Aperiodic OS Tasks Scheduling for Hard-Real-Time Reconfigurable Uniprocessor Systems

Tarek Amari[3], Hamza Gharsellaoui[1], Mohamed Khalgui[1,2], Samir Ben Ahmed[1,3]
[1]Laboratory of Computing for the Industrial Systems (LISI), INSAT Institute, Tunisia
[2]ITIA Institute - CNR Research Council, Italy
[3]FST Faculty - University of Tunis El Manar, Tunisia

abstract The scheduling of tasks is an essential requirement in most real-time and embedded systems, but invariably leads to unwanted CPU overheads. This paper presents real-time scheduling techniques for reducing the response time of aperiodic tasks scheduled with real-time periodic tasks on uniprocessor systems. Two problems are addressed in this paper: (i) the scheduling of aperiodic when they arrive in order to obtain a feasible system , and (ii) the scheduling of periodic and aperiodic tasks to minimize their response time. In order to improve the responsiveness to both types of problems, efficient hybrid approach is proposed based on the combination of the Polling Server (PS) and the Background Server (BS). The effectiveness and the performance of the designed approach is evaluated through simulation studies.

1 INTRODUCTION

Real-time systems are used to control physical processes that range in complexity from automobile ignition systems to controllers for flight systems and nuclear power plants. In these systems, the correctness of system functions depends upon not only the results of computation but also on the times at which results are produced. A real-time task is generally placed into one of four categories based upon its arrival pattern and its deadline. If meeting a given task's deadline is critical to the system's operation, then the task's deadline is considered to be hard. If it is desirable to meet a task's deadline but occasionally missing the deadline can be tolerated, then the deadline is considered to be soft. Tasks with regular arrival times are called periodic tasks. A common use of periodic tasks is to process sensor data and update the current state of the real-time system on a regular basis. Periodic tasks, typically used in control and signal-processing applications, have hard deadlines. Tasks with irregular arrival times are aperiodic tasks. Aperiodic tasks are used to handle the processing requirements of random events such as operator requests. An aperiodic task typically has a soft deadline. Aperiodic tasks that have hard deadlines are called sporadic tasks. We assume that each task has a known worst-case execution time. In summary, we have Hard and soft deadline periodic tasks. A periodic task has a regular interarrival time equal to its period and a deadline that coincides with the end of its current period. Periodic tasks usually have hard deadlines, but in some applications the deadlines can be soft. Soft deadline aperiodic tasks. An aperiodic task is a stream of jobs arriving at irregular intervals. Soft deadline aperiodic tasks typically require a fast average response time. Sporadic tasks. A sporadic task is an aperiodic task with a hard deadline and a minimum interarrival time (Mok 1983). Note that without a minimum interarrival time restriction, it is impossible to guarantee that a sporadic task's deadline would always be met. To meet the timing constraints of the system, a scheduler must coordinate the use of all system resources using a set of well-understood real-time scheduling algorithms that meet the following objectives: Guarantee that tasks with hard timing constraints will always meet their deadlines. Attain a high degree of schedulable utilization for hard deadline tasks (periodic and sporadic tasks). Schedulable utilization is the degree of resource utilization at or below which all hard deadlines can be guaranteed. The schedulable utilization attainable by an algorithm is a measure of the algorithm's utility: the higher the schedulable utilization, the more applicable the algorithm is for a range of real-time systems. Provide fast average response times for tasks with soft deadlines (aperiodic tasks). Ensure scheduling stability under transient overload. In some applications, such as radar tracking, an overload situation can develop in which the computation requirements of the system exceed the schedulable resource utilization. A scheduler is said to be stable if during overload it can guarantee the deadlines of critical tasks even though it is impossible to meet all task deadlines. The quality of a scheduling algorithm for real-time systems is judged by how well the algorithm meets these objec-

tives. This article develops advanced hybrid approach to schedule aperiodic tasks. For soft deadline aperiodic tasks, the goal is to provide fast average response times. For hard deadlines aperiodic tasks (sporadic tasks), the goal is to guarantee that their deadlines will always be met. The new hybrid approach presented here meet both of these goals and are still able to guarantee the deadlines of hard deadline periodic tasks. Each periodic task τ_i is characterized according to [2], by an initial offset S_i (a release time), a worst-case execution time C_i, a relative deadline D_i and a period T_i. Each aperiodic task τ_i is characterized by a worst-case execution time C_i and a relative deadline D_i. A task is synchronous if its release time is equal to 0. Otherwise, it's asynchronous. We assume in this work that all the tasks are independent, periodic and aperiodic. A tool named RT-Reconfiguration is developed in our research laboratory at INSAT university to support this new proposed approach. The organization of this original paper is as follows. The next section formalizes some known concepts in the real-time scheduling theory, section III presents the state of the art. In section IV, we define a new theoretical approach. In section V, our proposed approach is implemented, simulated and analyzed. Finally, section VI presents a summary and conclusions of this paper.

2 SYSTEM MODEL

We present the following well-known concepts in the theory of aperiodic real-time scheduling [2]:

- An aperiodic task τ_i $(C_i; D_i)$ is an infinite collection of jobs that have their request times constrained by a Worst Case Execution Time (WCET) C_i and a relative deadline D_i,
- Deadline: The time when a task must be finished executing.
- Worst Case Execution Time (WCET): The longest possible execution time for a task on a particular type of system.
- Response time: The time it takes a task to finish execution. Measured from release time to execution completes, including preemptions.
- Preemptive scheduling: an executing task may be interrupted at any instant in time and have its execution resumed later.
- Release/ready time: The time a task is ready to run and just waits for the scheduler to activate it.
- A busy period is defined as a time interval $[a, b)$ such that there is no idle time in $[a, b)$ (the processor is fully busy) and such that both a and b are idle times,
- $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$ is the processor utilization factor. In the case of synchronous and asynchronous, independent and periodic tasks. $U = \sum_{i=1}^{n} \frac{C_i}{min(T_i, D_i)} \leq 1$ is a sufficient condition but not necessary for the EDF-based scheduling of real time tasks.
- A hard real-time task is never allowed to miss a deadline because that can lead to complete failure of the system. A hard real-time task can be safety-critical and this means that if a deadline is missed it can lead to catastrophically consequences which can harm persons or the environment.
- A soft real-time task is a task when a deadline is allowed to be missed, while there is no complete failure of the system it can lead to decreased performance.
- **Polling Server** is a periodic task whose purpose is to service aperiodic requests with a period T_S, a computation time C_S (capacity) and scheduled in the same way as periodic tasks.
- **Background Server** schedules aperiodic tasks in background (when no periodic task is running) and schedule of periodic tasks is not changed.

3 STATE OF THE ART

A real-time system often has both periodic and aperiodic tasks. Lehoczky, Sha, and Strosnider (1987) in [3] developed the Deferrable Server algorithm, which is compatible with the rate monotonic scheduling algorithm and provides a greatly improved average response time for soft deadline aperiodic tasks over polling or background service algorithms while still guaranteeing the deadlines of periodic tasks. The scheduling problem for aperiodic tasks is very different from the scheduling problem for periodic tasks. Scheduling algorithms for aperiodic tasks must be able to guarantee the deadlines for hard deadline aperiodic tasks and provide good average response times for soft deadline aperiodic tasks even though the occurrences of the aperiodic requests are nondeterministic. For a detailed analysis of aperiodic servers see [4] and [5]. The aperiodic scheduling algorithm must also accomplish these goals without compromising the hard deadlines of the periodic tasks. For the aperiodic scheduling, authors presented Slack stealing [8] and aperiodic servers, such as the sporadic server [6] and the deferrable server [7], allow aperiodic tasks to be

handled within a periodic task framework. Our approach try by allowing periodic tasks to be handled with an aperiodic ones by an hybrid approach in the same framework. To the author's knowledge, no result is available in the state of the art for scheduling both periodic and aperiodic tasks, except that we propose in our original work where an approach to deal with complex timing constraints and with minimizing the response time is proposed.

4 APERIODIC TASK SCHEDUI ING

The scheduling problem for aperiodic tasks is very different from that for periodic tasks. Scheduling algorithms for aperiodic tasks must be able to guarantee the deadlines for hard deadline aperiodic tasks and provide good average response times for soft deadline aperiodic tasks even though the occurrence of the aperiodic requests are nondeterminstic. The aperiodic scheduling algorithm must also accomplish these goals without compromising the hard deadlines of the periodic tasks.

4.1 Contribution

One hybrid approach composed of the combination of two common approaches for servicing aperiodic requests are background processing and polling tasks. Background servicing of aperiodic requests occurs whenever the processor is idle (i.e., not executing any periodic tasks and no periodic tasks pending). If the load of the periodic task set is high, then utilization left for background service is low, and background service opportunities are relatively infrequent. Polling consists of creating a periodic task for servicing aperiodic requests. At regular intervals, the polling task is started and services any pending aperiodic requests. However, if no aperiodic requests are pending, the polling task suspends itself until its next period and the time originally allocated for aperiodic service is not preserved for aperiodic execution but is instead used by periodic tasks. Note that if an aperiodic request occurs just after the polling task has suspended, then the aperiodic request must wait until the beginning of the next polling task period or until background processing resumes before being serviced. Even though polling tasks and background processing can provide time for servicing aperiodic requests, they have the drawback that the average wait and response times for these algorithms can be long, especially for background processing. Figure 2 illustrates the operation of background and polling aperiodic service using the periodic task set presented in the table of the same picture (Figure 1).

4.2 Motivating Example

Let us suppose a real-time embedded system $Sys1$ to be initially implemented by 2 characterized tasks as shown in figure 1. These tasks are feasible because the processor utilization factor U = 0.7 $\leq$ 1. These tasks should meet all required deadlines defined in user requirements and we have $Feasibility(Current_{Sys1}(t))$ $\equiv True$.

	Period / Activation	Calcul	Priority	Utilisation / Response
e1	7	3		17
e2	11	4		33
T1	20	6	3	0.3
T2	10	4	2	0.4
Polling Server	8	2	1	0.25

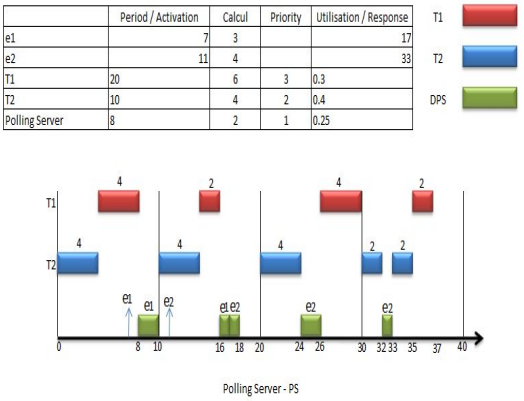

Figure 1: The simulation with only Polling Server

We suppose that a reconfiguration scenario is applied at t1 and t2 time units with the arrival of 2 new aperiodic tasks e_1 at t1 = 7 and e_2 at t2 = 11 time units. Therefore the system is feasible by applying the polling server to schedule the system but the response time is equal to 17 and 33 for both e_1 and e_2 respectively. Now by applying our new hybrid approach, the response time of the second arrival aperiodic task is decreased from 33 to 25 time units as we observe in figure 2.

	Period / Activation	Calcul	Priority	Utilisation / Response
e1	7	3		17
e2	11	4		25
T1	20	6	3	0.3
T2	10	4	2	0.4
Polling Server	8	2	1	0.25
Background Server	*	*	10	*

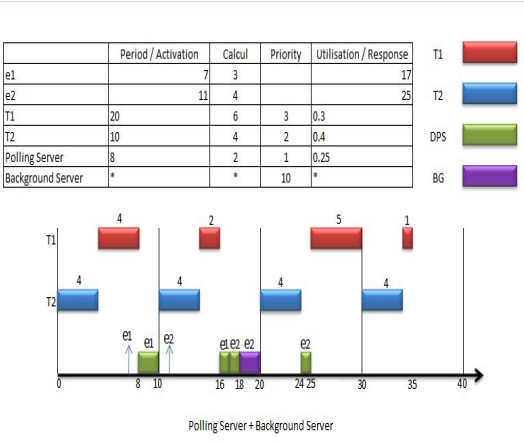

Figure 2: The simulation with Polling Server and Background server

4.3 Formalization

By considering real-time operating system (OS) tasks scheduling, let $n = n_1 + n_2$ be the number of a mixed workload of periodic and aperiodic tasks in $Current_\Gamma(t)$. The reconfiguration of the system $Current_\Gamma(t)$ means the modification of its implementation that will be as follows at t time units:

$$Current_\Gamma(t) = \xi_{new} \cup \xi_{old}$$

Where ξ_{old} is a subset of n_1 old periodic tasks which are periodic and not affected by the reconfiguration scenario (e.g. they implement the system before the time t), and ξ_{new} is a subset of n_2 new aperiodic tasks in the system. We assume that an updated task is considered as a new one at t time units. By considering a feasible System Sys before the application of the reconfiguration scenario, each task of ξ_{old} is feasible, e.g. the execution of each instance is finished before the corresponding deadline

5 EXPERIMENTAL ANALYSIS AND DISCUSSION

In this section, in order to check the suggested configurations of tasks allowing the system's feasibility and the response time minimization, we simulate the agent's behavior on several test sets in order to rate the performance of the polling server and the background server in our hybrid scenario.

5.1 Simulation

We have conducted several test sets in order to rate the performance of the polling server and the background server in our hybrid scenario. We have set up a real-time reconfiguration tool named RT-Reconfiguration that allows us to randomly generate task sets, schedule them according to the proposed hybrid method, and displays the schedules for visual control. Our test rows have been on each 1000 randomly generated task sets, while the number of tasks is significantly higher. We have scheduled task sets with the polling server and the proposed hybrid method.

5.2 Discussion

In each of these examples, many aperiodic requests occur at any moment of the time. The response time performance of only polling service or only background service for the aperiodic requests is poor. Since background service occurs when the resource is idle, with the polling server, the response time performance for the aperiodic requests is better than both single background service and single polling service for all requests. For these examples, a polling server is created with an execution time of 1 time unit and a period of 5 time units. Also note that since any aperiodic request only needs half of the polling server's capacity, the remaining half is discarded because no other aperiodic tasks are pending. Thus, these examples demonstrate how polling and background can provide an improvement in aperiodic response time performance over background service or polling one and are always able to provide immediate service for aperiodic requests. Finally, for both the polling server and the background server in our hybrid scenario approach performs best and yield improved average response times for aperiodic requests.

6 CONCLUSION AND FUTURE WORKS

In this paper, we propose a new theory for the minimization of the response time of aperiodic real-time tasks with the polling server and the background server that can be applied to uniprocessor systems and proved it correct. We showed that this theory is capable to reconfigure the whole system. Previous work in this area has been described, several and best solution has been suggested. This hybrid solution is primarily intended to reduce the processor demand and the response time of each task set independent of the number of tasks in a uniprocessor system. A tool is developed and tested to support all these services. As future work, we are planning to extend our study to the case of distributed systems and, we plan also to apply this contribution to other complex reconfigurable systems that we have chosen to not cover in this paper.

References

[1] Dertouzos. M. L., (1974). Control robotics: The procedural control of physical processes. Information Processing.

[2] Layland J. and Liu C., (1973). Scheduling algorithms for multi-programming in a hard-real-time environment, in Journal of the ACM, 20(1):46-61.

[3] Lehoczky, J. P., L. Sha, and J. K. Strosnider. 1987. Enhanced Aperiodic Responsiveness in Hard-Real-Time Environments. Proc. IEEE Real-Time Systems Symposium, San Jose, CA, pp. 261-270.

[4] Guillem B., (1998). Specification and Analysis of Weakly Hard Real-Time Systems. PhD thesis, Departament de Cincies Matematiques and Informatica. Universitat de les Illes Balears. Spain. http://www.cs.york.ac.uk/-bernat.

[5] Burns A. and Guillem B., (1999). New results on fixed priority aperiodic servers. In 20th IEEE Real-Time Systems Symposium, RTSS, pages 6878, Phoenix. USA.

[6] Sprunt B., Sha L., and Lehoczky J, (1989). Aperiodic task scheduling for hard-real-time systems. Real-Time Systems, 1(1):2760.

[7] Strosnider J. K., Lehoczky J. P., and Sha L., (1995). The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. IEEE Transactions on Computers, 44(1):7391.

[8] Thuel S. R. and Lehoczky J. P., (1994). Algorithms for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing. In Real-Time Systems Symposium, pages 2233, San Juan, Puerto Rico.