# Real-Time Scheduling of Reconfigurable Battery-Powered Multi-Core Platforms

Aymen Gammoudi, Adel Benzina, Mohamed Khalgui, Daniel Chillet

# Real-Time Scheduling of Reconfigurable Battery-Powered Multi-Core Platforms

Aymen Gammoudi
LISI Laboratory, INSAT
Tunisia Polytechnic School
University of Carthage, Tunisia
IRISA Laboratory, ENSSAT
University of Rennes1, France
aymen.gammoudi@irisa.fr

Adel Benzina
LISI Laboratory, INSAT
Tunisia Polytechnic School
University of Carthage, Tunisia
adel.benzina@isd.rnu.tn

Mohamed Khalgui
LISI Laboratory, INSAT
University of Carthage, Tunisia
SystemsControl Laboratory
University of Xidian, China
khalgui.mohamed@gmail.com

Daniel Chillet
IRISA Laboratory, ENSSAT
University of Rennes1, France
daniel.chillet@irisa.fr

***Abstract*—This paper deals with the real-time scheduling in a reconfigurable multi-core platform powered by a rechargeable battery. A reconfiguration scenario is defined as an operation that allows the addition-removal-modification of tasks which may result in timing unfeasibility. Such a system may face several scenarios: i) increased power consumption that, in the worst case, may surpass the available energy budget, ii) increased computing demand, which may lead to the violation of real-time constraints, and iii) increased memory demand, potentially exceeding the provided memory capacity. To prevent these problems during the execution, a new scheduling strategy is necessary. The proposal is based on the assignment of tasks to different processor cores to satisfy these constraints simultaneously after any reconfiguration scenario. The effectiveness and performance of the designed approach are evaluated through simulation studies. An intelligent tool named *Reconf-Pack* is developed in our research laboratory to support this new proposed approach and to simulate it over randomly generated tasks.**

***Keywords–Energy-efficient; Embedded multi-core platform; Reconfiguration; Real-time and low-power scheduling; Task simulation.***

## I. INTRODUCTION

Reconfigurable multi-core platforms are used in many application domains, manufacturing process control, telecommunications, robotics, sensor networks, vehicle navigation and consumer electronics. In all of these areas, there is a rapid technological progress, yet, energy concerns are still the bottleneck. When the system is powered by battery, the problem is yet more important and the system must ensure that between two recharges the energy is sufficient to support all the services.

In this context, we focus on reconfigurable real-time multi-core systems when the battery recharges are done periodically. A reconfiguration scenario is defined in this current paper as any internal or external event that leads to the addition or removal of software tasks to adapt the system's behavior [1]. This paper addresses the case of adding tasks as it is critical in real-time systems. The minimization of energy consumption is an important criterion for development of rechargeable reconfigurable real-time systems due to limitations in the capacity of their batteries. In addition, the battery life can be extended by reducing the power consumption [2]. When undergoing a reconfiguration, to reduce the energy consumption, these systems have to be changed and adapted to their environment without any disturbance. Any reconfiguration scenario may increase energy consumption and/or leads to violation of deadline for some software tasks. Dynamic reconfiguration (handled by software autonomous agents [3]) is important in embedded systems, where one does not necessarily have the luxury to stop a running system and apply a reconfiguration manually. For these reasons, we consider here dynamic reconfiguration and assume that the system executes $n$ real-time periodic tasks initially feasible towards real-time scheduling. The tasks run on a multi-core system. In this paper, we assume that: (i) All the tasks are independent, and (ii) The processor cores are homogeneous. A homogeneous multi-core system consists of cores with the same characteristics, i.e; having the same set of possible clock frequencies. We consider that the system battery is recharged periodically with a recharge period *RP*. The general goal of this paper is to ensure that any reconfiguration scenario changing the implementation of the multi-core system does not violate real-time constraints and does not result in fatal energy over consumption or in memory saturation. In such situations, the research studies in [2] and [4] propose solutions based on the modification of periods or WCETs (worst case execution times) of tasks in order to decrease the processor utilization. We propose in a previous paper [5] a dynamic methodology, according to the system and battery state, that proposes quantitative techniques to modify periods, reduce execution times of tasks or remove some of them to ensure real-time feasibility by avoiding memory overflow and by ensuring a rational use of remaining energy until the next recharge. These studies are useful but not applicable to distributed real-time computing architectures. Therefore, we extend the technique presented in [5] to multi-core processor. To evaluate the performance of the proposed approach, we present a new simulator *Reconf-Pack* for analyzing a reconfiguration and applying the proposed strategy for real-time multi-core systems. It is based upon

an other tool *Task-Generator* which generates random set of tasks. The organization of this paper is as follows. Section II presents the state of the art of reconfigurable multi-core systems, low power consumption and real-time scheduling. We formalize the problem in Section III. We present in Section IV the reconfiguration of tasks with the proposed run-time strategy. The fifth Section shows the *Reconf-Pack* architecture and its internal modules. Section VI explains the case study and evaluates approach. Finally, we conclude and present the future work in Section VII.

## II. State of the Art

The goal of real-time scheduling under energy constraints is to ensure the execution of all the tasks at run-time without missing the deadlines while ensuring a rational use of remaining energy until the next recharge. Several studies have been performed in this context such as the research works reported in [5], [4], [6], [7], [8], [9], [10] and [11] to solve the scheduling problem in real-time systems under energy constraints. The real-time reconfigurable embedded systems are required to produce more flexible and adaptable solutions.

### A. Low-Power Scheduling of Real-Time Systems

Real-time scheduling has been extensively studied in the last three decades [12]. The related studies propose several feasibility conditions for the dimensioning of real-time systems. These conditions are defined to enable a designer to guarantee that time constraints associated with an application are always met for all possible configurations. In the literature, there are many scheduling algorithms for multi-core and mono-core platforms. Two main classical scheduling policies are generally used in real-time embedded systems: RM (Rate Monotonic) and EDF (Earliest Deadline First) [13]. EDF is an optimal scheduling algorithm on preemptive processors. For a given set of *n* synchronous tasks, $\tau_1$, $\tau_2$,..., $\tau_n$ with time periods $T_1$, $T_2$, ..., $T_n$, and computation times (WCETs) $C_1$, $C_2$, ..., $C_n$ such that it is assumed that $T_i = D_i$ (period equals to deadline) for each task, the deadline driven schedule algorithm is feasible if and only if $U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$ [13]. RM is an on-line preemptive static-priority scheduling strategy for periodic and independent tasks. Higher priorities are assigned statically to tasks with higher frequencies (short periods). For a given set of *n* tasks, $\tau_1$, $\tau_2$, ..., $\tau_n$ with time periods $T_1$, $T_2$, ..., $T_n$, and computation times of $C_1$, $C_2$, ..., $C_n$, the Rate Monotonic scheduling algorithm is feasible if $U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1)$. In the current work, to ensure the task feasibility and the availability of energy after each reconfiguration scenario, we focus on adapting task parameters $T_i$ or $C_i$ to fulfill the feasibility condition. We propose to apply the dynamic policy EDF when the performance of the system is high, otherwise the static policy RM for systems with limited characteristics. We use as a common notation for this real-time feasibility condition : $U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq \alpha_{policy}$, where $\alpha_{policy}$ = 1 for EDF scheduling policy and $\alpha_{policy}$ = $n(2^{\frac{1}{n}} - 1)$ for RM scheduling policy.
In the literature, these two scheduling policies are used to address the problem of low-power scheduling. Power-reduction techniques can be classified into two categories: Static [14] and dynamic [7]. Thanks to the study in [4], the function that represents the power consumption $P$ of a DVS-enabled processor is at least quadratic with respect to the processor utilization, i.e.,

$$P \ \alpha \ U^2 \tag{1}$$

So, $P = k.U^2$. If the processor utilization is minimized, then the power consumption is automatically minimized.

### B. Reconfigurable Real-Time Systems

Several interesting academic and industrial works focused on reconfigurable systems where automatic reconfigurations are applied by intelligent agents [3]. Wang et al. present in [2] and [4] a simple run-time strategy to ensure that the system (mono-core in a processor) runs correctly after any reconfiguration scenario. They propose to modify the period of tasks $T_i$ by assigning a single value to all the tasks which is not reasonable in practice. Another solution proposed is to reduce WCETs $C_i$ of all the tasks. These solutions are interesting, but the main disadvantage is that it is not acceptable for a real-time system to change the period of tasks more than a certain limit according to user requirements. To improve these solutions and implement more realistic values, we propose in a previous paper [5] a new strategy based on the classification of tasks. This proposal is based on the definition of packs of tasks and the management of their parameters. We propose to group the tasks that have similar periods in packs by assigning a unique period to all the tasks related to a pack. For each reconfiguration scenario, specific modifications are performed on the parameters of the packs and their related tasks in order to satisfy the memory, real-time and energy constraints. We compare this strategy to the research works in [4] and [2] and show that the cost in terms of delaying tasks is significantly improved. Although these rich and useful contributions provide interesting results, no one is reported to address the problem of multi-core processor platforms.
The current paper addresses this issue and proposes an extended strategy based on the pack strategy presented in [5]. This strategy provides a new algorithm to initially assign tasks to the different cores of a system. This strategy minimizes the cost of delaying tasks for potential future reconfigurations. Then, after each reconfiguration scenario, a second algorithm is proposed to ensure that the new task assignment is feasible by providing different solutions for each core to remedy each problem. This idea is formalized in section III.

### C. Real-Time Scheduling of Multi-Core Architectures

There is a lot of successful researches addressing the problem of scheduling of real-time multi-core systems. Multi-core platforms provide a rich computing environment from which a wide range of problem domains, including real-time applications, can benefit. Efficient scheduling techniques have been made in the literature [15]. The major scheduling problem which has been addressed is that of assigning a set of tasks to different cores in the system, in order to minimize the total response time of the total set of tasks [16]. To conserve energy in multi-core platforms, Ching-Chi et al. in [17] and [18] propose task scheduling algorithms that leverage percore DVFS and achieve a balance between performance and energy consumption. They consider two task execution modes: The *batch* mode, which runs jobs in batches; and the *online* mode in which jobs with different time constraints, arrival times, and

computation workloads co-exist in the system. These studies are interesting, but the authors did not consider the reconfiguration problem, neither the memory overflow problem. In addition, the authors of the paper [17] and [18] have not studied the rechargeable systems with a well-defined period of recharge. The goal of the current approach that deals with the reconfiguration and scheduling of real-time systems is to construct software tasks that meet their hard deadlines and that guarantee a possible execution until the next recharge.

## III. Problem Formalization for Multi-Core Systems

In this section, we describe the platform model that consists of a reconfigurable real-time system powered by a battery and running on a multi-core processor. We assume that the system battery is recharged periodically. It is assumed also that the system has a unique memory device and a same processor speed for all processor cores. In this context, we consider that this system can be formalized by $Sys = \{H_w, S_w\}$ such that $H_w$ is the hardware platform and $S_w$ is the software one. $H_w$ contains a set $\pi$ of *p* cores $\pi = \{Core_1, Core_2, ..., Core_p\}$. The multi-core platform is supplied by a battery with a limited capacity. The software platform $S_w$ contains initially a set $\psi$ of $n$ software tasks $\psi = \{\tau_1, \tau_1, ..., \tau_n\}$ such that each one is assigned to a given core according to a technique detailed afterwards.

### A. Task Model

Let us suppose that each task $\tau_i$ is characterized by seven parameters $\tau_i$=$\{R_i, C_i, T_i, T_imax, D_i, I_i, MF_i\}$. As described in [13], each task $\tau_i$ of $\psi$ is defined by: (i) A release time $R_i$; (ii) A worst case execution time (WCET) $C_i$; (iii) A period $T_i$; (iv) A maximum period $T_imax$ which is the maximum period $T_i$ that should not be exceeded according to the system specification; (v) A deadline $D_i$; (vi) An importance factor $I_i$ which is an integer variable between 0 and 15. If a task has a high value $I_i$, then the task is less important, else it is paramount. When the embedded system has a low energy, then it is possible to remove some tasks according to their importance factors. The tasks that have $I_i = 0$ are considered critical real-time tasks that can not admit any change in their parameters; (vii) A memory footprint $MF_i$ which is the memory space used by task $\tau_i$. In this paper, we assume that $T_i = D_i$ for each task. After each reconfiguration scenario, it is necessary to check the feasibility of the real-time scheduling in each processor core $Core_j$ by verifying the equation $\sum_{i=1}^{m_j} \frac{C_i}{T_i} \leq \alpha_{policy}$, where $m_j$ is the number of tasks running on the $Core_j$ ($j \in [1..p]$).

### B. Energy Model

We consider that the embedded multi-core system in this paper is periodically fully recharged. The energy model is characterized by: (i) A quantity of energy available at full recharge $E_{max}$, (ii) An energy available at time $t$ : $\Delta E(t)$ and (iii) A recharge period $RP$. As defined in Section II-A, the power consumption $P$ is proportional to the processor utilization $U$: $P = k.U^2$. The total power consumption is the sum of the consumption of all processor cores. The power consumption is then calculated by:

$$P = k.U^2 = k.\sum_{j=1}^{p}(\sum_{i=1}^{m_j} \frac{C_{i,j}}{T_{i,j}})^2 \quad (2)$$

where $C_{i,j}$ and $T_{i,j}$ are respectively the worst case execution time and the period of task $\tau_i$ running on $Core_j$.
We consider in this paper that $k = 1$. To ensure that the system runs correctly until the next recharge, it is necessary that at a time *t*:

$$P(t).\Delta t \leq \Delta E(t) \quad (3)$$

where $P(t)$ is the power consumption at $t$ and $\Delta t$ the time remaining until the next recharge. That means the average power consumption $P(t) \leq \frac{\Delta E(t)}{\Delta t}$. We define $P_{limit}(t) = \frac{\Delta E(t)}{\Delta t}$. Therefore, after each reconfiguration scenario, we have to satisfy the energy constraint $P(t) \leq P_{limit}(t)$: This is the energy constraint.

### C. Memory Model

In the embedded multi-core system, we suppose that the memory model is characterized by a memory size $MS$. Each task occupies at run-time $MF_i$ amount of memory. After each reconfiguration scenario, we must satisfy that $\sum_{i=1}^{n} MF_i < MS$: This is the memory constraint.

### D. Reconfiguration Problem

We suppose that *Sys* is initially composed of $n$ tasks and $p$ cores at $t0$: $\pi(t0) = \{Core_1, Core_2, ..., Core_p\}$ and $\psi(t0) = \{\tau_1, \tau_1, ..., \tau_n\}$. We assume that $Sys(t0)$ is feasible. A system is feasible if and only if it satisfies the three constraints (real-time, energy and memory constraints). We assume in the following that the system $Sys$ is dynamically reconfigured at run-time at $t1$ such that its new implementation of tasks is $\psi(t1) = \{\tau_1, \tau_1, ..., \tau_n, \tau_{n+1}, ..., \tau_m\}$. The subset $\{\tau_{n+1}, ..., \tau_m\}$ is added to the initial implementation $\{\tau_1, \tau_2, ..., \tau_n\}$. The added tasks must be assigned to the different cores according to a specific strategy. To ensure that the system runs correctly after this reconfiguration scenario, it is necessary to check whether the new configuration satisfies these three constraints:

1) Real-time scheduling feasibility constraint (RTConstraint): For each $Core_j$, *j*=$\{1...p\}$, $Sys$ must verify:
$$U_{Core_j} = \sum_{i=1}^{m_j} \frac{C_{i,j}}{T_{i,j}} \leq \alpha_{policy}$$
2) Energy constraint (EgConstraint), $Sys$ must verify:
$$P(t1) \leq P_{limit}(t1)$$
3) Memory constraint (MemConstraint), $Sys$ must verify:
$$\sum_{i=1}^{m} MF_i < MS$$

After each reconfiguration scenario, one or more of these constraints can be violated. In such case, we have to find the suitable solution to each problem.

## IV. Feasible Reconfiguration of Multi-Core Architectures

In this section, two algorithms are presented, the first is to initially assign tasks to different cores after a cold start. The second is executed after each reconfiguration in order to verify all constraints and apply the suitable solution.

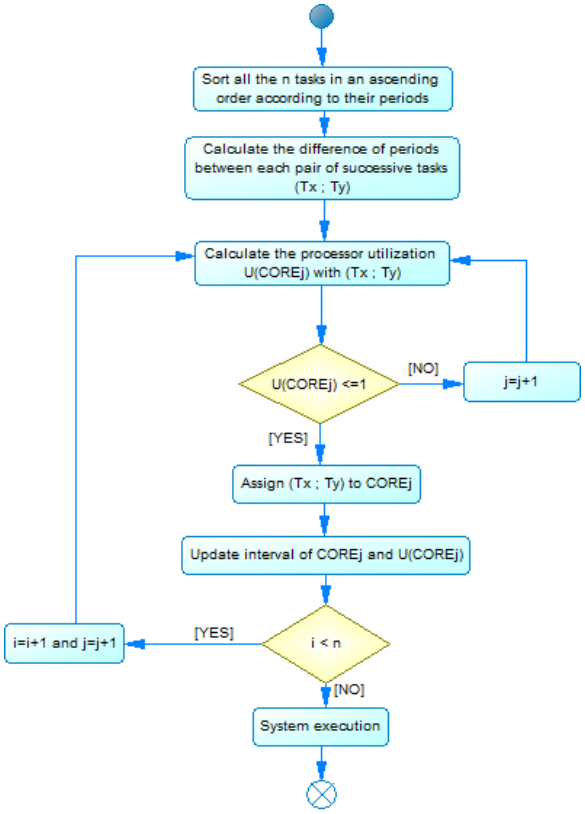


Figure 1. Initial assignment of tasks to cores.

### A. Initial Assignment of Tasks to Cores

The initial assignment of tasks to different cores is arranged to facilitate a future potential reconfiguration while applying the pack strategy.

Before starting the execution of the application, we apply the proposed assignment strategy as follows: We sort all the $n$ tasks in an ascending order according to their periods. Then we calculate the difference of periods between each pair of successive tasks. We start to affect the pair of tasks with the smallest difference to $Core_1$, and the next smallest difference to $Core_2$ and so on until all the initial tasks are assigned. Note that it is necessary to check the processor utilization, if it is higher than 1, then we change this assignment to the next $Core_j$ ($j \in [1..p]$). When arriving to the last $Core_p$, we return cyclically to $Core_1$. Once we finish the assignment of all tasks, the initial system startup is activated. This assignment strategy ensures that each core gets pairs of tasks with "close" periods and the cost of any future reconfiguration will be minimized. In order to apply the proposed strategy presented in IV-B, we suppose that the initial assignment is always feasible. Figure 1 explains how we initially assign the initial tasks to different cores.

### B. New Solutions for Feasible Low-Power Real-Time Reconfigurable Multi-Core Platforms

The proposed strategy offers five solutions detailed and justified in [5]. If one core $Core_x$ violates a real-time constraint (RT), Then the proposed strategy executes one of these solutions (SolutionA and SolutionB) in order to ensure that the system satisfies this constraint. The other two solutions (SolutionC and SolutionD) are used if the energy constraint (Eg) is not satisfied.

*1) Solution A: Modification of Periods under Real-Time Scheduling Constraint:* In order to satisfy the real-time constraint according to a scheduling policy "$\alpha_{policy}$", we propose to extend the period $T_i$ of $\tau_i$ as a multiple of $T^{New}_{RT,Core_x}$:

$$T^{New}_{RT,Core_x} = \left\lceil \frac{\sum_{i\in Pk_1} C_{i,x} + \sum_{i\in Pk_2} \frac{C_{i,x}}{2} + ... + \sum_{i\in Pk_j} \frac{C_{i,x}}{j}}{\alpha_{policy}} \right\rceil \quad (4)$$

In this paper, we detail the demonstration of solution A.

*Proof.* We have $\sum_{i=1}^{m} \frac{C_i}{T_i} \leq \alpha_{policy}$. We assign each task to its pack according to its period $T_i$, Then:

$$\sum_{i\in Pk_1} \frac{C_i}{T} + \sum_{i\in Pk_2} \frac{C_i}{2.T} + ... + \sum_{i\in Pk_j} \frac{C_i}{j.T} \leq \alpha_{policy}$$

So,

$$\frac{1}{T}.\left( \sum_{i\in Pk_1} C_i + \sum_{i\in Pk_2} \frac{C_i}{2} + ... + \sum_{i\in Pk_j} \frac{C_i}{j} \right) \leq \alpha_{policy}$$

Then,

$$T^{New}_{RT,Core_x} = \frac{\sum_{i\in Pk_1} C_i + \sum_{i\in Pk_2} \frac{C_i}{2} + ... + \sum_{i\in Pk_j} \frac{C_i}{j}}{\alpha_{policy}}$$

Since the periods are integer:

$$T^{New}_{RT,Core_x} = \left\lceil \frac{\sum_{i\in Pk_1} C_i + \sum_{i\in Pk_2} \frac{C_i}{2} + ... + \sum_{i\in Pk_j} \frac{C_i}{j}}{\alpha_{policy}} \right\rceil$$

where $T^{New}_{RT,Core_x}$ is the new period affected to the tasks of pack $Pk_1$, $j * T^{New}_{RT,Core_x}$ to tasks of $Pk_j$ in order to satisfy the real-time constraint.

*2) Solution B: Modification of WCETs under Real-Time Scheduling Constraint:* To ensure that the system is feasible after any reconfiguration scenario, we propose to reduce WCETs $C_i$ of tasks as multiples of:

$$C^{New}_{RT,Core_x} = \left\lfloor \frac{\alpha_{policy}}{\sum_{i\in Pk_1} \frac{1}{T_{i,x}} + \sum_{i\in Pk_2} \frac{2}{T_{i,x}} + ... + \sum_{i\in Pk_j} \frac{j}{T_{i,x}}} \right\rfloor \quad (5)$$

We assign $C^{New}_{RT,Core_x}$ to tasks of $Pk_1$, $j * C^{New}_{RT,Core_x}$ to tasks of $Pk_j$. After the modification of the WCETs, the processor utilization of tasks is reduced and can satisfy the real-time constraint.

*3) Solution C: Modification of Periods under Energy Constraint:* If the system risks a fatal increase in energy consumption, then it is necessary that the current power $P(t) = k.U^2$ should be less than the critical power $P_{limit}$. To resolve this problem, we propose to extend the periods of tasks as follows:

$$T^{New}_{Eg} = \left\lceil \frac{\sum_{i\in Pk_1} C_i + \sum_{i\in Pk_2} \frac{C_i}{2} + ... + \sum_{i\in Pk_j} \frac{C_i}{j}}{\sqrt{\frac{P_{limit}(t)}{k}}} \right\rceil \quad (6)$$

where $T^{New}_{Eg}$ is the new period assigned to the tasks of $Pk_1$ to satisfy the energy constraint ($Eg$), $j * T^{New}_{Eg}$ is the period assigned to the tasks of $Pk_j$.

*4) Solution D: Modification of WCETs under Energy Constraint:* The extended WCET $C_i$ of task $\tau_i$ is multiple of $C_{Eg}^{New}$:

$$C_{Eg}^{New} = \left\lfloor \frac{\sqrt{\frac{P_{limit}(t)}{k}}}{\sum_{i \in Pk_1} \frac{1}{T_i} + \sum_{i \in Pk_2} \frac{2}{T_i} + \ldots + \sum_{i \in Pk_j} \frac{j}{T_i}} \right\rfloor \quad (7)$$

We assign $C_{Eg}^{New}$ to the tasks of $Pk_1$, $j * C_{Eg}^{New}$ to the tasks of $Pk_j$. After the modification of the WCETs, the processor utilization of tasks is reduced and can satisfy the energy constraint.

*5) Solution E: Removal of Tasks:* This solution proposes the removal of less important tasks according to an importance factor $I_i$ defined in [5] in order to minimize the energy consumption.
For each solution, a new period $T^{New}$ (a new $C^{New}$) is calculated and assigned to tasks to satisfy their energy or real-time constraints. All the tasks of a core will be assigned this new period (or WCET). The cost of this change is a delay in the period of tasks that we calculate and assume that it is the total cost of the solution to maintain the system feasible and up till the next recharge.

### C. Heuristic Solution for Real-Time and Low-Power Scheduling of Reconfigurable Multi-Core Platforms

In this subsection, we present the operating mode to allow a feasible multi-core system after any reconfiguration scenario. To satisfy the memory, real-time and energy constraints after such a scenario, the system should start by checking the memory availability. If this constraint is satisfied, then the energy and also real-time constraints have to be checked. If one or more constraints are violated, then this program ensures a deterministic choice between different solutions presented in Section IV-B. To understand these steps, Algorithm 1 explains this strategy after a reconfiguration scenario. The complexity of this algorithm is *O(n)*. Algorithm 1 implements the following functions:
**ProcUtiliz(Sol X)**: Is a function that returns the processor utilization value when it runs with solution $X$.
**Execution(Sol X)**: System execution by applying solution $X$.
**Max(Sol X, Sol Y)**: It is a function that returns the maximum value between solution $X$ and solution $Y$.
**Min(Sol X, Sol Y)**: It is a function that returns the minimum value between solution $X$ and solution $Y$.

**Algorithm 1** Deterministic strategy

**Input:** $Sys(t0)$ is feasible $\dotplus$ application of reconfiguration scenario at $t1$
**while** Reconfiguration **do**
  **if** (!MemConstraint) **then**
    $Execution(SolE)$
  **else if** (RTConstraint) AND (EgConstraint) **then**
    Reconfiguration=false
  **else if** (!RTConstraint) AND (EgConstraint) **then**
    **if** (ProcUtiliz(Sol A) < ProcUtiliz(Sol B)) **then**
      $Execution(SolA)$
    **else**
      $Execution(SolB)$
    **end if**
  **else if** (RTConstraint) AND (!EgConstraint) **then**
    **if** (ProcUtiliz(Sol C) < ProcUtiliz(Sol D)) **then**
      $Execution(SolC)$
    **else**
      $Execution(SolD)$
    **end if**
  **else**
    SolPeriod = Max{Sol A,Sol C}
    SolWcet = Min{Sol B,Sol D}
    **if** (ProcUtiliz(SolPeriod) < ProcUtiliz(SolWcet) **then**
      $Execution(SolPeriod)$
    **else**
      $Execution(SolWcet)$
    **end if**
  **end if**
**end while**
**Output:** $Sys$ is feasible after reconfiguration scenario

## V. *Reconf-Pack* Architecture

In this section, we present the global architecture and the operation mode (Figure 2) of the tool named *Reconf-Pack*. In fact, *Reconf-Pack* is a scheduling simulator that compares two scheduling strategies by showing the gain of the best solution. The first strategy is presented in [4] and the second is the proposed strategy of the current paper. To launch *Reconf-Pack*, we initially propose the tasks list and choose the scheduling algorithm (EDF or RM). *Reconf-Pack* takes these values and launches the simulation before showing a comparative histogram in few milliseconds. The comparison is the cost in terms of delaying tasks. So, we can evaluate the gain of our proposed strategy. In order to generalize the performance evaluation of the contribution, another module called *Task-Generator* has been developed and added to *Reconf-Pack* to generate random systems and execute randomly reconfiguration scenarios. *Autorun* is an supplementary module implemented in *Reconf-Pack* to automate the random generation tasks system. We use *Autorun* in order to evaluate the proposed strategies with infinity randomly generated tasks. All task lists are stored in *Excel* files which are already exist in a database.

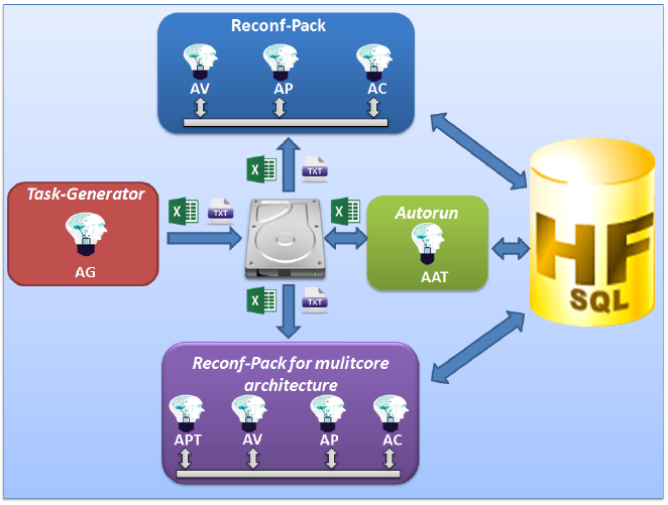


Figure 2. *Reconf-Pack* Architecture.

## VI. Case Study

We present in this section a case study that illustrates the different problems and the developed strategy.

### A. Presentation

Let us suppose that the system has one processor with four cores ($Core_1$, $Core_2$, $Core_3$, $Core_4$). We assume that it supports the tasks described in the first column of Table I. Each

$Core_j$ executes its local tasks by using the EDF scheduling policy ($\alpha_{policy} = 1$). Initially, we sort the tasks according to their periods (Table I, second column).

TABLE I. Tasks assigning steps.

| Initial system tasks | | | Ordered tasks | | |
|---|---|---|---|---|---|
| Task | $C_i$ | $T_i$ | Task | $C_i$ | $T_i$ |
| $\tau_1$ | 5 | 33 | $\tau_5$ | 2 | 15 |
| $\tau_2$ | 6 | 66 | $\tau_8$ | 1 | 20 |
| $\tau_3$ | 1 | 69 | $\tau_6$ | 3 | 26 |
| $\tau_4$ | 2 | 38 | $\tau_1$ | 5 | 33 |
| $\tau_5$ | 2 | 15 | $\tau_9$ | 4 | 35 |
| $\tau_6$ | 3 | 26 | $\tau_4$ | 2 | 38 |
| $\tau_7$ | 3 | 46 | $\tau_7$ | 3 | 46 |
| $\tau_8$ | 1 | 20 | $\tau_2$ | 6 | 66 |
| $\tau_9$ | 4 | 35 | $\tau_3$ | 1 | 69 |
| $\tau_{10}$ | 2 | 74 | $\tau_{10}$ | 2 | 74 |

Then, we calculate the difference of periods between each pair of successive tasks as presented in Table II.

TABLE II. Difference of periods.

| Difference period between | Value |
|---|---|
| $\tau_5$ and $\tau_8$ | 5 |
| $\tau_8$ and $\tau_6$ | 6 |
| $\tau_6$ and $\tau_1$ | 7 |
| $\tau_1$ and $\tau_9$ | 2 |
| $\tau_9$ and $\tau_4$ | 3 |
| $\tau_4$ and $\tau_7$ | 8 |
| $\tau_7$ and $\tau_2$ | 20 |
| $\tau_2$ and $\tau_3$ | 3 |
| $\tau_3$ and $\tau_{10}$ | 5 |

The scheduler assigns the couple of tasks with the smallest difference to $Core_1$. Thereafter, the second pair of tasks will be assigned to the second core $Core_2$...etc. The time interval of the tasks of each core is updated as described in Table III:

TABLE III. Assigning Tasks to Cores.

| Core | Tasks | Time Interval |
|---|---|---|
| $Core_1$ | $\tau_1$ and $\tau_9$ | [33..35] |
| $Core_2$ | $\tau_2$, $\tau_3$ and $\tau_{10}$ | [66..74] |
| $Core_3$ | $\tau_5$, $\tau_8$ and $\tau_6$ | [15..26] |
| $Core_4$ | $\tau_4$ and $\tau_7$ | [38..46] |

We verify the system feasibility condition for the four processor cores (Table IV):

TABLE IV. Processor utilization for each core.

| | $Core_1$ | $Core_2$ | $Core_3$ | $Core_4$ |
|---|---|---|---|---|
| Processor utilization $U$ | 0.266 | 0.132 | 0.947 | 0.454 |

For the first assignment, the feasibility condition is satisfied because the processor utilization $U$ for each core is less than 1. We suppose now that after a certain execution time, a first reconfiguration is performed at a particular time to add two tasks $\tau_{11}$ and $\tau_{12}$ as described in Table V.

TABLE V. Scenario 1: Added tasks.

| Task | $C_i$ | $T_i$ | $T_imax$ | $I_i$ |
|---|---|---|---|---|
| $\tau_{11}$ | 10 | 45 | 100 | 10 |
| $\tau_{12}$ | 2 | 34 | 100 | 10 |

According to the period of tasks, we assign these tasks to the corresponding core: $\tau_{11}$ must be affected to $Core_4$ (45 $\in$ [38..46]) and $\tau_{12}$ to $Core_1$ (34 $\in$ [33..35]). Due to this reconfiguration, we must verify if the system satisfies the feasibility condition. We compute $U$ for $Core_4$ and $Core_1$. $U_{Core_1} = \sum_{i=1}^{3} \frac{C_{i,1}}{T_{i,1}} = 0.325 \leq 1$ and $U_{Core_4} = \sum_{i=1}^{3} \frac{C_{i,4}}{T_{i,4}} = 0.676 \leq 1$. The feasibility condition is then satisfied.

We suppose that at this time *t1*, $P_{limit}(t1)$ = 2.500 Watt. We calculate the power consumption at this time as follows: $P(t1) = k.U^2 = k.\sum_{j=1}^{4}(\sum_{i=1}^{m_j} \frac{C_{i,j}}{T_{i,j}})^2$=1.2012 Watt (we assume that $k$=1). $P(t1)$ is less than $P_{limit}(t1)$, then the energy constraint is satisfied.
At a particular time *t2* ($t1<t2$), we assume that a second reconfiguration scenario has been applied to the system and new task $\tau_{13}$ is added as described in Table VI.

TABLE VI. Scenario 2: added tasks.

| Task | $C_i$ | $T_i$ | $T_imax$ | $I_i$ |
|---|---|---|---|---|
| $\tau_{13}$ | 18 | 45 | 100 | 10 |

According to the period of $\tau_{13}$, we affect $\tau_{13}$ to $Core_4$ (45 $\in$ [38..46]). We compute then the processor utilization $U_{Core_4}$ for $Core_4$ as follows: $U_{Core_4} = \sum_{i=1}^{4} \frac{C_{i,4}}{T_{i,4}} = 1.07 \geq 1$. Because the value of $U_{Core_4}$ is greater than 1, then the system is not feasible. Furthermore, we check the energy constraint at this time $t2$: $P(t2) = k.U^2 = k.\sum_{j=1}^{4}(\sum_{i=1}^{m_j} \frac{C_{i,j}}{T_{i,j}})^2 = 2.2380$ Watt. As $P(t2)$ is less than $P_{limit}(t2)$, then the energy constraint is satisfied. Therefore, for this reconfiguration scenario, the real-time constraint is not satisfied.

### *B. Application of Solutions*

To resolve the real-time problem, we propose solutionA and solutionB described in [5].

***Solution A: Modification of Periods under Real-Time Scheduling Constraints:***

According to Eq. 4, the new period $T_{RT,Core_4}^{New}$ that satisfies the real-time constraint is equal to 46. Then, $U_{Core_4}$ is equal to 0.999 $\leq$ 1. It is obvious that the real-time constraint is satisfied after applying this reconfiguration scenario.

***Solution B: Modification of WCETs under Real-Time Scheduling Constraints:***

After execution of Eq. 5, $U_{Core_4}$ is equal to 0.917 $\leq$ 1. It is obvious that the real-time constraint is satisfied after applying this reconfiguration scenario. Both solutions can resolve the real-time problem introduced by the reconfiguration scenarios. To ensure the low-power consumption of system, the proposed strategy executes the solution that minimizes the processor utilization, in this case, SolutionB is more useful than SolutionA.

### *C. Performance Evaluation and Discussion*

In this section, we consider the performance evaluation of the proposed approach and compare the current paper's contribution to the related works in [2] and [4]. We assume a case study of a system composed of 40 tasks as described in Table VII that can be reconfigured at run-time under memory and energy constraints. We adopted the same set of tasks used in [2] to compare the results. Using *Reconf-Pack* simulator

(Figure.3), we can calculate the cost of our solutions compared to the proposed solutions in [2] and [4].

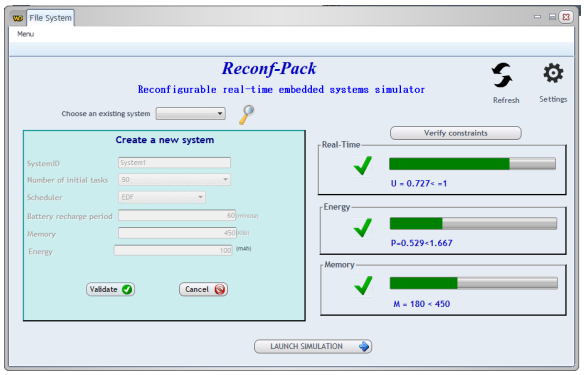


Figure 3. *Reconf-Pack* main interface.

As presented in Section IV, the cost of a solution is the total delay introduced to the periods $T_i$ of all the tasks in all the cores.

TABLE VII. System tasks.

| Task | $C_i$ | $T_i$ | Task | $C_i$ | $T_i$ |
|---|---|---|---|---|---|
| $\tau_1$ | 5 | 300 | $\tau_{21}$ | 2 | 100 |
| $\tau_2$ | 4 | 230 | $\tau_{22}$ | 3 | 80 |
| $\tau_3$ | 2 | 90 | $\tau_{23}$ | 2 | 70 |
| $\tau_4$ | 1 | 90 | $\tau_{24}$ | 4 | 250 |
| $\tau_5$ | 3 | 100 | $\tau_{25}$ | 3 | 120 |
| $\tau_6$ | 2 | 120 | $\tau_{26}$ | 2 | 130 |
| $\tau_7$ | 2 | 110 | $\tau_{27}$ | 2 | 90 |
| $\tau_8$ | 6 | 300 | $\tau_{28}$ | 3 | 120 |
| $\tau_9$ | 3 | 200 | $\tau_{29}$ | 3 | 210 |
| $\tau_{10}$ | 2 | 250 | $\tau_{30}$ | 1 | 110 |
| $\tau_{11}$ | 1 | 100 | $\tau_{31}$ | 2 | 100 |
| $\tau_{12}$ | 2 | 130 | $\tau_{32}$ | 3 | 80 |
| $\tau_{13}$ | 2 | 20 | $\tau_{33}$ | 2 | 80 |
| $\tau_{14}$ | 3 | 100 | $\tau_{34}$ | 3 | 60 |
| $\tau_{15}$ | 3 | 120 | $\tau_{35}$ | 3 | 80 |
| $\tau_{16}$ | 4 | 270 | $\tau_{36}$ | 1 | 20 |
| $\tau_{17}$ | 2 | 220 | $\tau_{37}$ | 2 | 75 |
| $\tau_{18}$ | 3 | 120 | $\tau_{38}$ | 3 | 120 |
| $\tau_{19}$ | 3 | 80 | $\tau_{39}$ | 3 | 120 |
| $\tau_{20}$ | 2 | 130 | $\tau_{40}$ | 3 | 120 |

*Reconf-Pack* implements all the tasks and executes the different reconfiguration scenarios. If the system violates one or more constraints, then *Reconf-Pack* applies the solutions proposed by Wang et al. [4] and applies our proposed strategy. In addition, we intend to develop *Reconf-Pack* towards an open source environment that supports and performs required comparisons to other related works.

According to the algorithm in Figure.1, we obtain the following distribution of tasks presented in Table VIII.

We notice that the feasibility condition is satisfied because the processor utilization $U$ for each core is less than 1.

TABLE VIII. Assigning Tasks to Cores.

| Core | Tasks | Proc. Utiliz $U$ |
|---|---|---|
| $Core_1$ | $\tau_{36},\tau_{13},\tau_{27},\tau_3,\tau_{15},\tau_6,\tau_{20},\tau_{12},\tau_{29},\ \tau_9,\tau_{26}$ | 0.311 |
| $Core_2$ | $\tau_{19},\tau_4,\tau_{11},\tau_5,\tau_{25},\tau_{18},\tau_{24},\tau_{10},\tau_2,\tau_{17},\tau_{16}$ | 0.205 |
| $Core_3$ | $\tau_{32},\tau_{22},\tau_{21},\tau_{14},\tau_{38},\tau_{28},\tau_8,\tau_1,\tau_{31}$ | 0.231 |
| $Core_4$ | $\tau_{35},\tau_{33},\tau_{30},\tau_7,\tau_{40},\tau_{39},\tau_{37},\tau_{23},\tau_{34}$ | 0.245 |

TABLE IX. System reconfiguration scenarios.

| Task | $C_i$ | $T_i$ | $T_imax$ |
|---|---|---|---|
| Reconfiguration scenario 1 | | | |
| $\tau_{41}$ | 10 | 20 | 110 |
| $\tau_{42}$ | 5 | 20 | 100 |
| Reconfiguration scenario 2 | | | |
| $\tau_{43}$ | 10 | 100 | 190 |
| $\tau_{44}$ | 20 | 80 | 160 |
| $\tau_{45}$ | 40 | 80 | 120 |
| Reconfiguration scenario 3 | | | |
| $\tau_{46}$ | 40 | 80 | 150 |
| $\tau_{47}$ | 30 | 100 | 170 |

We consider now the following three reconfiguration scenarios described in Table IX.

According to task periods, $\tau_{41}$ and $\tau_{42}$ are assigned to $Core_1$, $\tau_{43}$, $\tau_{44}$ and $\tau_{45}$ are assigned to $Core_2$, $\tau_{46}$ and $\tau_{47}$ are assigned to $Core_3$.

**Case 1: Reconfiguration scenario 1 ($Core_1$)**

We have $U_{Core_1}(t0)$=0.311 and $U_{Core_1}(t1)$=1.061 $\geq$ 1. The system is not feasible. After applying solution A, the system becomes feasible. We compare the proposed solution to the one presented by Wang et al. in [2], [4]. Both solutions introduce a change on the period of tasks. The cost of this change for each task is equal to: The new period - the initial period. We compare in Figure.4 the performance of our solution (bars in red) compared to the approaches in [2] and [4] (bars in blue).

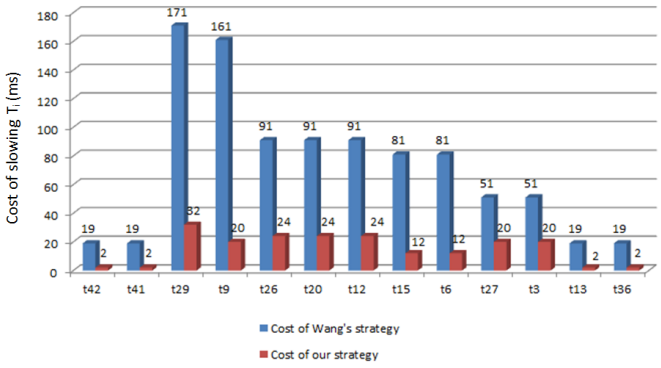


Figure 4. Cost of modification of periods $T_i$ (Solution A).

**Case 2: Reconfiguration scenario 2 ($Core_2$)**

We have $U_{Core_2}(t0)$=0.205 and $U_{Core_2}(t1)$=1.055 $\geq$ 1. Because the value of $U$ is greater than 1, the system is not feasible after the reconfiguration and Solution A is applied (Figure.5).

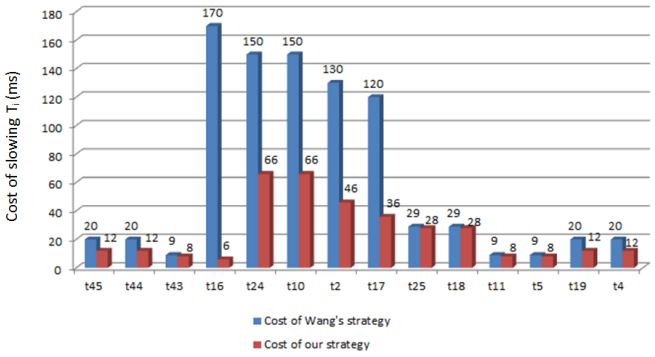


Figure 5. Cost of modification of periods $T_i$ (Solution A).

**Case 3: Reconfiguration scenario 3 ($Core_3$)**

We have $U_{Core_3}(t0)$=0.231 and $U_{Core_3}(t1)$=1.031 $\geq$ 1. Figure.6 presents the costs when SolutionA is applied.

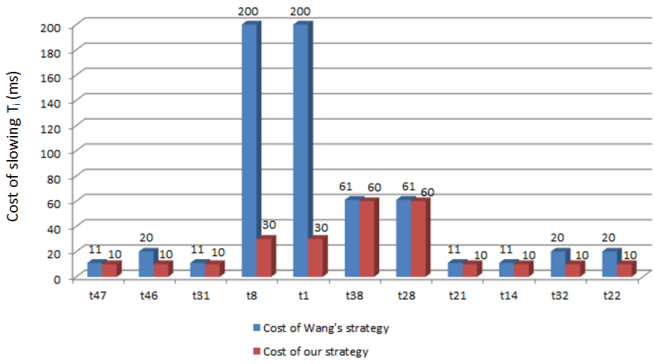


Figure 6. Cost of modification of periods $T_i$ (Solution A).

Finally, Figure.7 shows the total cost after the application of solution A on the three reconfiguration scenarios. The total

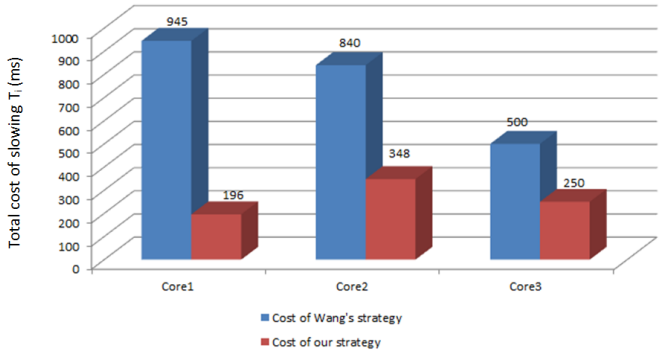


Figure 7. Total cost of modification of periods $T_i$ (Solution A)

cost is the sum of all these costs for each reconfiguration scenario. The red one represents the total cost of the proposed solution after each scenario. The one in blue represents the total cost after applying the solution described in [2] and [4]. The proposed solution introduces less delay while satisfying the three constraints after each reconfiguration.

Now, in order to generalize the performance evaluation of the proposed solution, the second module *Task-Generator* generates 200 random systems and plans randomly reconfiguration scenarios. We use then *Reconf-Pack* to apply both strategies for each system. We notice that the proposed solution introduces less delay in 100% of randomly generated tests. Moreover, the average delay introduced by our strategy to keep the system feasible and up until the next recharge is 43% only of the average delay introduced by Wang's strategy [4].

## VII. Conclusion

In this paper, the task scheduling problem for reconfigurable real-time multi-core systems powered by a battery has been considered. To ensure that the system is feasible after any reconfiguration scenario, we propose a new strategy based on the concept of packs [5]. This new strategy can initially assign tasks to different multi-core platforms and after any reconfiguration scenario, a new solution is applied to ensure the system feasibility. This strategy ensures a low-cost real-time and low-power reconfigurations of multi-core platforms. In a future work, we will focus on the implementation of the paper's contribution in a real-time operating system that will be evaluated by assuming real case studies.

## References


[1] I. R. Quadri, A. Gamatié, P. Boulet, S. Meftali, and J.-L. Dekeyser, "Expressing embedded systems configurations at high abstraction levels with uml marte profile: Advantages, limitations and alternatives," Journal of systems architecture, vol. 58, no. 5, 2012, pp. 178–194.

[2] X. Wang, M. Khalgui, and Z. Li, "Dynamic low power reconfigurations of real-time embedded systems." in PECCS, 2011, pp. 415–420.

[3] M. Khalgui, O. Mosbahi, Z. Li, and H.-M. Hanisch, "Reconfigurable multiagent embedded control systems: From modeling to implementation," Computers, IEEE Transactions on, vol. 60, no. 4, 2011, pp. 538–551.

[4] X. Wang, I. Khemaissia, M. Khalgui, Z. Li, O. Mosbahi, and M. Zhou, "Dynamic low-power reconfiguration of real-time systems with periodic and probabilistic tasks," Automation Science and Engineering, IEEE Transactions on, vol. 12, no. 1, 2015, pp. 258–271.

[5] A. Gammoudi, A. Benzina, M. Khalgui, and D. Chillet, "New pack oriented solutions for energy-aware feasible adaptive real-time systems," in Intelligent Software Methodologies, Tools and Techniques. Springer, 2015, pp. 73–86.

[6] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced cpu energy," in Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on. IEEE, 1995, pp. 374–382.

[7] G. Quan and X. S. Hu, "Minimal energy fixed-priority scheduling for variable voltage processors," Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, vol. 22, no. 8, 2003, pp. 1062–1071.

[8] T. Chantem, X. S. Hu, and M. D. Lemmon, "Generalized elastic scheduling for real-time tasks," Computers, IEEE Transactions on, vol. 58, no. 4, 2009, pp. 480–495.

[9] A. Bart, C. Truchet, and E. Monfroy, "Verifying a real-time language with constraints," in Tools with Artificial Intelligence (ICTAI), 2015 IEEE 27th International Conference on. IEEE, 2015, pp. 844–851.

[10] S. Chen, J. Hao, G. Weiss, S. Zhou, and Z. Zhang, "Toward efficient agreements in real-time multilateral agent-based negotiations," in Tools with Artificial Intelligence (ICTAI), 2015 IEEE 27th International Conference on. IEEE, 2015, pp. 896–903.

[11] S. P. Dwivedi, "Adaptive scheduling in real-time systems through period adjustment," arXiv preprint arXiv:1212.3502, 2012.

[12] S. Baruah and J. Goossens, "Scheduling real-time tasks: Algorithms and complexity," Handbook of scheduling: Algorithms, models, and performance analysis, vol. 3, 2004.

[13] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," Journal of the ACM (JACM), vol. 20, no. 1, 1973, pp. 46–61.

[14] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," in Design Automation Conference, 1999. Proceedings. 36th. IEEE, 1999, pp. 134–139.

[15] A. Gharbi, M. Khalgui, and S. B. Ahmed, "Inter-agents communication protocol for distributed reconfigurable control software components," in The International Conference on Ambient Systems Networks and Technologies (ANT), 2010, pp. 8–10.

[16] H. Gharsellaoui and S. B. Ahmed, "Real-time reconfigurable scheduling of sporadic tasks," in Software Technologies. Springer, 2013, pp. 24–39.

[17] C.-C. Lin, Y.-C. Syu, C.-J. Chang, J.-J. Wu, P. Liu, P.-W. Cheng, and W.-T. Hsu, "Energy-efficient task scheduling for multi-core platforms with per-core dvfs," Journal of Parallel and Distributed Computing, vol. 86, 2015, pp. 71–81.

[18] C.-C. Lin, C.-J. Chang, Y.-C. Syu, J.-J. Wu, P. Liu, P.-W. Cheng, and W.-T. Hsu, "An energy-efficient task scheduler for multi-core platforms with per-core dvfs based on task characteristics," in 2014 43rd International Conference on Parallel Processing. IEEE, 2014, pp. 381–390.