GeMRTOS: diagramación de particiones híbridas para sistemas multiprocesador embebidos.

Ricardo Cayssials

Universidad Tecnológica Nacional – Facultad Regional Bahía Blanca
Universidad Nacional del Sur
rcayssials@frbb.utn.edu.ar

Resumen. La teoría de tiempo real es una teoría madura para sistemas monoprocesador. Los mecanismos de diagramación que son eficientes u óptimas en sistemas monoprocesador pueden no serlo en sistemas multiprocesador. La extensión de los mecanismos de tiempo real monoprocesador a sistemas multiprocesador se basan tanto en una partición local, en la cual el conjunto de tareas se divide entre los procesadores del sistema; en una particiones global, donde el conjunto de tareas se diagrama sobre el conjunto de procesadores; o en la diagramación de particiones mixtas, en la que las tareas se diagraman en un subconjunto de procesadores. Estos enfoques de partición se basan en un concepto similar al de un sistema con múltiples monoprocesador, en lugar de un sistema multiprocesador. GeMRTOS es un sistema operativo en tiempo real genérico para sistemas embebidos que introduce listas de diagramación híbridas para una asignación flexible de tareas y procesadores. Las listas de diagramación permiten un mecanismo en el que tanto las tareas como los procesadores se asignan a listas de diagramación para facilitar la implementación de todos los subsistemas. Este artículo describe la diagramación de particiones híbridas innovadora y dirigida por eventos introducida por GeMRTOS.

Palabras clave: Multiprocessor RTOS, Partition scheduling, Embedded Systems.

GeMRTOS: hybrid partition scheduling for embedded multiprocessor systems

Ricardo Cayssials

Universidad Tecnológica Nacional – Facultad Regional Bahía Blanca
Universidad Nacional del Sur
rcayssials@frbb.utn.edu.ar

Abstract. Real-time theory is a mature theory for uniprocessor systems. Scheduling policies that are efficient or optimal in uniprocessor systems may not be so in multiprocessor systems. Extending real-time uniprocessor mechanisms to multiprocessor systems are based on either a local partition in which the set of tasks is partitioned among the system processors, a global partition scheduling in which the set of tasks is schedule over the set of processors or mixed partition scheduling in which task are schedule over a subset of processors. These partition approaches are based on a concept akin to a system with multiple uniprocessors rather than a multiprocessor system. GeMRTOS is a generic multiprocessor RTOS for embedded systems that introduces hybrid scheduling lists for a flexible assignment of tasks and processors. The scheduling lists enable a hybrid partition scheduling mechanism, in which tasks as well as processors are assigned to scheduling lists to make the implementation of all subsystems easier. This paper describes the innovative, event-driven hybrid partition schedule introduced by GeMRTOS.

Keywords: Multiprocessor RTOS, Partition scheduling, Embedded Systems.

I. INTRODUCTION

Embedded real-time systems are found in many application areas with rapid technological progress. In Moore's law era (Moore, 2003), future processors should meet the requirements of more demanding applications. However, power consumption and excessive heat dissipation limited the increase in processor speeds, leading to the use of multiprocessor systems for high-end real-time applications (Hennessy & Patterson, 2019).

Real-time scheduling theory is a mature research area with important results for uniprocessor systems (Davis & Burns, 2011); however, "Few of the results obtained for a single processor generalize directly to the multiple processor case; bringing in additional

processors adds a new dimension to the scheduling problem. The simple fact that a task can use only one processor even when several processors are free at the same time adds a surprising amount of difficulty to the scheduling of multiple processors."

Earliest deadline first (EDF) and fixed priority (FP) are two well-established priority scheduling algorithms for real-time uniprocessor systems. However, they present some anomalies when they are applied to multiprocessor systems (Andersson et al., 2001; Graham, 1971; Lauzac et al., 1998). Therefore, real-time uniprocessor scheduling results cannot be straightforwardly applied to multiprocessor systems. Rather than considering an increase in the number of processors of the system, real-time theory introduces *resource augmentation* or a *speedup factor* (Kalyanasundaram & Pruhs, 2000) to determine the increase in processing speed that would be required to schedule a real-time system.

Modern embedded systems are not just a set of tasks that must be executed "faster" but a set of subsystems, each one consisting of several tasks, which must be integrated into the system in order to implement different functionalities. Real-time tasks, user interfaces, communication support, update and upgrade support, and edge processing are some examples of functionalities that a modern embedded system should include. Most of these functionalities are implemented using off-the-shelf software and are integrated as subsystems, each one including several tasks. Under these current premises, speeding up an embedded uniprocessor system until the system meets its functionalities makes no sense. Embedded multiprocessor systems allow (1) increasing the processing power by increasing the number of processors and (2) sharing the processing power more adequately among the subsystems according to their processing needs. More processors may be assigned to the more demanding subsystems.

In operating system theory, scheduling policies in multiprocessor systems can be local partitioned or global (Phillips et al., 2002). In a local partitioned scheduling policy, a task is assigned to a processor, and it is scheduled with all the tasks assigned to the same processor. No migration is allowed in a partitioned scheduling policy. On the other hand, when migration is allowed, the scheduling policy is global. In global scheduling, no processor remains idle when a task requires execution, and the highest demanding tasks must share the processor with all the tasks of the system. In semi-partitioned scheduling (Al-bayati et al., 2015), some tasks are statically assigned to processors, while other tasks are allowed to migrate.

GeMRTOS (US Patent 11.321.126B2) (*G-RTOS/GeMRTOS*, 2021/2021) implements the scheduling mechanism based on scheduling lists that allow implementing the innovative *hybrid scheduling*. Each task is assigned to a scheduling list, statically or dynamically, and the processors are assigned to one or more scheduling lists. In this way, partitioned scheduling may be partially implemented, and no processor remains idle when a task requires execution in the scheduling lists assigned to it.

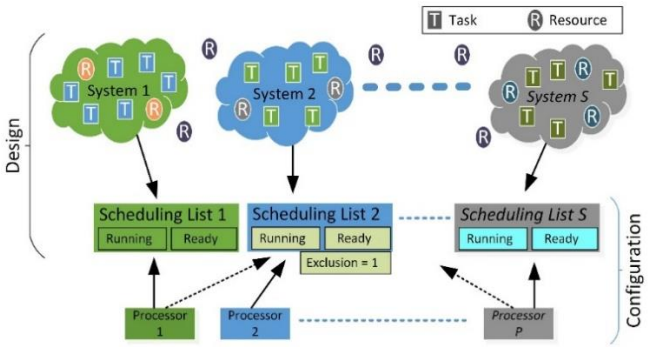

Fig. 1. Layout of the GeMRTOS scheduling (*GeMRTOS*, n.d.).

Fig.1 shows the structure in which tasks and processors are assigned to scheduling lists, allowing a flexible mechanism for scheduling tasks in embedded multiprocessor systems. GeMRTOS takes the migration concept to a higher abstraction level since it is possible to migrate tasks, as well as processors, between different scheduling lists.

In this paper, the GeMRTOS hybrid scheduling is described as an overarching mechanism for designing embedded multiprocessor systems.

The paper is organized as follows. Section II outlines the control blocks and the linked lists that support the GeMRTOS event-driven methodology. The GeMRTOS scheduling lists are described in Section III. Section IV details the data structures based on highly linked lists that hold the state of the whole system. The GeMRTOS code structure and function callbacks are briefly described in Section V. Section VI explains the GeMRTOS runtime event management. Remarks on the GeMRTOS hybrid scheduling are given in Section VII. Conclusions are drawn in Section VIII.

II. Real-time system model and multiprocessor anomalies

Real-time systems are modeled as a set $\mathbf{\Pi} = \{\tau_1, \tau_2, ..., \tau_n\}$ of n tasks scheduled across m processors. A uniprocessor system has $m = 1$; a multiprocessor system has $m > 1$.

Each real-time task τ_i is defined by its period (T_i), worst-case execution time (C_i), and deadline (D_i). T_i is the minimum time between task releases; each release generates a new instance requiring at most C_i processing time. A task is schedulable if its instance completes before its deadline D_i, typically equal to T_i. [14]

A task is ready when it has an uncompleted instance. The scheduler selects the m highest-priority ready tasks for execution on available processors. The priority discipline determines task priorities.

Fixed-Priority (FP) scheduling assigns priorities at design time. Rate-Monotonic (RM) is an FP scheme prioritizing shorter-period tasks. RM is optimal for uniprocessor FP

scheduling: if a system is unschedulable under RM, it's unschedulable under any other FP scheme. A uniprocessor system's schedulability under FP is determined by:

$$\forall i \in \{1,2,\dots,N\}, \exists t \text{ such that } C_i + \sum_{h=1}^{i-1} C_h \left\lceil \frac{t}{T_h} \right\rceil \leq D_i \quad (1)$$

where ⌈ ⌉ denotes the ceiling function. Equation (1) ensures sufficient processing time for each task τ_i, accounting for higher-priority task invocations before its deadline D_i.

Earliest Deadline First (EDF) prioritizes tasks based on their deadlines. EDF is optimal for uniprocessor scheduling; unschedulability under EDF implies unschedulability under any other scheme. A uniprocessor system's schedulability under EDF requires:

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1 \quad (2)$$

where C_i/T_i represents the utilization factor of task τ_i.

Equations (1) and (2) show that uniprocessor schedulability can improve by:

- increasing task periods (T_i), and/or
- reducing worst-case execution times (C_i).

Both alternatives reduce the total utilization factor and consequently an uniprocessor system may transition from unschedulable to schedulable, or maintain schedulability following these options (Kalyanasundaram & Pruhs, 2000; Phillips et al., 2002; Zhao et al., 2022). However, these criteria don't hold for multiprocessor systems due to real-time anomalies.

Consider three real-time tasks ($D_i = T_i$) [9]:

Table 1. Example for real-time anomaly.

Task	Period (T)	Worst-Case Execution Time (C)
τ1	4	1
τ2	5	1
τ3	20	18

The periods and worst-case execution times are expressed in arbitrary units. Figure 1 depicts the task scheduling on two processors, showing that all deadlines are met.

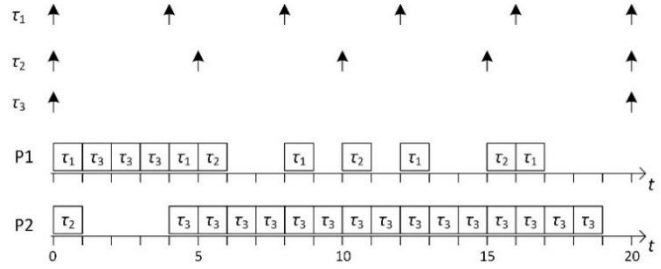

Fig. 2. Schedulable system on a two-processor system.

Increasing τ_1's period from 4 to 5 demonstrates the period anomaly: increased period leads to unschedulability.

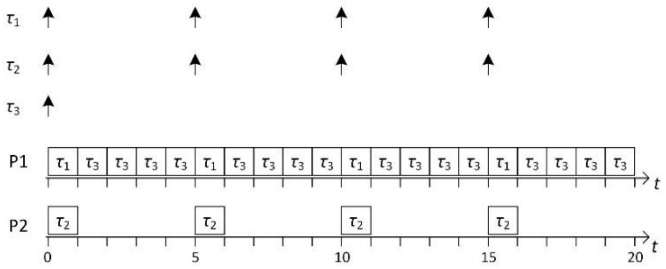

Fig. 3. Unschedulable system on a two-processor system.

Figure 3 shows that, with the modified period for τ_1, task τ_3 misses its deadline due to only 16 time units being available before its deadline. The simultaneous release of τ_1 and τ_2 leaves no processor capacity for τ_3, illustrating the significant challenges of directly applying uniprocessor schedulability analysis to multiprocessor systems.

GeMRTOS addresses this challenge with its innovative hybrid scheduling approach, detailed in a later section.

III. Real-time Mixed-Criticality Systems and Multiprocessor Challenges

Integrating subsystems of various criticality levels onto an embedded platform is increasingly popular in real-time systems design [18], [19], [20]. Criticality refers to the assurance level needed against failure for a system task. Multiprocessor architectures provide potential benefits for implementing applications with different criticality levels, facilitating resource sharing among subsystems. However, optimizing multiprocessor performance while ensuring real-time schedulability in mixed-criticality systems is a challenging task.

Local partitioning approaches often guarantee real-time schedulability but can adversely affect system performance, as processors might remain idle despite having ready tasks. Conversely, multiprocessor RTOSs, evolved from uniprocessor architectures, allow real-time tasks to run across multiple processors, enhancing system performance but lacking a systematic approach for implementing mixed-criticality systems.

The complexity of modern systems requires flexible multiprocessor architectures capable of accommodating various subsystems, each with unique processor requirements [14]. Real-time system design must progress beyond seeing systems as mere collections of real-time tasks. Instead, they should be viewed as systems composed of mixed-criticality components, each containing multiple real-time tasks. The multiprocessor architecture

should consider these components as subsystems, aligning the application's real-time criticality needs with system performance goals.

Multiprocessor systems introduce new challenges in real-time system design, particularly regarding the choice between local and global partitioned scheduling. Local partitioned scheduling allows utilizing uniprocessor real-time theory, wherein the system can be examined as a series of independent uniprocessor systems. Global partitioned scheduling enhances system efficiency by ensuring no processor is idle when a task can be executed. However, straightforward application of uniprocessor real-time results is complex due to real-time anomalies.

In uniprocessor systems, resource-sharing mechanisms are relatively simple. Preventing concurrent task execution can be managed by increasing task priority making the task non-preemptive. However, this can worsen priority inversion, as higher-priority tasks must wait until the executing task finishes. Although synchronization mechanisms can mitigate priority inversion, they add to application complexity and decrease runtime performance.

Multiprocessor systems require advanced synchronization. Simply disabling preemption doesn't ensure exclusive access to critical sections, particularly in global or partially global scheduling schemes. Multiple tasks could compete simultaneously for a single resource if they are running on different processors. Thus, effective management of resource contention in multiprocessor systems demands sophisticated synchronization techniques.

IV. The GeMRTOS hardware architecture

The GeMRTOS architecture is a sophisticated system designed to optimize the coordination and efficiency of the multiprocessor architecture. At its core, it comprises N processors that share access to a common memory and M input/output (I/O) devices, all facilitated through a unified system bus. This architecture is anchored by the GeMRTOS controller, which plays a crucial role in managing several key functions. These include the synchronization of system time, ensuring mutual exclusion among processors to prevent conflicts, and efficiently handling external events. This controller is integral to maintaining the overall stability and performance of the system, which is crucial in real-time processing environments. Figure 3 typically illustrates this architectural setup, showcasing the central role and functionality of the GeMRTOS controller within the multiprocessor system [21], [22].

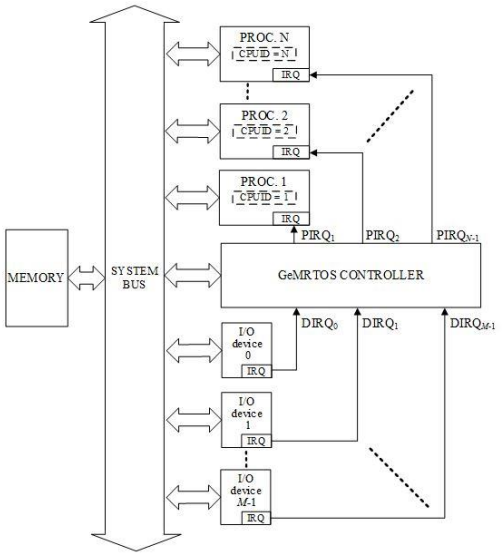

Fig. 4. GeMRTOS hardware architecture (Cayssials, 2022).

The GeMRTOS controller operates as the central hub of the system, efficiently managing communication between peripheral input/output (I/O) devices and the processors. It achieves this by receiving device interrupt requests (DIRQs) from I/O devices and issuing processor interrupt requests (PIRQs) to the processors, fundamentally altering the traditional role of processors. In contrast to uniprocessor and extended multiprocessor systems where processors are actively involved in interrupt handling, GeMRTOS relegates them to the role of code execution units. This shift allows the controller to handle tasks such as interrupt management, thus enabling processors to focus solely on executing code.

When processors do not have tasks to execute, the GeMRTOS controller issues a wait-request signal, transitioning them into a wait state and effectively halting all activity. This approach optimizes power consumption, as processors are active only during the execution of necessary code, thereby avoiding unnecessary energy use associated with device polling or redundant low-power state transitions.

A notable feature of the GeMRTOS controller is its scalability, which allows it to manage an arbitrary number of processors and external events. Its functionality is highly configurable, thanks to a set of internal registers that can be configured via the system bus. This flexibility ensures that the GeMRTOS controller can be customized to suit the specific requirements of a broad spectrum of multiprocessor real-time applications, ranging from small-scale embedded systems to large, complex mixed-criticality systems.

V. GeMRTOS hybrid scheduling lists in real-time systems

GeMRTOS presents an innovative hybrid scheduling approach aimed at optimizing task execution across multiple processors through the use of hybrid scheduling lists. In this system, tasks can be assigned to these lists either statically or dynamically, providing flexibility in task management. Processors are then allocated to one or more of these lists. This adaptable assignment strategy supports a partial implementation of local/global partitioned scheduling, which is instrumental in facilitating real-time design and ensuring effective utilization of processors.

The hybrid scheduling lists in GeMRTOS allow for more nuanced and efficient task distribution, accommodating the varying demands of real-time applications. This method enhances the system's ability to balance workload and maintain high levels of processor efficiency. Figure **1** typically depicts this task and processor assignment methodology, showcasing how the hybrid scheduling list system operates within the GeMRTOS framework.

In GeMRTOS, each processor is associated with a primary scheduling list, referred to as the foreground list, and may also have additional background lists. A processor is considerd free when there are no tasks requiring execution in its foreground list, thereby enabling it to execute tasks from the background lists. When there are no tasks available in any of the associated scheduling lists, the GeMRTOS controller places the processor into a wait state to conserve energy.

GeMRTOS provides the flexibility for multiple processors to be assigned to the same scheduling list. To address potential real-time anomalies and ensure optimal processor utilization, an exclusion parameter is defined. This parameter helps to maximize processing power by preventing idle processors when tasks are ready to be executed. The exclusion feature offers a dynamic mechanism for load balancing, effectively managing real-time anomalies within multiprocessor systems.

This strategy stands in contrast to traditional approaches in other operating systems, which often require pausing other processors to prevent the execution of lower-priority tasks during the execution of higher-priority tasks [23], [24]. The GeMRTOS method thus promotes a more seamless and efficient task management system, enhancing real-time performance and processor efficiency.

VI. Managing Real-Time Schedulability with GeMRTOS's Hybrid Scheduling Lists

Directly applying uniprocessor scheduling theory to multiprocessor systems can introduce unexpected challenges, as depicted in Figure 3. In this scenario, adjusting the period of $\tau 1$ caused a scheduling conflict, leading to $\tau 3$ missing its deadline. A traditional solution might involve local partitioning, where $\tau 1$ and $\tau 2$ are assigned to one processor while $\tau 3$ is

allocated to another. However, this approach can restrict system flexibility and potentially degrade system performance and load balancing.

GeMRTOS addresses these challenges through the implementation of hybrid scheduling lists, which operate as follows in this scenario:

- **List 1:** Tasks τ_1 and τ_2 are assigned in this list.
- **List 2:** Task τ_3 is assigned to a separate list.

Processors are then strategically allocated to these lists as follows:

- List 1: Both processors are assigned to List 1, with one acting as a background processor. This setup ensures that the background processor executes tasks from List 1 only when no tasks from List 2 are available.
- List 2: At least one processor is dedicated to List 2 as its foreground list, ensuring it prioritizes executing tasks from List 2 and maximizing efficiency for those tasks.

This flexible configuration accomplishes two primary objectives:

1) *Guaranteeing τ_3 schedulability*: By having a processor assigned as foreground list for **List 2**, GeMRTOS ensures that τ_3 will always have processing resources available, thus guaranteeing its schedulability even when tasks in **List 1** demand more processing time.
2) *Improving multiprocessor performance*: In the absence of ready tasks in **List 2**, the background processor from **List 1** can efficiently execute tasks from that list, thereby maximizing processor utilization and enhancing overall system performance.

This simple example shows the robust flexibility of GeMRTOS's hybrid scheduling lists. By dynamically managing task and processor allocations, GeMRTOS surmounts the constraints of rigid local or global partitioning, offering a more adaptable solution to real-time scheduling that enhances both schedulability and mixed-criticality performance in multiprocessor systems.

VII. remarks on GeMRTOS hybrid scheduling

The GeMRTOS hybrid scheduling allows the system to be conceived at a higher abstraction level. The system may be designed as subsystems to cope with the different functionalities and processing powers that each one requires. Moreover, free processing power may be configured to serve the background scheduling lists.

Each scheduling list may implement any arbitrary priority discipline. It only depends on how the task priorities are assigned when the events associated with the ready and running lists take place.

The exclusion of processor assignment to scheduling lists is supported. It can be used for load balancing or the avoidance of real-time anomalies in real-time subsystems, while preserving processor sharing among subsystems.

VIII. Conclusions

Several alternatives were proposed for real-time multiprocessor architectures. Local versus global scheduling policies, partitioned versus task migration assignments, are some of the alternatives in multiprocessor systems. However, these concepts focus on multiprocessor architectures as a set of uniprocessor systems. Multiprocessor systems need not to be conceived with the limitations of uniprocessor systems and adequate for mixed-criticality specifications.

The GeMRTOS hybrid scheduling allows managing both the task and the processor assignments. The whole application can be abstractly conceived as one or many sets of tasks running in several multiprocessor subsystems. Processors may serve several scheduling lists, and scheduling lists may be served by one or several processors.

In this paper, the main features of the GeMRTOS hybrid scheduling were described. It is shown to be an adequate and flexible scheduling scheme for mixed-criticality real-time multiprocessor architectures. Flexibility allows implementing modern strategies suitable for real-time embedded multiprocessor systems.

References

Al-bayati, Z., Zhao, Q., Youssef, A., Zeng, H., & Gu, Z. (2015). Enhanced partitioned scheduling of Mixed-Criticality Systems on multicore platforms. The 20th Asia and South Pacific Design Automation Conference, 630–635. https://doi.org/10.1109/ASPDAC.2015.7059079

Andersson, B., Baruah, S., & Jonsson, J. (2001). Static-priority scheduling on multiprocessors. Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420), 193–202. https://doi.org/10.1109/REAL.2001.990610

Cayssials, R. L. (2022). United States Patent: 11321126 - Multiprocessor system for facilitating real-time multitasking processing (Patent 11321126).

Davis, R. I., & Burns, A. (2011). A Survey of Hard Real-time Scheduling for Multiprocessor Systems. ACM Comput. Surv., 43(4), 35:1-35:44. https://doi.org/10.1145/1978802.1978814

GeMRTOS. (n.d.). GitHub. Retrieved 21 August 2024, from https://github.com/G-RTOS/GeMRTOS/blob/a51cd4b62ce642248304ad72253db008a59096b6/MANUAL.md

Graham, R. L. (1971). Bounds on multiprocessing anomalies and related packing algorithms. Proceedings of the May 16-18, 1972, Spring Joint Computer Conference, 205–217. https://doi.org/10.1145/1478873.1478901

G-RTOS/GeMRTOS. (2021). [HTML]. GeMRTOS. https://github.com/G-RTOS/GeMRTOS (Original work published 2021)

Hennessy, J. L., & Patterson, D. A. (2019). A New Golden Age for Computer Architecture. Commun. ACM, 62(2), 48–60. https://doi.org/10.1145/3282307

Kalyanasundaram, B., & Pruhs, K. (2000). Speed is as powerful as clairvoyance. Journal of the ACM, 47(4), 617–643. https://doi.org/10.1145/347476.347479

Lauzac, S., Melhem, R., & Mosse, D. (1998). Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multiprocessor. Proceeding. 10th EUROMICRO Workshop

on Real-Time Systems (Cat. No.98EX168), 188–195. https://doi.org/10.1109/EMWRTS.1998.685084

Moore, G. E. (2003). No exponential is forever: But 'Forever' can be delayed! 2003 IEEE International Solid-State Circuits Conference, 2003. Digest of Technical Papers. ISSCC., 20–23 vol.1. https://doi.org/10.1109/ISSCC.2003.1234194

Phillips, Stein, Torng, & Wein. (2002). Optimal Time-Critical Scheduling via Resource Augmentation. Algorithmica, 32(2), 163–200. https://doi.org/10.1007/s00453-001-0068-9

Zhao, T., Li, W., & Zomaya, A. Y. (2022). Real-Time Scheduling with Predictions. 2022 IEEE Real-Time Systems Symposium (RTSS), 331–343. https://doi.org/10.1109/RTSS55097.2022.00036