Real-time Reconfigurable Scheduling of Aperiodic OS Tasks on Multiprocessor Systems

Hamza Gharsellaoui[1,2,3], Atef Gharbi[2] and Samir Ben Ahmed[2,4]
[1]*Higher School of Technology and Computer Science (ESTI), Carthage University, Tunis, Tunisia*
[2]*National Institute of Applied Sciences and Technology (INSAT), Carthage University, Tunis, Tunisia*
[3]*Al-Jouf College of Technology, Technical and Vocational Training Corporation, Al-Jouf, K.S.A.*
[4]*Faculty of Mathematical, Physical and Natural Sciences of Tunis (FST), Tunis El Manar University, Tunis, Tunisia*

Keywords: Preemptive Scheduling, Real-time Reconfiguration, Aperiodic OS Tasks, Multiprocessor Embedded Systems.

Abstract: The paper deals with the real-time scheduling of aperiodic OS tasks that composed multiprocessor embedded systems which can change their behaviors at run-time for each reconfiguration scenario. A reconfiguration scenario is assumed to be any run-time automatic addition, removal, or also update of OS tasks according to external events or also user requirements. We propose a new approach to check the system's feasibility of the tasks while minimizing their response times. An agent-based architecture is proposed to provide run-time technical solutions for users in order to reach again the system's feasibility after any reconfiguration scenario for the whole multiprocessor embedded systems. We discuss the paper's contribution by analyzing the experimental results that we did on a running example.

1 INTRODUCTION

This paper deals with the problem of hard scheduling of both periodic, sporadic and aperiodic tasks on multiprocessor real-time embedded systems in a critical real-time environment. In this work, we introduce an efficient scheduling algorithm to optimize the response time of the multiprocessors embedded system at run-time while ensuring that all periodic tasks meet their deadlines and to accept as many sporadic tasks, which can be guaranteed to meet their deadlines after a reconfiguration scenario ψ_h (h $\in$ 1..M) was applied. This efficient algorithm results in the dynamic scheduling solutions. These solutions are presented by a proposed intelligent agent-based architecture where a software agent is used to evaluate the response time, to calculate the processor utilization factor and also to verify the satisfaction of real-time deadlines. The agent dynamically provides technical solutions for users where the system becomes unfeasible (e.g deadlines are violated).
The organization of this paper is as follows. Section 2 presents the reconfiguration of tasks in the case of a multiprocessor embedded systems and presents our new contribution and our proposed algorithm for optimal scheduling theory. Section 3 discusses experimental results of the proposed approaches research. Section 4 summarizes the main results and presents the conclusion of the proposed approaches.

2 RECONFIGURATION OF MULTIPROCESSOR REAL-TIME EMBEDDED SYSTEMS

Multiprocessor architectures provide a rich computing environment from which a wide range of problem domains, including real-time applications can benefit. Efficient and effective scheduling techniques have been made in the literature (Gharbi et al., 2010). The major scheduling problem which has been more addressed is that of assigning a set of tasks to different processors in the system, in order to minimize the total response time of the total task set.
Indeed, scheduling algorithms for multiprocessor architectures, including those for real-time applications can be divided into two main categories of static and dynamic scheduling. In static scheduling, the allocation of tasks to the processors is determined off-line prior to the start of task execution. In contrast, dynamic scheduling algorithms perform assigning tasks and processors allocation on-line in order to use more

Gharsellaoui H., Gharbi A. and Ben Ahmed S..
Real-time Reconfigurable Scheduling of Aperiodic OS Tasks on Multiprocessor Systems.
DOI: 10.5220/0004988100430048
In *Proceedings of the 9th International Conference on Software Paradigm Trends* (ICSOFT-PT-2014), pages 43-48
ISBN: 978-989-758-037-6

comprehensive and up-to-date knowledge of the tasks and the environment (architecture).
In recent real-time systems also, computation model has become more and more complex and embedded systems must operate in dynamic environments where human activities occur at any moment, then some tasks, such as emergency task, external event task, human interaction task (add, removal, update), etc., arrive aperiodically and for this reason a reconfiguration scenario ψ_h must be done at run-time to adapt the whole system to its environment and to handle all the user requirements.
The goal of our original approach dealing with the reconfiguration and scheduling of real-time systems is to construct systems that are guaranteed to meet all hard deadlines and that minimize the response time for all soft deadlines (Khalgui, 2010). This a challenge that has frequently not been met to our knowledge and that we aim to meet it in this work. Indeed, to obtain this goal, this system should be changed and automatically adapted to its environment on the occurrence of random disturbances such as hardware-software faults. A random disturbance is defined in this work as any random internal or external event allowing additions, removals or updates of tasks at run-time to adapt the system's behavior. Nevertheless, when an automatic reconfiguration scenario ψ_h is applied, the deadlines of new and old tasks can be violated. We mean by reconfiguration scenario ψ_h in this work, the removal, update or addition of new tasks when they arrive at run-time without prior knowledge in order to save the whole system when random disturbances happen at run-time. We define an agent-based architecture that checks the system's evolution and defines useful solutions for users when deadlines are not satisfied after any reconfiguration scenario ψ_h.
Two cases of suggestions are possible to be provided by our intelligent agent: modification of worst case execution times of tasks and the migration of some tasks from the corresponding processors to others that belongs to the inclusion set. We need by inclusion set in our work, the set of processors in which the tasks can be scheduled after any reconfiguration scenario ψ_h when a migration request has done and in this case all the relevant state information of that migration is transferred to the new processor. Otherwise, it is called exclusion set. The users should choose one of these solutions to re-obtain the system's feasibility and to minimize the response time of the soft tasks. We developed a tool RT-Reconfiguration and tested it in order to support the agent's services.
As well as we know, our approach is among the first and best solutions which consists in assigning the periodic tasks to the various processors for each reconfiguration scenario ψ_h is that we propose in this original work. It implies a large number of advantages, in particular to avoid the complexity of multiprocessor scheduling systems, and our proposed intelligent agent try to achieve this objective by focusing on evenly balancing the load among the processors and on reducing response times of the total task set.

2.1 Approach Description

To explain our approach well, we assume that there are K identical processors numbered from 1 to K, and m real-time tasks numbered from 1 to m that composed a feasible subset of tasks entitled ξ_{old} and need to be scheduled. At time t and before the application of the reconfiguration scenario ψ_h, each one of the tasks of ξ_{old} is feasible, e.g. the execution of each instance in each processor is finished before the corresponding deadline and the tasks are not assumed to be arranged in any specific order.
Each processor p assigns a set of periodic tasks $TS^p = \{\tau_1^p, \tau_2^p, ..., \tau_n^p\}$. This allocation is made with an allowance algorithm at the time of the design. These tasks are independent and can be interrupted at any time. Each task τ_i^p has an execution time (Worst Case Execution Time) C_i^p, one period T_i^p, a deadline D_i^p which is assumed to be less than or equal to its period, e.g. $D_i^p \leq T_i^p$. Each task instance k has to respect its absolute deadline, namely the k^{th} authority of the task τ_i^p, named $\tau_{i,k}^p$ which must be completed before time $D_{i,k}^p$ = (k-1)T_i^p + D_i^p. Each processor p will execute its local tasks by using EDF, it means that the priorities P_i^p of periodic tasks are dynamic and the scheduler guarantees that every instance of every task will run before its deadline. These tasks are handled by a global scheduler (GS), which assigns them to processors by using the state information of the local schedulers. Moreover, under EDF scheduling, a task will fit on a processor as long as the total utilization of all tasks assigned to that processor does not exceed unity (the total utilization factor = 1). Finally, for reasons of simplicity, we assume that all the overheads of context exchange, scheduling of tasks, the preemption of the tasks and the migration cost of the tasks are equal to zero.
We assume now the arrival at run-time of a second subset ξ_{new} which is composed of n real-time tasks at time t_1 ($t_1 = t + \Delta$t). We have a system $Current_{Sys}(t_1)$ composed of $m+n$ tasks. In this case a reconfiguration scenario ψ_h is applied. The reconfiguration of the system Sys^{ψ_h} means the modification of its implementation that will be as follows at time t_1:

$$\xi^{\psi_h} = Current_{Sys}^{\psi_h}(t_1) = \xi_{new}^{\psi_h} \cup \xi_{old}$$

Where ξ_{old} is a subset of old tasks which are not affected by the reconfiguration scenario ψ_h (e.g. they implement the system before the time t_1), and $\xi_{new}^{\psi_h}$ a subset of new tasks in the system. We assume that an updated task is considered as a new one at time t_1. When the reconfiguration scenario ψ_h is applied, two cases exist:

- If tasks of $\xi^{\psi_h} = \xi_{new}^{\psi_h} \cup \xi_{old}$ are feasible, then no reaction should be done by the agent
- Otherwise, the agent should provide different solutions for users in order to re-obtain the system's feasibility. We define the following such services

First Step
the agent tries to modify the execution times of tasks belonging to $\xi_{new}^{\psi_h}$ in order to meet all deadlines that correspond to tasks of ξ^{ψ_h},

Iterative Second Step
the agent tries to consider old tasks of ξ_{old} as new tasks to be introduced in $\xi_{new}^{\psi_h}$. A computation of WCET of these tasks with the new tasks is applied for each reconfiguration scenario ψ_h.

Third Step
the agent tries to migrate some tasks of $\xi^{\psi_h} = \xi_{new}^{\psi_h} \cup \xi_{old}$ from their current processors to be scheduled in other ones which belong to their inclusion group. The inclusion group of each task is formed by a group of processors in which this task can be scheduled. When a task can't be scheduled in a list of processors, this group is called exclusion group. This technique is applied in the migration scenario ψ_h.

2.2 Feasibility Analysis for Tasks

By considering real-time tasks, the schedulability analysis should be done in each processor in an appropriate Hyper-Period. According to (Brocal V., 2011), a hyper-period is defined as $HP = [\zeta, 2 * LCM + \zeta]$, where LCM^{ψ_h} is the well-known Least Common Multiple of periods for each reconfiguration scenario ψ_h of all the tasks that composed the system ξ^{ψ_h} and (ζ^{p,ψ_h}) is the largest task offset of all tasks τ_k^{p,ψ_h} for each reconfiguration scenario ψ_h on each processor p. Let $m + n$ be the number of tasks respectively in ξ_{old} and $\xi_{new}^{\psi_h}$. By assuming an unfeasible system at time t_1, and each processor p will execute its local tasks by using EDF. So, according to (Mok, 1983), the following formula is satisfied:

$\sum_{i=1}^{m+n} \frac{C_i^{\psi_h}}{T_i^{\psi_h}}$ should be $> K$, where K is the number of identical processors.

Our proposed algorithm provides guarantees for both old and new tasks in each processor p if and only if,

$$\sum_{i=1}^{n_1-j} \frac{C_i^{p,\psi_h}}{T_i^{p,\psi_h}} + \sum_{i=n_1-j+1}^{n_1+n_2} \frac{C_i^{p,\psi_h}}{T_i^{p,\psi_h}} \leq 1$$

where
$\sum_{i=1}^{n_1-j} \frac{C_i^{p,\psi_h}}{T_i^{p,\psi_h}}$ denotes the sum of utilization factor of n_1 old tasks in the processor p for each reconfiguration scenario ψ_h and,
$\sum_{i=n_1-j+1}^{n_1+n_2} \frac{C_i^{p,\psi_h}}{T_i^{p,\psi_h}}$ denotes the sum of utilization factor of new arrival n_2 tasks in the processor p for each reconfiguration scenario ψ_h.

2.3 Contribution: Agent-based Real-time Reconfigurable Model

Our main contribution is the efficient schedulability algorithm of multiprocessor real-time tasks implementing reconfigurable multiprocessor embedded systems. By applying a preemptive scheduling, the assumed system is characterized by tasks such that each one is defined by a tuple $(S_i; C_i; D_i; T_i; inclusion; exclusion)$. A system is called asynchronous, if its tasks have offsets and are not simultaneously ready. Note that in synchronous systems, all offsets are zero (all tasks are released at time t = 0).

Formalization
We propose for each reconfiguration scenario ψ_h a new expression for the hyper-period hp in the processor p by $HP = [\zeta, 2 * LCM + \zeta]$. Let $n^{p,\psi_h} = n_1^{p,\psi_h} + n_2^{p,\psi_h}$ be the number of periodic tasks in $Current_{\Gamma}^{p,\psi_h}(t)$ for each reconfiguration scenario ψ_h.

2.3.1 Agent's Principal

Let Γ^{p,ψ_h} be the set of all possible tasks that can implement the system in the processor p for each reconfiguration scenario ψ_h, and let us denote by $Current_{\Gamma}^{p,\psi_h}(t)$ the current set of periodic tasks implementing the system at time t. By considering a feasible system Γ^p before the application of the reconfiguration scenario ψ_h, each one of the tasks of ξ_{old}^p is feasible, e.g. the execution of each instance is finished before the corresponding deadline. In this case, we note that $Feasibility(Current_{\Gamma}^p(t)) \equiv True$. An embedded system can be dynamically reconfigured at run-time by changing its implementation to delete old or to add new real-time tasks. We denote in this research by ξ_{new} a list of new tasks to be added to $Current_{\Gamma}^p(t)$ after a particular reconfiguration scenario ψ_h. In this case, the intelligent agent should check the

system's feasibility that can be affected when tasks violate corresponding deadlines, and should be able to propose technical solutions for users.
Now, we apply at time t a dynamic reconfiguration scenario ψ_h in order to adapt the system's behavior to guarantee the system's feasibility which depends on two major goals of the reconfiguration: Consequently, the task τ_k^{p,ψ_h} can violate also its relative (corresponding) deadline and all the system $Current_{\Gamma}^{p,\psi_h}(t)$ will be unfeasible at time t. In this case the following formula is satisfied for each reconfiguration scenario ψ_h:

$$\sum_{i=1}^{n^{p,\psi_h}} \frac{C_i^{p,\psi_h}}{min(T_i^{p,\psi_h},D_i^{p,\psi_h})} > 1$$

- The first major goal to control the problem's complexity is to minimize the response time of tasks of $Current_{\Gamma}^{p,\psi_h}(t) = \xi_{new}^{p,\psi_h} \cup \xi_{old}^{p}$, then the agent will not modify the ξ_{old} tasks and should provide different solutions for users by reconfiguring only ξ_{new}^{p,ψ_h} which is composed by n_2^{p,ψ_h} tasks in order to satisfy functional requirements,
- The second major goal of obtaining the system's feasibility is to meet deadlines of periodic tasks. Then, the agent should react by updating the global system $Current_{\Gamma}^{p,\psi_h}(t) = \xi_{new}^{p,\psi_h} \cup \xi_{old}^{p}$, which is composed by n_1^{p,ψ_h} and n_2^{p,ψ_h} periodic tasks in order to re-obtain the system's feasibility and provides different solutions for users.

2.3.2 Meeting Deadlines of Periodic Tasks

- **Solution 1: Modification of Worst Case Execution Times**

The agent proceeds as a first solution to modify the Worst Case Execution Times (WCET) of tasks of ξ_{new}^{p,ψ_h} and ξ_{old}^{p} in the processor p for each reconfiguration scenario ψ_h. To obtain a feasible system, the following formula should be satisfied:

$$\sum_{i=1}^{n_1^{p,\psi_h}-j} \frac{C_i^{p,\psi_h}}{min(T_i^{p,\psi_h},D_i^{p,\psi_h})} + \sum_{i=n_1^{p,\psi_h}-j+1}^{n_2^{p,\psi_h}+n_1^{p,\psi_h}} \frac{C_i^{p,\psi_h}+\alpha_i^{p,\psi_h}}{min(T_i^{p,\psi_h},D_i^{p,\psi_h})} = 1$$ based on the (Layland J., 1973) theorem.

$$\longrightarrow \sum_{i=n_1^{p,\psi_h}-j+1}^{n_2^{p,\psi_h}+n_1^{p,\psi_h}} \frac{C_i^{p,\psi_h}+\alpha_i^{p,\psi_h}}{min(T_i^{p,\psi_h},D_i^{p,\psi_h})} = 1 - \sum_{i=1}^{n_1^{p,\psi_h}-j} \frac{C_i^{p,\psi_h}}{min(T_i^{p,\psi_h},D_i^{p,\psi_h})}$$

$$\longrightarrow \sum_{i=n_1^{p,\psi_h}-j+1}^{n_2^{p,\psi_h}+n_1^{p,\psi_h}} \frac{\alpha_i^{p,\psi_h}}{min(T_i^{p,\psi_h},D_i^{p,\psi_h})} = 1 - \sum_{i=1}^{n_1^{p,\psi_h}-j} \frac{C_i^{p,\psi_h}}{min(T_i^{p,\psi_h},D_i^{p,\psi_h})} - \sum_{i=n_1^{p,\psi_h}-j+1}^{n_2^{p,\psi_h}+n_1^{p,\psi_h}} \frac{C_i^{p,\psi_h}}{min(T_i^{p,\psi_h},D_i^{p,\psi_h})}$$

$$\longrightarrow \sum_{i=n_1^{p,\psi_h}-j+1}^{n_2^{p,\psi_h}+n_1^{p,\psi_h}} \frac{\alpha_i^{p,\psi_h}}{min(T_i^{p,\psi_h},D_i^{p,\psi_h})} = 1 - \sum_{i=1}^{n_2^{p,\psi_h}+n_1^{p,\psi_h}} \frac{C_i^{p,\psi_h}}{min(T_i^{p,\psi_h},D_i^{p,\psi_h})}$$

Let γ_j^{p,ψ_h} be the following constant: $\gamma_j^{p,\psi_h} = \alpha_i^{p,\psi_h} = Constant$,

$$\longrightarrow \gamma_j^{p,\psi_h} = \left\lceil \frac{1-\sum_{i=1}^{n_2^{p,\psi_h}+n_1^{p,\psi_h}} \frac{C_i^{p,\psi_h}}{min(T_i^{p,\psi_h},D_i^{p,\psi_h})}}{\sum_{i=n_1^{p,\psi_h}-j+1}^{n_2^{p,\psi_h}+n_1^{p,\psi_h}} \frac{1}{min(T_i^{p,\psi_h},D_i^{p,\psi_h})}} \right\rceil = constant$$

The new WCET of Γ^{p,ψ_h} tasks in the processor p for each reconfiguration scenario ψ_h is therefore deduced from γ_j^{p,ψ_h}.

- **Solution 2: Migration of Tasks**

The agent proceeds now as a second solution to migrate some tasks of ξ_{new}^{p,ψ_h} and ξ_{old}^{p} in the processor p for each reconfiguration scenario ψ_h. Indeed, the agent is responsible for allocating the tasks to the K computing processors in a good way. In order to react to varying run-time conditions, the system feasibility requires homogeneous task migration capabilities.

Run-time task migration can be defined as the relocation of an executing task from its current location, the source processor i, to a new location, the destination processor j, (i $\neq$ j; i,j = 1..K). This allows the OS to e.g. minimize energy savings and response time of the whole system. It also enables processors management by moving tasks away from processors with a high amount of workload or which have their utilization factors $>$ 1. In order to relocate a task, the intelligent agent notifies the task by means of a migration request signal$^{(1)}$. Whenever that signaled task reaches a migration point (MP), it checks if there is a pending migration request or the destination processor j belongs to the exclusion group of the current migrated task for each reconfiguration scenario ψ_h. In such case of these two reasons, all the relevant state information of that migration point is transferred to the intelligent agent$^{(2)}$. Consequently, the intelligent agent will instantiate the same task on a

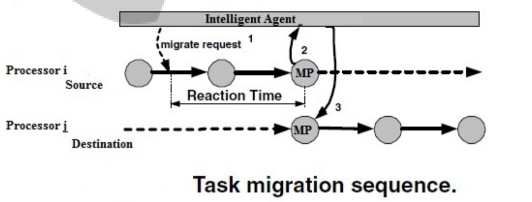

Figure 1: Processor Utilization.

different processor. The new task instantiation will be initialized by using the state information previously captured by the intelligent agent[3]. Finally, the task resumes execution at the corresponding migration point (MP).
One of the main issues in homogeneous (we suppose before that all the processors are identical) task migration is the overhead incurred by checking for a pending migration request during normal execution (i.e. when there is no pending migration request). Especially since a task requires frequent migration points in order to reduce the reaction time. The reaction time (Figure 1) is the time elapsed between selecting a task for migration and the selected task reaching the next migration point. In order to minimize the checking overhead during normal execution, further denoted as migration initiation, we propose a novel technique for the new generation of embedded systems. This novel technique uses the inclusion and exclusion groups information of each task for each reconfiguration scenario ψ_h in order to reduce the area search feasibility of such systems and to minimize the reaction time and consequently the response time will be minimized too.

Final Conclusion.
In conclusion, we can deduce that by arrival of $\xi_{new}^{\psi_h}$ tasks at run-time, the following formula is satisfied for each reconfiguration scenario ψ_h:

$$\sum_{i=1}^{(m+n)^{\psi_h}} \frac{C_i^{\psi_h}}{T_i^{\psi_h}} > K,$$ where K is the number of identical processors.

Then, after the reconfiguration scenario ψ_h was applied at run-time to the whole system by the intelligent agent, our proposed algorithm provides guarantees to both old and new tasks if and only if, we have in each processor p for each reconfiguration scenario ψ_h:

$$\sum_{i=1}^{(m+n)^{(p,\psi_h)}} \frac{C_i^{(p,\psi_h)}}{T_i^{(p,\psi_h)}} \leq 1,$$ in each processor p for each reconfiguration scenario ψ_h, (sufficient condition).

Moreover, we have calculated $R_k^{(p,\psi_h)^{opt}}$ = min($R_{k,1}^{(p,\psi_h)}$ and $R_{k,2}^{(p,\psi_h)}$); so we obtain also:
$\sum_{i=1}^{(m+n)^{(p,\psi_h)}} \frac{C_i^{(p,\psi_h)}}{T_i^{(p,\psi_h)}} < 1$, in each processor p for each reconfiguration scenario
ψ_h with $1 \leq$ p $\leq$ K, $1 \leq$ h $\leq$ M. We can observe that our proposed approach provides an efficient or near-optimal global scheduling algorithm which schedules tasks according to EDF in each processor p for each reconfiguration scenario ψ_h. All tasks meet their deadlines after a reconfiguration scenario ψ_h was applied at run-time. We can also observe, that our proposed algorithm selects tasks to migrate from one processor source i to another processor destination j in an optimal way such that overall utilization of task set is minimum. Parameters of tasks i.e., period, deadline and worst case execution time, are generated randomly. We have illustrated that our proposed algorithm outperforms other scheduling multiprocessor algorithms and a number of scheduling events are much lower than appearing in others.

3 EXPERIMENTATION RESULTS

In this section, we analyze the performance of our proposed approach for both periodic synchronous and asynchronous tasks. The simulation runs on our tool RT-Reconfiguration and proven by the real-time simulator Cheddar (Singhoff L.M.F., 2004) with a task set composed of old tasks (ξ_{old}^{p,ψ_h}) and new tasks (ξ_{new}^{p,ψ_h}) in the processor p for each reconfiguration scenario ψ_h. We illustrate this experimentation with a simplified example. The task set considered for this example is given in table 1 and it is composed now of 10 tasks. The sum of utilization of all tasks is given in table 1 and is equal to 426.1%. In table 1, the first column represents the task identifier, the second column represents the worst case execution time (WCET), the third column represents the period and the fourth column represents the deadline of each task which is less or equal to the period in this example of real time tasks.

Table 1: Task Parameters.

Task	C_i	T_i	D_i
τ_1	2	9	7
τ_2	3	21	20
τ_3	2	9	9
τ_4	2	13	10
τ_5	3	15	9
τ_6	14	21	19
τ_7	10	24	16
τ_8	8	18	18
τ_9	13	16	17
τ_{10}	5	11	12

We have 3 identical processors in our system to schedule these tasks. In this case, we assume that each task's deadline is less or equal to its period. The worst case execution times, deadlines, and the time periods of all tasks are generated randomly.
In this experiment, our task set example is initially implemented by 5 characterized old tasks ($\xi_{old} = \{\tau_1; \tau_2; \tau_3; \tau_4; \tau_5\}$). These tasks are feasible

because the processor utilization factor U = 1.19 $\leq$ 3. These tasks should meet all required deadlines defined in user requirements and we have $Feasibility(Current_{\xi_{old}}(t)) \equiv True$.
Firstly, tasks are partitioned; task τ_1 is executed on first processor, τ_2 and τ_3 are executed on processor 2 while task τ_4 and τ_5 are executed on processor 3. We have three sets of local tasks. As there is only one task on first processor then task τ_1 utilization factor is the same as the first processor utilization factor (utilization factor = 0.285 $\leq$ 1) while utilization factors of processor 2 and processor 3 are calculated as follows:

$$U^2 = \sum_{i=1}^{(2)^2} \frac{C_i^2}{T_i^2} = 0.372 < 1,$$

$$U^3 = \sum_{i=1}^{(2)^3} \frac{C_i^3}{T_i^3} = 0.533 < 1,$$

We suppose that a first reconfiguration scenario ψ_1 (h = 1) is applied at time t1 to add 5 new tasks $\xi_{new}^{\psi_1} = \{\tau_6; \tau_7; \tau_8; \tau_9; \tau_{10}\}$. The new processor utilization becomes $U^{\psi_1} = 4.261 > 3$ time units. Therefore the system is unfeasible.
$Feasibility(Current_{\xi}^{\psi_1}(t1)) \equiv False$.
Indeed, if the number of tasks increases, then the overload of the system increases too.
We apply our contribution to this running example and we could observe that the recalculation points of the utilization factor, when parameters of new tasks are modified, decreases and becomes less or equal to 1 and we can deduce that the system is now feasible.
Moreover, if the number of solutions presented by the intelligent agent to the user increases, then chances of executing more new added tasks increase and the performance of the real-time scheduling is more efficient. This is due to the fact that the reconfiguration issues are increased, the user selects the best solution which gives the minimum utilization factor of the system, ameliorates the response time and hence the chances of executing more new tasks are increased as well.
These results were suggested by the tool RT-Reconfiguration and give a feasible system which is proven also by Cheddar (Singhoff L.M.F., 2004).

4 CONCLUSION

In this paper, we study the functional feasibility in multiprocessor systems with a shared memory. We proposed an efficient scheduling algorithm to optimize response time while ensuring that all periodic tasks meet their deadlines with partitioning scheduling and to accept as many tasks as possible. Furthermore, with this efficient solution, these tasks can be guaranteed to meet their deadlines after a reconfiguration scenario ψ_h and were applied by an efficient EDF based scheduling algorithm on multiprocessor system. We assume that our proposed algorithm uses an independent task sets in order to minimize the interaction between tasks to limit the number of messages transmitted and overloads conditions. Finally, we verify also, the correctness of the whole system with minimizations of response times.

REFERENCES

Brocal V., BalbastreP., B. R. R. L. (2011). *Task period selection to minimize hyperperiod, Emerging Technologies & Factory Automation (ETFA), IEEE conference on, pp.1-4, 2011.* doi: 10.1109/ETFA.2011.6059178, Toulouse, France, 16th edition.

Gharbi, A., Khalgui, M., and Ahmed, S. B. (2010). Inter-agent communication protocol for distributed reconfigurable control software components. *Ant 2010.*

Khalgui, M. (2010). Nces-based modelling and ctl-based verification of reconfigurable embedded control systems. *Computers in Industry, Vol.61, N.3.*

Layland J., Liu, C. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM, 20(1):46-61.*

Mok, A. K. (1983). Fundamental design problems of distributed systems for the hard real time environment. *PhD Dissertation, MIT, May.*

Singhoff L.M.F., Legrand, J. (2004). Cheddar : a flexible real time scheduling framework. *ACM SIGAda Ada Letters, volume 24, number 4, pages 1-8. Edited by ACM Press, ISSN:1094-3641.*