (or jobs) are allowed to migrate between two or more processors. The main idea of this technique is to improve the utilization bound of partitioned scheduling by globally scheduling the tasks that cannot be assigned to only one processor due to the limitations of the bin-packing heuristics. The tasks that cannot be completely assigned to one processor will be split up and allocated to different processors. The process of assigning tasks to processors is done offline.

  Semi-partitioning approach is also investigated for fixed priority scheduling and sporadic tasks. Lakshmanan et al. [15] developed a semi-partitioning method based on fixed priority scheduling of sporadic task sets with implicit or constrained deadlines. Their method, called the Partitioned Deadline Monotonic Scheduling with Highest Priority Task Split (PDMS HPTS), splits only a single task on each processor: the task with the highest priority. A split task may be chosen again for splitting if it has the highest priority on another processor. PDMS HPTS takes advantage of the fact that, under fixed-priority preemptive scheduling, the response time of the highest-priority task on a processor is the same as its worst-case execution time, leaving the maximum amount of the original task deadline available for the part of the task split on to another processor.

  Although, semi-partitioned algorithms increases utilization bound by using spare capacities left by partitioning via global scheduling, it has a inherent disadvantage of offline task splitting. It is ongoing state of the art research to efficiently split the tasks with maximum efficiency to reduce overhead related to migration and preemptions.

- Cluster Based Scheduling

  Cluster based scheduling can be seen as a hybrid approach combining benefits of both partitioned and global scheduling. The main idea of the cluster based scheduling is to divide m processors into $\lceil m/c \rceil$ sets of c processors each [16]. Both partitioned and global scheduling can be seen as extreme cases of clustering with c = 1 and c = m respectively.

  Initially the notion of clustering is thought to be similar to partitioning approach where the task set is assigned to dedicated processors during an offline partitioning phase. In case of clustering, this becomes assigning tasks to a particular cluster and give each cluster a set of processors. It simplifies the bin-packing problem of partitioning mentioned earlier as now tasks have to be distributed into clusters. Different heuristics can be applied to assign tasks to cluster to improve the utilization, reduce overhead due to migration and response time. Each cluster handles small number of tasks on small number of dedicated processors and thus removes problem of long task queue experienced by the global scheduling algorithms. Clustering also gives flexibility in the form of creating clusters for different types of tasks such as low or high utilization tasks. Another flexibility offered by clustering is that it is possible to create clusters with different resource capacity such as cluster with large or small number of processors, having same second level cache, etc. Shin et al. [17] further expanded this flexibility by analysing cluster based multiprocessor scheduling for virtual clustering. In contrast to the normal clustering approach known as physical clustering where processors are dedicated for a cluster, virtual clustering assigns processors to cluster dynamically during runtime. Shin et al. proposed the Multiprocessor Periodic Resource (MPR) interface to represent virtual cluster and presented hierarchical scheduling analysis and algorithms for them on identical multiprocessor platform. Easwaran et al. [18] extended this hierarchical scheduling framework with optimal algorithms for allocating tasks to clusters.

Generally, partitioned scheduling is preferable in HRT systems, and global scheduling is preferable in SRT systems. Partitioned approaches have lower run-time overheads, but processing capacity may be wasted due to bin-packing problems. In contrast, global approaches eliminate bin-packing issues and are particularly effective in SRT systems. A drawback of global scheduling is increased the system overheads associated with contention of shared scheduler state [19].

Hybrid approaches represent a “middle ground” between partitioned and global scheduling algorithms. For example, semi-partitioned algorithms require less processors than partitioned algorithms to schedule certain task sets. At the same time, these algorithms do not incur large memory overheads and task migration and preemption overheads like global algorithms [20].

III. Hierarchical Scheduling

A. Overview

Hierarchical scheduling (also referred to as resource reservation) is a hot topic within the research of real-time systems. The main idea is to partition resources (processors, memory, etc.) into well defined slots. This technique is rarely used in the most common real-time applications; however, it is used in the avionics industry to isolate error propagation between system parts, and to facilitate analysis of the system [21].

Hierarchical scheduling has shown to be a useful mechanism in supporting modularity of real-time software by providing

temporal partitioning among applications. This approach has been introduced to support processor multiplexing in combination with different scheduling policies. One of the main advantages of hierarchical scheduling is that it provides means for decomposing a complex system into well-defined parts (subsystems). According to this approach, a system can be hierarchically divided into a number of subsystems that are scheduled by a global scheduler. Each subsystem contains a set of tasks that are scheduled by a local scheduler. Hierarchical scheduling allows for a subsystem to be developed and analyzed in isolation, with its own local scheduler, and then at a later stage, using an arbitrary global scheduler, it allows for the integration of multiple subsystems without violating the temporal properties of the individual subsystems analyzed in isolation. The integration involves a system level schedulability test, verifying that all timing requirements are met. Hence, hierarchical scheduling frameworks naturally support concurrent development of subsystems.

In essence, hierarchical scheduling gives rise to time-predictable composition of coarse-grained subsystems. Hierarchical scheduling has several advantages, besides improving response time of event-triggered tasks. It enables parallel development and testing of subsystems, simplifies integration of subsystems (analysis), supports runtime temporal partitioning and safe execution of tasks, can facilitate fault isolation, structured analysis, legacy system integration [22]. Hierarchical scheduling also facilitates reusability of subsystems, since their computational requirements are characterized by well defined interfaces.

Implemented schemes of hierarchical scheduling is called Hierarchical Scheduling Framework (HSF).

B. Strategies of hierarchical scheduling

In order to achieve hierarchical scheduling, several strategies [23] can be used:

- server-based scheduling;
- compositional scheduling;
- flat scheduling.

1) Server-based scheduling

The improvement in aperiodic servers, as well as a better understanding of the isolation properties of these mechanisms, refocused the application of such servers to what was called “bandwidth servers” or “resource reservation protocols”.

The global scheduler is in charge of scheduling servers according to own scheduling policy and the local scheduler handles the scheduling of tasks. Servers enable real-time tasks to execute in a dynamic environment under a temporal protection mechanism, so that each server will never exceed a predefined bandwidth, independently of its actual workload requests. The scheduling algorithm at any level can be arbitrary. Server based hierarchical scheduling is illustrated in Fig. 2.

The server interface defines the amount of processors that will be reserved for the particular server. It is usually a time window (referred to as budget) that re-occurs at a specific interval of time called period, i.e., similar to the periodic task model.

The servers are re-started periodically. The order of the execution of the servers is affected by the priorities. Servers can have different priorities. Each server could potentially host more than one task each.

In this way, servers act as application containers, providing temporal isolation to applications.

The advantages of this technique include the great amount of research available. However, its main drawback is the difficulty to handle complex task models.

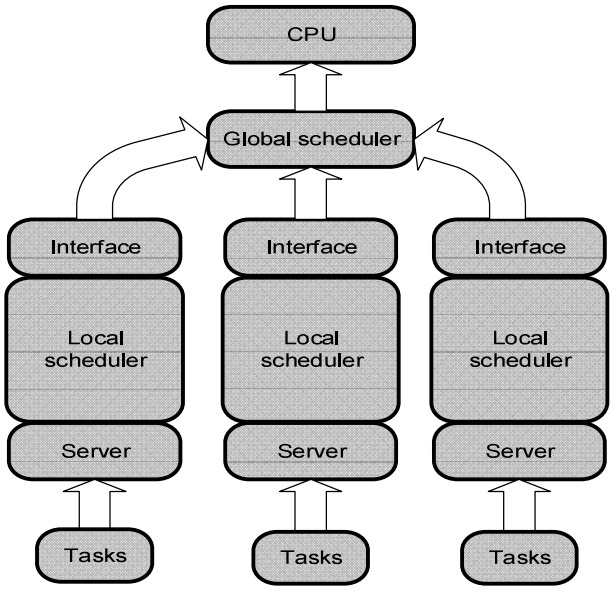

Fig. 2. Server based hierarchical scheduling framework

2) Compositional hierarchical scheduling

The basic idea of this approach is to extend the classical and widely used “divide and conquer” strategy to the temporal requirements. This technique has been widely accepted as a methodology for designing large complex systems through systematic abstraction and composition. The complexity of each component is hidden and abstracted through a clean and well-defined interface.

This approach has the following main advantage:

- Clean isolation of scheduling concerns between partition developers and system integrator. The partition developers do not have to provide details about its internal operation (task attributes), just the temporal abstract interface of the partition.

On the other side, it has some drawbacks:

- The more partitions are in the system, the less processor utilization can be granted.
- Inter-partition resource sharing may be difficult to implement and to take into account in the schedulability analysis.

- The algorithm is not optimal. Some feasible systems cannot be scheduled with this approach.
- The restriction imposed on the periods of the partitions (they must be harmonic) is a limitation needed to produce "efficient" schedules (as well as to be able to use a deadline-monotonic policy at partition level). Otherwise, the resulting schedule may be far from optimal.

3) Flat hierarchical scheduling

In many cases, it is difficult to hide the internal task structure of the partitions because of the need to specify execution flows conducted by input/output operations that involve tasks in different partitions. A flat model approach considers all tasks, independently of the partition where they belong, as a global system. A single global scheduler can then be in charge of managing all the tasks, and a global schedulability analysis can be carried out. The last step is to adapt the solution back to the partitioned system by grouping (trying to put together) the tasks of each partition in order to reduce the number of partition context switches.

This approach has the following advantages:

- Dependencies between tasks of different partitions can be analyzed and solved.
- Mature theory support for this model.
- The resulting schedule (or scheduling policy) can be very efficient. Depending on the task model, it may be possible to find the optimal solution.

It has also the following drawbacks:

- If an optimized solution is desired, a deep knowledge of the timing attributes of all the tasks is needed in order to carry out schedulability analysis.
- There is no clean separation of concerns between partition developers and system integrator, or even among partition developers.
- A change of an attribute of a task may require the whole schedule to be reworked.

B. Standards and specifications

The computing infrastructures supporting onboard aerospace systems, given the criticality of the mission being pursued, have strict dependability and real-time requisites [24]. They also require flexible resource reallocation, and reduced size, weight and power consumption (SWaP). To cater to resource allocation and SWaP requirements, there has been a trend in the aerospace industry towards integrating the multiple hosted functions in the same computing platform. As these functions may have different degrees of criticality and predictability, and originate from multiple providers or development teams, safety issues might arise, which are mitigated by employing time and space partitioning (TSP). In TSP, onboard applications are functionally separated into logical containers — partitions. Partitioning allows containing faults in the domain in which they occur, and enables independent software verification and validation (easing the overall certification process). The issues of this paper are more tightly bound to the aspect of temporal partitioning, which ensures applications executing in one partition will not disrupt the use of any shared resource (most notably the processor) by applications in other partitions. This is essential to ensure the fulfilment of real-time guarantees and enable independent temporal analysis of the applications [25].

TSP concepts have been deployed in the civil aviation world, through the Integrated Modular Avionics (IMA) [26] and ARINC-653 specifications. The interest from space industry partners in applying TSP concepts [27] originated the international consortium, sponsored by the European Space Agency (ESA), within which was developed the AIR (ARINC-653 in Space RTOS) architecture [28].

AIR is designed to fulfil the requirements for robust TSP, and foresees the use of different partition operating systems (POS), either real-time or generic non-real-time ones. The AIR Partition Management Kernel (PMK) ensures robust temporal and spatial partitioning. Temporal partitioning is achieved through a two-level hierarchical scheduling scheme. AIR supports mode-based partition schedules, among which the system can switch throughout its execution for (self-)adaptation to mission changes [29]. The Application Executive (APEX) provides a standard interface between applications and the underlying core software layer.

The ARINC-653 specification (AEEC, 1997) prescribes a two-level hierarchical scheduling framework to guarantee temporal isolation between the applications, each hosted in a logical containment unit — partition. On the first level, a cyclic global level scheduler selects partitions according to a predefined partition scheduling table. When each partition is active according to such schedule, its tasks compete according to a local-level scheduler, which is specified to be preemptive and priority-based [30].

IV. MULTICORE HIERARCHICAL SCHEDULING

A. Overview

Nowadays, there are a lot of implementation of hierarchical scheduling. We chose four well-known approaches of HSF for real-time systems on multicore platform offered by Shin et al.[16], Åsberg et al. [31], Checconi et al. [32] and Nemati et al [33].

1) Virtual Cluster HSF

The Virtual Cluster (VC) Hierarchical Scheduling Framework (VC-HSF) developed by Shin is a generalization of physical clustering with a new feature of sharing processors between different clusters. Unlike physical clusters, where processors are dedicated to a cluster offline, VC allows allocation of physical processors to the clusters during run-time. This dynamic allocation scheme requires an interface to capture the execution and concurrency requirements within a cluster to use hierarchical scheduling techniques. The interface proposed by Shin et al. which is known as the MPR model.

In VC-HSF for each cluster a MPR interface is generated using schedulability analysis presented in Shin et al. Then each of this interface is transformed into a set of implicit deadline periodic servers for inter cluster scheduling. The periodic

servers idle their budget if there is no active task running inside the server.

VC-HSF has two-level hierarchy of schedulers: inter-cluster and intra-cluster. Inter-cluster level provide dynamic allocation virtual clusters, i.e. assign physical processors to virtual cluster. Inter (external/global) assign sub-set of cores to each subsystem (group of tasks and server) and schedules on global EDF (G-EDF) [34]. The same is true for the inter-cluster scheduler of VC except that each cluster can have up to m active servers. All the servers from all the clusters are queued according to the global scheduling policy. Queue of tasks sorting by the shortest deadline and tasks allocate to cores, until take their full. However, tasks are not assigned to any particular server, rather these only belong to a specific cluster. Intra-cluster level provide scheduling on physical level within a cluster. All clusters is non-intersect, therefore clusters cannot interrupt each other. Intra (internal) schedules on G-EDF. The intra-cluster executes tasks of the cluster by consuming the budgets of its scheduled servers. Unlike regular hierarchical scheduling, the local or intra-cluster scheduler also has to use a multiprocessor global scheduling algorithm as there can be multiple active servers of a cluster. Architecture of VC-HSF is illustrated in Fig. 3.

As a result, VC can be described as global scheduling in two level. Easwaran et al. mentioned different global scheduling algorithms like G-EDF and McNaughton's algorithm [35] that can be used in VC.

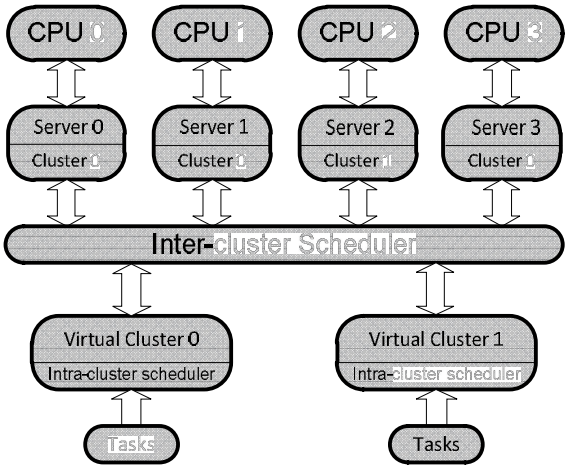

Fig. 3. VC-HSF

2) Checconi HSF

Checconi was proposed own approach of HSF that consists of two-level hierarchy schedulers: local and global. On local level tasks schedule with global multicore fixed priority. On global level each core have own Hard Constant Bandwidth Server (H-CBS) scheduler that scheduling a single server in each subsystem. Each subsystem have number of servers which equal number of cores. Each subsystem have access to all cores. Global scheduling runs parallel by H-CBS schedulers. H-CBS hard assigned on the core. Tasks within the server can migrate to another server that resides on a different core. The server has a share of each core at its disposal.

3) Nemati HSF

Another approach of HSF was implemented by Nemati et al. Their hierarchical scheduling has a scheme where the servers are scheduled with a global multicore scheduling scheme (fixed or dynamic priority), and locally, each subsystem is scheduled with partitioned multicore (fixed or dynamic priority) scheduling, i.e., each subsystem has maximum one server (that may run on any core), so tasks always execute on the same core.

4) Sequential HSF

Last implementation of hierarchical scheduling in our review is the Sequential HSF offered by Asberg et al. This approach provide scheduling servers in sequence, thereby scheduling tasks, within each subsystem, with global multicore on all cores. Also proposed scheme will execute the subsystems in sequential order, occupying each core with one server.

B. Comparison

VC-HSF, Checconi HSF, Nemati HSF and Sequential HSF were implemented for identical multiprocessors platform. The implementation complexity of these approaches differ in both the local and the global levels.

VC-HSF has the most complex server scheduler (global level), since the maximum number of cores that may be utilized, and the occupied cores must be checked when scheduling a server. Checconi HSF is more simple since all servers are assigned to a core statically offline. In fact, one (H-CBS) global scheduler could be sufficient to handle all servers (which would make it similar to our sequential approach). Nemati HSF is quite simple since there is maximum one server per subsystem, but the availability of cores need to always be checked since servers are not statically assigned to cores. Sequential HSF is simple since servers are statically assigned to cores.

Looking at the local scheduling, VC-HSF and Checconi HSF have similar local scheduling schemes. Both of them schedules global multicore scheduling, on a subset of cores. Nemati HSF uses simple partitioned scheduling (unicore scheduling), while Sequential HSF schedules global multicore scheduling on all cores. To summarize, VC-HSF and Checconi HSF have slightly more complex scheduling than the sequential approach, while Nemati HSF is the most simple.

Looking at shared resources, due to that Nemati HSF uses partitioned scheduling, shared resources within a subsystem becomes less complex with this approach, compared to the other three schemes.

Summary of comparing results of four HSF schemes by types of multicore scheduling and parallelism of servers is shown in Table I.

TABLE I. MULTICORE HIERARCHICAL SCHEDULING SCHEMES

Strategy	Multicore scheduling	Server parallelism	Assigning servers to core
VC-HSF	Cluster	Yes	Online
Checconi HSF	Global	Yes	Offline
Nemati HSF	Partitioned	Yes	Online
Sequential HSF	Global	No	Offline

V. Conclusion

In this paper, we reviewed many aspects of hierarchical scheduling for multicore systems that provides predictable timing and temporal isolation. We described main features of hierarchical scheduling, kinds of multiprocessors (identical, uniform heterogeneous and unrelated heterogeneous), types of multiprocessor scheduling (partitioned, global and hybrid), strategies of hierarchical scheduling (its features, advantages and backwards), standards and concepts (ARINC-653, TSP, IMA etc.). Also we described and compared several hierarchical scheduling approaches (VC-HSF, Checconi HSF, Nemati HSF, Sequential HSF) for identical multiprocessors, defined it's features, which may be important for a field of its use.

This theme is very important for many industry domains and have a great potential. Our further research is required for the boundary of the performance of hierarchical multiprocessor scheduling for various task types (implicit-/arbitrary-deadline) on various processor types (identical/heterogeneous/unrelated). Finally, the hierarchical scheduling theory with dependency constraints (task dependency/resource sharing) is bad developed. We would like to work on these problems and complete the hierarchical scheduling theory.

References

[1] K. Chakma, S. Debbarma, N. Kar, N. Debbarma, and T. Debbarma, "*A Hierarchical Scheduling Approach for Symmetric Multiprocessing Based Real Time Systems on VxWorks*," in Lecture Notes on Software Engineering, Vol. 1, No. 1, February 2013.

[2] B. Leiner, M. Schlager, R. Obermaisser, and B. Huber. *A comparison of partitioning operating systems for integrated systems.* In F. Saglietti and N. Oster, editors, SAFECOMP, volume 4680 of Lecture Notes in Computer Science, pages 342–355. Springer, 2007.

[3] A. Boudjadar, A. David, J. Kim, K. Larsen, U. Nyman, A. Skou, "*Schedulability and Energy Efficiency for Multi-core Hierarchical Scheduling Systems*," in Proceedings of the International Congres on Embedded Real Time Software and Systems ERTS, 2014.

[4] F. Nemati, "*Resource Sharing in Real-Time Systems on Multiprocessors*," Ph.D. dissertation, Malardalen University, May 2012.

[5] B. Akesson, A. Molnos, A. Hansson, J. Ambrose Angelo, and ˚ K. Goossens, "*Composability and Predictability for Independent Application Development, Verification, and Execution*," in Multiprocessor System-on-Chip — Hardware Design and Tool Integration. Springer, Dec 2010.

[6] M. Asberg and T. Nolte, S. Kato, "*Towards Partitioned Hierarchical Real-Time Scheduling on Multi-core Processors*," in Proc. of the 1st International Workshop on Virtualization for Real-Time Embedded Systems, 2013.

[7] ARINC, ARINC 653: Avionics Application Software Standard Interface (Draft 15). Airlines Electronic Engineering Committee (AEEC), 1996.

[8] Krishna Kavi and Robert Akl, "*Real-Time Systems: An Introduction and the State-ofthe-art*", Wiley Encyclopedia of Computer Science and Engineering, 2008

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

[10] 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

[11] C. Liu and J. Layland, "*Scheduling Algorithms for Multi-Programming in a Hard-Real-Time Environment*," ACM, vol. 20, no. 1, pp. 46–61, 1973

[12] B. Brandenburg, "*Scheduling and Locking in Multiprocessor Real-Time Operating Systems*," Ph.D. dissertation, University of North Carolina at Chapel Hill, 2011

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

[14] R. I. Davis and A. Burns. *A survey of hard real-time scheduling for multiprocessor systems.* ACM Comput. Surv., 43(4):1–44, 2011

[15] K. Lakshmanan, R. Rajkumar, and J. P. Lehoczky. *Partitioned fixed-priority preemptive scheduling for multicore processors.* Real-Time Systems, 2009. ECRTS '09. 21st Euromicro Conference on, pages 239–248, January

[16] J. M. Calandrino, J. H. Anderson, and D. P. Baumberger. *A hybrid real-time scheduling approach for large-scale multicore platforms.* Real-Time Systems, 2007. ECRTS '07. 19th Euromicro Conference on, pages 247–258, April

[17] I. Shin, A. Easwaran, and I. Lee. *Hierarchical scheduling framework for virtual clustering of multiprocessors.* In Proceedings of the 2008 Euromicro Conference on Real-Time Systems, pages 181–190. IEEE Computer Society, 2008

[18] A. Easwaran, I. Shin, and I. Lee. *Optimal virtual cluster-based multiprocessor scheduling.* Real-Time Syst., 43(1):25–59, 2009

[19] N. Kim, J. Erickson, and J. Anderson, "*Mixed-Criticality on Multicore (MC2): A Status Report*", Proceedings of the 10th Annual Workshop on Operating Systems Platforms for Embedded Real-Time Applications, pp. 45-50, July 2014.

[20] E. Cannella, M. A. Bamakhrama, and T. Stefanov, "*System-level scheduling of real-time streaming applications using a semi-partitioned approach*," in DATE, 2014.

[21] M. Asberg, "*On the Development of Hierarchical Real-Time Systems*," Licentiate Thesis, Mälardalen University, Sweden, 2012.

[22] M. Asberg and T. Nolte, S. Kato, "*A Loadable Task Execution Recorder for Hierarchical Scheduling in Linux*," in Proc. of the Embedded and Real-Time Computing Systems and Applications (RTCSA), vol. 1, 2011.

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

[24] J. Craveiro, J. Rufino, and F. Singhoff, "*Architecture, mechanisms and scheduling analysis tool for multicore time- and space-partitioned systems*," ACM SIGBED Rev., vol. 8, no. 3, pp. 23–27, Sep. 2011, special issue of ECRTS 2011 WiP session, Porto, Portugal, July 2011.

[25] J. Rushby, "*Partitioning in avionics architectures: Requirements, mechanisms and assurance*," SRI Int'l., California, USA, NASA Contractor Report CR-1999-209347, 1999.

[26] AEEC, "*Design guidance for Integrated Modular Avionics*," Aeronautical Radio, Inc., ARINC Report 651-1, Nov. 1997.

[27] J. Windsor and K. Hjortnaes, "*Time and space partitioning in spacecraft avionics*," in Proceedings of the 3rd IEEE International Conference on Space Mission Challenges for Information Technology, Pasadena, CA, USA, Jul. 2009, pp. 13–20

[28] J. Rufino, J. Craveiro, and P. Verissimo, "*Architecting robustness and timeliness in a new generation of aerospace systems*," in Architecting Dependable Systems VII, ser. Lecture Notes in Computer Science, A. Casimiro, R. de Lemos, and C. Gacek, Eds. Springer, 2010, vol. 6420, pp. 146– 170.

[29] J. Craveiro and J. Rufino, "*Adaptability support in timeand space-partitioned aerospace systems*," in Proceedings of the Second International Conference on Adaptive and Self-adaptive Systems and Applications (ADAPTIVE 2010), Lisbon, Portugal, Nov. 2010.

[30] J. Craveiro, "*Real-Time Scheduling in Multicore Time- and Space-Partitioned Architectures*," Ph.D. dissertation, Universidade de Lisboa, Portugal, 2013.

[31] M. Åsberg, T. Nolte, and S. Kato, *Towards Hierarchical Scheduling in Linux/Multi-core Plateform*, MRTC Mälardalen University, Technische Universiteit Eindhoven, Sweden, The University of Tokyo,2010.

[32] F. Checconi, T. Cucinotta, D. Faggioli, and G. Lipari, "*Hierarchical Multiprocessor CPU Reservations for the Linux Kernel*," in Proc. of the 5th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications, June 2009.

[33] F. Nemati, M. Behnam, and T. Nolte, "*Multiprocessor Synchronization and Hierarchical Scheduling,"* in Proc. of the 1st International Workshop on Real-time Systems on Multicore Platforms: Theory and Practice, in conjunction with ICPP'09, September 2009

[34] M. Peloquin, "*A Comparison of Scheduling Algorithms for Multiprocessors*", December 2010

[35] S. Abdullah, "*Virtual Clustered-based Multiprocessor Scheduling in Linux Kernel*", Ph.D. dissertation, Malardalen University, June 2013