On-line Scheduling in Multiprocessor Systems based on continuous control using Timed Continuous Petri Nets

G. Desirena-López [1], C.R. Vázquez[2], J.L. Briz[3], A. Ramírez-Treviño[1], and D. Gómez-Gutiérrez[1] .

***Abstract*— This work presents a *fluid*-time scheduler based on a sliding mode controller where the sliding surface is related to fluid task executions. The scheduler is applied to a model of tasks and $CPUs$ designed with Timed Continuous Petri Nets (*TCPN*) under the infinite server semantics (ISS). Also, the paper proposes an implementation of this fluid scheduler as a feasible discrete scheduler where the number of task migrations and preemptions is bounded.**

Keywords: TCPN, Modeling, Control, Scheduling.

I. INTRODUCTION

Modern embedded systems are increasingly more demanding, mostly due to the large number of tasks they must execute in a very short period, subject to real time and energy constraints. Real-time schedulers assure that all tasks meet their period and deadlines. They must be correct, but also feasible to implement, and optimal in terms of CPU usage.

A real-time scheduler ensures that a task's instance activates according to a period and finishes before a deadline. With this purpose, some schedulers order tasks according to a fixed or dynamic priority. Rate Monotonic (RM) and Deadline Monotonic (DM) analysis ensure that a schedule is correct if the periodic instances (*jobs*) of each task are given a fixed priority according to their period (RM) or deadline (DM) [1]. Alternatively, dynamic priority schedulers such as EDF (Earliest Dead Line First) and LLF (Least Laxity First) dynamically compute a job's priority according to the current system state. RM guarantees valid schedules on uniprocessor systems at the cost of wasting about 30% of the CPU time whereas EDF and LLF are optimal [1], [2], but this optimality does not hold on multiprocessors [3], [4]. *Fair* scheduling algorithms try to follow as closely as possible the *fluid* schedule, i.e., a schedule in which each task is executed at constant rate, to guarantee an optimal schedule in multiprocessor systems [5]. *Fair* schedulers allow a better control over the progress of each job while honoring periods and deadlines, but their implementation entails a large number of task preemptions (context switches), which hampers performance in practice. Proportionate Fairness (P-Fair) schedulers [6] are based on the rationale of fluid schedulers. In P-Fair all tasks' jobs are executed at an approximately uniform rate by splitting them into sub-tasks which run for a time quantum. Other scheduling algorithms have emerged from this notion of proportionate fairness, like PD [7], PD^2 [8], $DP-Fair$ [5] and $BFair$ [9]. As all fair schedulers, they are optimal in multiprocessor systems but suffer from a large number of task preemptions and migrations. Hence, while these algorithms are theoretically optimal or high-performing, they are not feasible in real applications. Moreover, they cannot easily incorporate some continuous requirements demanded by modern embedded systems such as power consumption or thermal issues.

The main contribution of this paper is to propose an on-line *fluid* scheduler based on a sliding mode control technique [10], capable of integrating the CPU thermal models presented in [11]. We first propose a methodology to model the period, deadline and CPU cycles of each task, along with a set of $CPUs$, by means of a Timed Continuous Petri Net ($TCPN$). Based on this model, we build the on-line *fluid* scheduler. Through a Lyapunov stability analysis, the *fluid* scheduling is guaranteed, so it is optimal. We also introduce an algorithm to implement this *fluid* schedule as a discrete scheduler, where the number of task migrations and preemptions is bounded. We leverage the $TCPN$ instantaneous marking to define priority firing rules and determine how tasks are allocated to processors. We prove that the size of the $TCPN$ models increase linearly with the number of tasks and the complexity of the proposed algorithm is polynomial.

The paper is organized as follows. Section II provides basic definitions. Section III explains the model of periodic real-time tasks. Section IV introduces the CPU model, the allocation of tasks to $CPUs$ and the global model. The *fluid* scheduler based on a sliding mode controller is presented in Section V. Section VI presents a discretization of the *fluid* scheduler. Section VII shows an illustrative example of a multiprocessor system. Finally, Section VIII concludes the paper.

II. FUNDAMENTALS

This section introduces basic definitions concerning Petri nets and continuous Petri nets. An interested reader may also consult [12], [13] to get a deeper insight in the field. In the sequel, given a matrix $\mathbf{A}$ and sets of indices (or nodes) $I = \{i_1, ..., i_n\}$ and $J = \{j_1, ..., j_m\}$, it will be denoted as $\mathbf{A}[I, J]$ the matrix built with the elements in the rows related to I and the columns related to J. The same notation will be used with vectors. Furthermore, a_j will be used to denote the j-th element of vector $\boldsymbol{a}$.

*This work is partially supported by grants TIN2013-46957-C2-1-P and T48 RG (Aragon Gov. and ESF).

[1]G. Desirena-López, A. Ramírez-Treviño and D. Gómez-Gutiérrez are with the CINVESTAV-IPN Unidad Guadalajara, Av. del Bosque 1145, CP 45019, Zapopan, Jalisco, Mexico {gdesirena},{art},{dgomez}@gdl.cinvestav.mx

[2]C.R. Vázquez is with ITESM, Av. Ramón Corona 2514, CP 45201,Zapopan, Jalisco, Mexico cr.vazquez@itesm.mx

[3]J.L. Briz is with the DIIS/I3A Univ. de Zaragoza, María de Luna 1 - 50018 Zaragoza, España. briz@unizar.es

A. Timed continuous Petri nets

Definition 2.1: A (discrete) Petri net structure (PN) is a graph described by a 4-tuple $N = (P, T, \boldsymbol{Pre}, \boldsymbol{Post})$ where P and T are finite disjoint sets of places and transitions, respectively. $\boldsymbol{Pre}$ and $\boldsymbol{Post}$ are $|P| \times |T|$ $\boldsymbol{Pre}-$ and $\boldsymbol{Post}-$ incidence matrices, where $\boldsymbol{Pre}[i,j] > 0$ (resp. $\boldsymbol{Post}[i,j] > 0$) if there is an arc going from p_i to t_j (resp. going from t_j to p_i), $\boldsymbol{Pre}[i,j] = 0$ (resp. $\boldsymbol{Post}[i,j] = 0$) otherwise.

Definition 2.2: A (discrete) Petri net system is the pair $Q = (N, M)$ where N is a Petri net and $\boldsymbol{M} : P \to \mathbb{N} \cup \{0\}$ is the marking function assigning zero or a natural number to each place. The value $\boldsymbol{M}[p_i]$ is named the tokens residing into p_i. $\boldsymbol{M}_0$ is named the initial marking.

Definition 2.3: A timed continuous Petri net (*TCPN*) is a time-driven continuous-state system described by the tuple $(N, \boldsymbol{\lambda}, \boldsymbol{m}_0)$ where N is a *PN* structure and the vector $\boldsymbol{\lambda} \in \{\mathbb{R}^+ \cup 0\}^{|T|}$ represents the transitions rates determining the temporal evolution of the system. Transitions fire according to certain speed, which generally is a function of the transition rates and the current marking. Such function depends on the semantics associated to the transitions. Under infinite server semantics, the flow (the transitions firing speed, denoted as $\boldsymbol{f(m)}$) through a transition t_{ω_i} is defined as the product of the rate, λ_i, and $enab(t_i, \boldsymbol{m})$, the instantaneous enabling of the transition, i.e., $f_i(\boldsymbol{m}) = \lambda_i \, enab(t_i, \boldsymbol{m}) = \lambda_i min_{p_j \in {}^\bullet t_i}(\boldsymbol{m}[p_j]/\boldsymbol{Pre}[p_j, t_i])$.
The firing rate matrix is defined by $\boldsymbol{\Lambda} = diag(\lambda_1, ..., \lambda_{|T|})$. For the flow to be well defined, every continuous transition must have at least one input place, hence in the following we will assume $\forall t \in T, |{}^\bullet t| \geq 1$. The "min" in the above definition leads to the concept of configuration. A configuration of a *TCPN* at $\boldsymbol{m}$ is a set of (p, t) arcs describing the effective flow of each transition, in that case we say that p_i constrains t_j for each arc (p_i, t_j) in the configuration. A configuration matrix is defined for each configuration as follows:

$$\boldsymbol{\Pi(m)} = \begin{cases} \frac{1}{\boldsymbol{Pre}[i,j]} & \text{if } p_i \text{ is constraining } t_j \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The flow through the transitions can be written in a vectorial form as $\boldsymbol{f(m) = \Lambda \Pi(m) m}$. We can apply a control action to the dynamical behavior of a *PN* system by adding a term $\boldsymbol{u}$ to every transition t_i such that $0 \leq u_i \leq f_i$, indicating that its flow can be reduced. Thus, the controlled flow of transition t_i becomes $w_i = f_i - u_i$ and the forced state equation is

$$\begin{gathered} \dot{\boldsymbol{m}} = \boldsymbol{C}[\boldsymbol{f} - \boldsymbol{u}] = \boldsymbol{Cw} \\ 0 \leq u_i \leq f_i \end{gathered} \tag{2}$$

III. Modeling tasks

In this work we consider that a) Tasks are periodic, and their time periods are fixed and known; b) Tasks are independent, meaning that there is no precedence in their execution and that they do not share resources or interact to each other; c) Each task's execution time, given in CPU cycles, is known and fixed; d) Each task has an associated deadline. Since tasks are periodic, this deadline is relative to the starting of the corresponding period. We formalise these assumptions by considering a set of independent periodic real-time tasks $\mathcal{T} = \{\tau_1, ..., \tau_n\}$, where each task $\tau_i = (cc_i, d_i, \omega_i)$ consists of an infinite sequence of jobs that are released according to a period ω_i. The $k-th$ job of τ_i, denoted as τ_i^k, runs during cc_i CPU cycles, becoming enabled at time $(k-1) \cdot \omega_i$. The job must be completed before its deadline $(k-1) \cdot \omega_i + d_i$ ($k-th$ relative deadline). We assume that there exists m identical processors and n tasks. We also assume that all task parameters, including task period and execution time are integers and that any task can be preempted at any time. We define the hyper-period as the period equal to the least common multiple of periods $H = lcm(\omega_1, \omega_2, \ldots, \omega_n)$ of the n periodic tasks. A feasible schedule can be repeated every hyper-period [1].

The proposed modeling methodology models each task as the $TCPN$ module of Fig. 1(a), which is explained below.

1) Modeling task execution: The period ω_i of task τ_i implies that, in average, $\frac{1}{\omega_i}$ jobs arrive per second (i.e., arriving frequency). This is captured in the $TCPN$ module of Fig. 1(a) as the firing rate $\lambda_i^\omega = \frac{1}{\omega_i}$ of transition t_i^ω.

2) Modeling task deadline: The relative deadline d_i of task τ_i is represented in the model of Fig. 1(a) by the marking d_i at place p_i^d.

3) Modeling duration of a task: The duration of a task is represented in the $TCPN$ model of Fig. 1(a) by the arc going from transition t_i^ω, representing the jobs arrival, to the place p_i^{cc}, representing the jobs that have arrived. The weight cc_i of the arc is included so the marking at p_i^{cc} represents the jobs that have arrived (to be executed) in CPU cycle units.

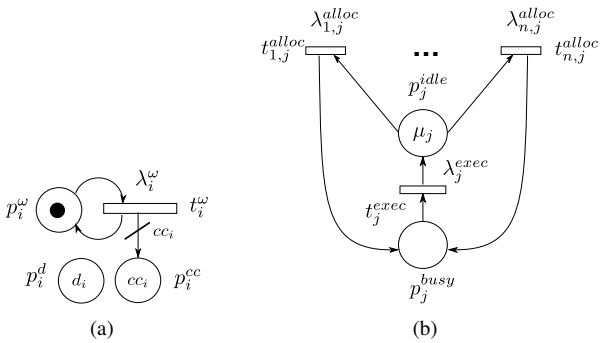

Fig. 1. (a) $TCPN$ module for task τ_i. (b) $TCPN$ module for CPU_j.

IV. Modeling task allocation CPU and global model

We assume in this paper that all CPUs have the same capabilities, although this modeling methodology can be applied to heterogeneous cores. Thus, any task in the set $\mathcal{T}$ can be allocated to any CPU in the set $\mathcal{P} = \{CPU_1, \ldots, CPU_m\}$. We also cling to the common assumption that task migration and preemption have no cost and every task's job must execute sequentially on at most one processor at any given instant in time.

A. Modeling task allocation to CPU

A single CPU_j is modelled by the $TCPN$ module of Fig. 1(b), consisting of places p_j^{busy}, p_j^{idle} and transitions t_j^{exec}, $t_{1,j}^{alloc}$, ... , $t_{n,j}^{alloc}$. The marking at place p_j^{idle} models the number of available CPU_j cycles per second (throughput capacity). The initial marking at p_j^{idle} is the maximum throughput μ_j of CPU_j. Place p_j^{busy} represents the busy state of the processor, the marking at this place represents the number of CPU_j cycles per second reserved for tasks execution (throughput being used). Transition t_j^{exec} models the CPU_j execution rate. Transitions $t_{1,j}^{alloc}$, ... , $t_{n,j}^{alloc}$ model the allocation of tasks τ_1 , ..., τ_n, respectively, to processor CPU_j.

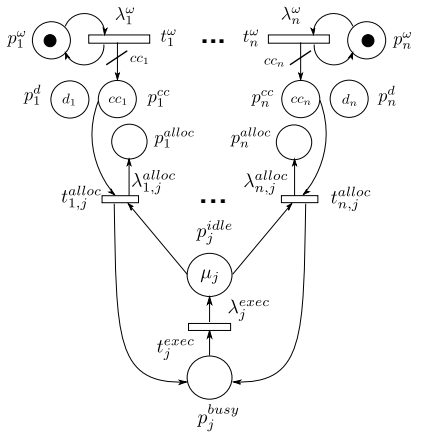

Fig. 2. TCPN module allocation of n tasks to a single CPU_j.

1) Building the global model: The global model is obtained by merging the tasks models (Fig. 1(a)) and the allocation $CPUs$ models (Fig. 1(b)). The global model for a single processor CPU_j is depicted in Fig. 2. The arcs from places p_i^{cc} to transitions $t_{i,j}^{alloc}$ represent that jobs of τ_i are being allocated to processor CPU_j. To merge the models, we add places p_i^{alloc} and arcs going from $t_{i,j}^{alloc}$ to p_i^{alloc}. The marking of place p_i^{alloc} stands for the total amount of jobs of τ_i that has been allocated to CPU_j from the initial time.

The global model, encompassing n tasks and m $CPUs$, is depicted in Fig 3. Note that the resulting global model is a deadlock-free Petri net.

B. Fundamental equation of the global model

This equation dictates the dynamic evolution of the global $TCPN$ model (Fig. 3), and is given by:

$$\begin{aligned} \dot{\boldsymbol{m}} &= \boldsymbol{C}\boldsymbol{\Lambda}\boldsymbol{\Pi}(\boldsymbol{m})\boldsymbol{m} - \hat{\boldsymbol{C}}\boldsymbol{u} \\ \mathbf{0} &\leq \boldsymbol{u} \leq (\boldsymbol{\Lambda}\boldsymbol{\Pi}(\boldsymbol{m})\boldsymbol{m})[T^{alloc}] \end{aligned} \tag{3}$$

where $\hat{\boldsymbol{C}}$ has the columns of $\boldsymbol{C}$ corresponding to transitions $t_{i,j}^{alloc}$ (denoted as T^{alloc}) in Fig. 3. The flow through these transitions controls the rate at which an scheduler allocates jobs to each CPU_j. The control vector $\boldsymbol{u}$ represents the speed reductions (from the autonomous evolution) for the allocation of jobs to $CPUs$, i.e., if $\boldsymbol{u} = \mathbf{0}$ then tasks are allocated according to the available jobs and $CPUs$ (given by $(\boldsymbol{\Lambda}\boldsymbol{\Pi}(\boldsymbol{m})\boldsymbol{m})[T^{alloc}]$) but if $\boldsymbol{u} = (\boldsymbol{\Lambda}\boldsymbol{\Pi}(\boldsymbol{m})\boldsymbol{m})[T^{alloc}]$ then no job is allocated.

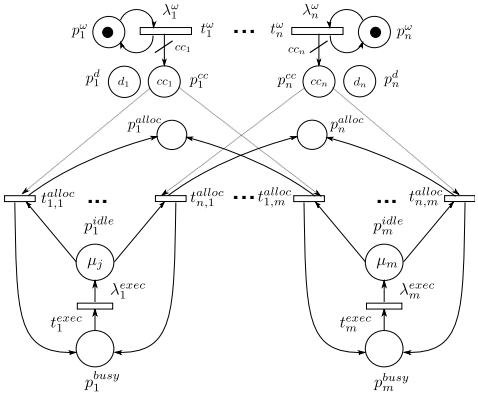

Fig. 3. Global $TCPN$ model of n tasks allocated on m $CPUs$.

V. Fluid scheduling of a set of real-time tasks

Since the obtained global $TCPN$ is modeled by differential state equations, different control techniques may be applied. This section proposes a *fluid* schedule based on a sliding mode control technique [10]. From now on, ζ represents the current time. The control approach herein reported starts by computing the task *fluid*-schedule function:

$$FSC_{\tau_i}(\zeta) = \frac{cc_i}{\omega_i}\zeta \tag{4}$$

According to the *fluid* algorithms [6] [9], this function represents the optimal *fluid* execution of task τ_i. Through this work, we will say that this function represents the *execution percentage* of task τ_i at time ζ.

The *execution error* of task τ_i (denoted $E_{\tau_i}(\zeta)$) is the difference between the marking of places p_i^{alloc} (total amount of jobs of τ_i allocated) in the global $TCPN$ model and the optimal *fluid* execution.

$$E_{\tau_i}(\zeta) = m_i^{alloc}(\zeta) - FSC_{\tau_i}(\zeta) \tag{5}$$

Thus, if $E_{\tau_i}(\zeta) = 0$, $\forall \zeta > \zeta_a$, then the marking in places p_i^{alloc} is equal to the optimal fluid schedule; hence the firing of transitions $t_{i,j}^{alloc}$ represents the optimal task allocation to $CPU's$. In order to bring the error to zero, for each task τ_i and CPU_j, the following continuous sliding-mode control law is used [10]:

$$u_{i,j}^{alloc}(\zeta) = \left[\tfrac{1}{2} + \tfrac{1}{2}sign(E_{\tau_i}(\zeta))\right] f_{i,j}^{alloc}(\zeta) \tag{6}$$

where

$$sign(x) = \begin{cases} 1 & if \quad x \geq 0 \\ -1 & if \quad x < 0 \end{cases}$$

and

$$f_{i,j}^{alloc}(\zeta) = \lambda_{i,j}^{alloc} min(m_i^{cc}(\zeta), m_j^{idle}(\zeta))$$

When this control law is applied to the system (3), for each task τ_i and CPU_j, then each $E_{\tau_i}(\zeta)$ becomes zero and the *fluid* schedule meets the time constraints. This is formalized in the following proposition.

Proposition 5.1: Let $\mathcal{T}$ and $\mathcal{P}$ be the sets of n tasks and m processors, respectively, where the fluid execution tasks $FSC_{\tau_i}(\zeta)$ and execution errors $E_{\tau_i}(\zeta)$ are defined.

If the control law given by (6) is applied to the system (3), for each task τ_i and CPU_j, then the *execution errors* converge to zero asymptotically.

Proof: In order to proof that each E_{τ_i} converges to zero asymptotically, the following assumptions are made: first, since the fluid schedule can allocate all the jobs in such a way that they are executed in time by all the processors, the model assumes that there are enough CPU cycles available in each place p_j^{idle}, so these places do not restrict the flows $f_{i,j}^{alloc}$, consequently $f_{i,j}^{alloc} = \lambda_{i,j}^{alloc} m_i^{cc}$ for each τ_i and CPU_j. Second, places p_j^{busy} and p_j^{idle} do not affect the control. Therefore, the global model will be analysed in the sequel without considering these places. The controlled flow of a transition $t_{i,j}^{alloc}$ is $w_{i,j}^{alloc} = f_{i,j}^{alloc} - u_{i,j}^{alloc}$, where $u_{i,j}^{alloc}$ is the control action as defined in (6). Note that when $w_{i,j}^{alloc} = f_{i,j}^{alloc} - u_{i,j}^{alloc} > 0$, transition $t_{i,j}^{alloc}$ is fired, i.e., jobs of τ_i are being allocated to CPU_j. Therefore, the evolution of task τ_i is described by the following state variables:

$$\begin{aligned} \overset{\bullet}{m}_i^{\omega} &= 0 \\ \overset{\bullet}{m}_i^{cc} &= \tfrac{c_i}{\omega_i} - \sum_{j=1}^{m} w_{i,j}^{alloc} \\ \overset{\bullet}{m}_i^{alloc} &= \sum_{j=1}^{m} w_{i,j}^{alloc} \end{aligned} \tag{7}$$

We have to drive the variables E_{τ_i} to zero by means of the control actions $u_{i,1}^{alloc}, \ldots, u_{i,m}^{alloc}$. In order to prove the asymptotic stability of (5), a Lyapunov function (see, for instance, [14]) can be defined, satisfying $V(0) = 0$, $V(x) > 0$ and $\overset{\bullet}{V}(x) < 0 \ \forall x \neq 0$. The candidate Lyapunov function V considered here is

$$V(E_{\tau_1}, \ldots, E_{\tau_n}) = \tfrac{1}{2}E_{\tau_1}^2 + \ldots + \tfrac{1}{2}E_{\tau_n}^2 \tag{8}$$

Note that $V = 0$ iff each $E_{\tau_i} = 0$. Furthermore, $V > 0$ if any $E_{\tau_i} \neq 0$. Thus, V can be considered a Lyapunov function (and thus (5) is asymptotic stable) iff $\overset{\bullet}{V} < 0$ for any $E_{\tau_i} \neq 0$. To prove this, the derivative of V is computed as:

$$\begin{aligned} \overset{\bullet}{V} &= \sum_{i=1}^{n} E_{\tau_i} \overset{\bullet}{E}_{\tau_i} \\ &= \sum_{i=1}^{n} \left(\sum_{j=1}^{m} \left[\tfrac{1}{2} E_{\tau_i} f_{i,j}^{alloc} - \tfrac{1}{2} |E_{\tau_i}| f_{i,j}^{alloc} \right] - E_{\tau_i} \tfrac{cc_i}{\omega_i} \right) \end{aligned} \tag{9}$$

Now, two cases are analyzed for each term of the first sum (corresponding to each task):

$$\sum_{j=1}^{m} \left[\frac{1}{2} E_{\tau_i} f_{i,j}^{alloc} - \frac{1}{2} |E_{\tau_i}| f_{i,j}^{alloc} \right] - E_{\tau_i} \frac{cc_i}{\omega_i} \tag{10}$$

a) When E_{τ_i} is positive. The term (10) for the task τ_i becomes $-E_{\tau_i} \frac{cc_i}{\omega_i}$. Note that $0 < cc_i/\omega_i$, then the term is negative.

b) When E_{τ_i} is negative. The term (10) for the task τ_i becomes $|E_{\tau_i}|(\frac{cc_i}{\omega_i} - \sum_{j=1}^{m} f_{i,j}^{alloc})$.

Now, the flow $f_{i,j}^{alloc}$ through each transition $t_{i,j}^{alloc}$ is computed as $f_{i,j}^{alloc} = \lambda_{i,j}^{alloc} m_i^{cc}$. Thus, the term above is negative iff $\frac{cc_i}{\omega_i} - \lambda_i^{alloc} m_i^{cc} < 0$. where $\sum_{j=1}^{m} \lambda_{i,j}^{alloc} = \lambda_i^{alloc}$. In order to prove this inequality, it is required to compute a lower bound for m_i^{cc}. For this, the dynamic of m_i^{cc} is represented as $\overset{\bullet}{m}_i^{cc} = \frac{cc_i}{\omega_i} - \sum_{j=1}^{m} \left(f_{i,j}^{alloc} - u_{i,j}^{alloc} \right)$. Since $E_{\tau_i} < 0$, then each control action $u_{i,j} = 0$. Thus, $\overset{\bullet}{m}_i^{cc} = cc_i/\omega_i - \lambda_i^{alloc} m_i^{cc}$. The solution of this differential equation leads to

$$m_i^{cc}(\zeta) = \tfrac{cc_i}{\omega_i \lambda_i^{alloc}} + \left(m_i^{cc}(0) - \tfrac{cc_i}{\omega_i \lambda_i^{alloc}} \right) e^{-\lambda_i^{alloc} \zeta}$$

Therefore, $m_i^{cc}(\zeta)$ is bounded, in fact $m_i^{cc}(\zeta) > min(m_i^{cc}(0), cc_i/(\omega_i \lambda_i^{alloc}))$. Assume that $\omega_i \lambda_i^{alloc} > 1$, i.e., the total allocation rate λ_i^{alloc} (not execution rate) for task τ_i is larger than its arrival frequency $1/\omega_i$. Thus, since $m_i^{cc}(0) = cc_i$ then $m_i^{cc}(0) > cc_i/(\omega_i \lambda_i^{alloc})$. The bound for $m_i^{cc}(\zeta)$ is then given by $m_i^{cc}(\zeta) > cc_i/(\omega_i \lambda_i^{alloc})$, which implies $cc_i/\omega_i - \lambda_i^{alloc} m_i^{cc} < 0$, consequently (10) is negative.

Finally, since each term in the sum in (9) is negative then $\overset{\bullet}{V} < 0$.

■

A negative task error $E_{\tau_i}(\zeta)$ reveals the existence of unattended jobs, so the control (6) turns on transitions $t_{i,j}^{alloc}$ in order to allocate jobs to the processors. Otherwise, if $E_{\tau_i}(\zeta)$ is positive or zero, it means that the processors are executing jobs on time, and the control turns off the transitions $t_{i,j}^{alloc}$, i.e. stops allocating jobs. The complexity of the *fluid* scheduler depends on the numerical method used to solve the differential equation (3) (for instance, Runge-Kutta, Euler's Method, etc). All these methods and (3) are solved in polynomial time at every integration step.

VI. On-line discretization of a Fluid schedule

We have proved that the *fluid* scheduler proposed in the previous section is theoretically feasible (i.e, the execution error converges to zero and tasks meet the time constraints). As all *fluid* schedulers, it triggers an unfeasible number of task preemptions and migrations. To deal with this issue we provide a discrete implementation described by Algorithm 1.

Due to the periodicity of the schedule, we can limit the schedule of the tasks up to the hyper-period (from time 0 to time H) [6]. As mentioned before, a job τ_i^k must be completed before its $k-th$ deadline $sd_i^k = (k-1)\omega_i + d_i$, in absolute time. We define $SD_i = \{sd_i^1, sd_i^2, ...\}$ as the set of all deadlines for all task's jobs between zero and H. By defining $SD = SD_1 \cup ... \cup SD_{|\mathcal{T}|}$, the elements of SD are renamed in ascendant order, according to their value, as $SD = \{sd_0, ..., sd_r\}$. Algorithm 1 requires a time period (quantum) Q, which is defined as the greatest common

divisor of the elements $sd_i \in SD$ and the values of the function FSC_{τ_i} evaluated at sd_i .

Algorithm 1 On-line discretization of a *fluid* schedule

1: **Input** The $TCPN$ and discrete PN of the set of tasks $\mathcal{T}$, the ordered set SD where any $sd_k \in SD$ is lower or equal than H. The quantum Q. The task *fluid*-schedule functions FSC_{τ_i}.
2: **Output** A feasible discrete schedule.
3: **Initialize** $i = 1$, $sd = sd_i$, $\zeta = 0$
4: **while** $\zeta \leq H$ **do**
5: *All tasks are preempted from the processors*
6: *Compute remaining jobs:* $RE_{\tau_i}(\zeta) = FSC_{\tau_i}(sd) - M_{\tau_i}(\zeta)$
7: *Compute the set of transitions* t_{τ_i} *to be fired in the discrete model:* $ET(\zeta) = \{t_{\tau_i} | RE_{\tau_i}(\zeta) > 0\}$
8: *Compute the priority for every transition* t_{τ_i} *in ET:* $PR_{\tau_i}(\zeta) = m_i^{alloc}(\zeta) - M_{\tau_i}(\zeta)$
9: ***for*** $j = 1$ ***to*** m ***do***
10: *Select* t_{τ_a} *with the highest priority value* PR_{τ_a} *in ET*
11: *Fire* t_{τ_a} *in the discrete PN, assign task* τ_a *to processor j from* ζ *to* $\zeta + Q$
12: *Remove* t_{τ_a} *from ET*
13: ***end for***
14: *SIMULATE the global TCPN model from* ζ *to* $\zeta + Q$
15: *Update time:* $\zeta = \zeta + Q$
16: ***if*** $\zeta == sd$ ***then***
17: $i = i + 1, sd = sd_i$
18: ***end if***
19: **end while**

Algorithm 1 computes a discrete schedule from the fluid schedule introduced in Section V. It requires of a new discrete Petri net where this PN has one source transition t_{τ_i} and one sink place p_{τ_i} per task τ_i. The firing of a transition t_{τ_i}, determined by the algorithm, means that Q CPU cycles of task τ_i are allocated. The marking of a place p_{τ_i}, denoted M_{τ_i}, represents the total amount of CPU cycles of task τ_i that has been allocated from the initial time, and it constitutes the analogue of place p_i^{alloc} in the fluid model of Fig. 3. The discrete schedule resulting from Algorithm 1 equals the fluid schedule at every deadline time $sd_k \in SD$, i.e. it ensures that the discrete schedule meets all deadlines of all tasks.

If at any time ζ, $sd_i < \zeta < sd_{i+1}$, it holds that $FSC(sd_k) > M_{\tau_i}(\zeta)$ (the required fluid schedule at the end of the interval is bigger than the current discrete schedule), then τ_i must be allocated in a CPU. Thus the m tasks with the current positive greatest *remaining jobs execution* ($RE_{\tau_i}(\zeta) = FSC(sd_k) - M_{\tau_i}(\zeta)$) and *task priority function* $PR_{\tau_i}(\zeta) = m_i^{alloc}(\zeta) - M_{\tau_i}(\zeta)$ must be allocated to a CPU.

In this paper, the value of $m_i^{alloc}(\zeta)$ can be replaced by $FSC(\zeta)$ since they have the same value. However, we use it here because in our target systems we will include thermal characteristics, making these values no longer be equal because we will have to balance thermal and temporal trade-offs. Note that the algorithm executes $I = \frac{H}{Q}$ times, where H is the hyper-period, thus the loop in step 4 of the algorithm runs I times. The instruction inside this loop runs in polynomial time in the size of the transitions of the $TCPN$. As mentioned in previous section, the execution on the $TCPN$ is polynomial and therefore the algorithm is polynomial too.

Proposition 6.1: If a feasible *fluid* schedule is given as input to Algorithm 1, then the resulting discrete schedule has the following properties.

1) It meets task time constraints at every scheduling point $sd_k \in SD$.
2) The number of task migrations and preemptions is bounded.

Proof: Part 1) Sentence 1.

From the definition of quantum, we know that the time interval $[sd_k, sd_{k+1}]$ is divided by the quantum into $D_k^{k+1} = (sd_{k+1} - sd_k)/Q$ time sub-intervals. From [6] we know that the *fluid* schedule meets the task time constraints at every time, which is specially true at the sd_k points. Moreover, since the *fluid* schedule is feasible then, at time sd_k, the $CPU's$ are capable to execute the required percentage $FSC_{\tau_a}(sd_k)$ of any task τ_a. Assuming that there exist m processors and n tasks, and since the m processors are capable to execute the *fluid* schedule, then:

$$m \cdot D_k^{k+1} \geq \sum_{i=1}^{n} \left(FSC_{\tau_i}(sd_{k+1}) - FSC_{\tau_i}(sd_k)\right) \quad (11)$$

At time zero, both the *fluid* schedule and discrete schedule have executed zero percentage of each task, thus the discrete schedule meets task time constraints at sd_0.

Now, we will show that if the discrete schedule meets the *fluid* schedule at any sd_k then it meets the *fluid* schedule at any sd_{k+1} as well.

Proceeding by contradiction, assume that the discrete schedule does not meet the *fluid* schedule at sd_{k+1}. Then the remaining jobs functions are positive for some tasks (i.e. the discrete PN has executed these tasks in a lower percentage than the *fluid* one). For the sake of explanation, suppose that there exists only one task τ_a such that $RE_{\tau_a}(sd_{k+1}) = N_a > 0$. Since (11) holds, then the processors have the capability to execute the required percentage of tasks at time sd_{k+1}. However, since τ_a was not allocated in the required percentage ($FSC_{\tau_a}(sd_{k+1})$), then τ_a had some remaining jobs at time $\zeta = sd_{k+1} - Q$. Therefore, two cases are possible:

Case 1: If $N_a = \alpha Q$, where $\alpha = 1$ (i.e., $PR_{\tau_a} = Q$)

In this case two possibilities arise:

a) τ_a was fired at time ζ, then τ_a finishes the required percentage of execution, i.e., $RE_{\tau_a}(sd_{k+1}) = 0$, which is a contradiction.

b) τ_a was not allocated at time ζ. Thus, according to step 6 of the algorithm, m tasks, different from τ_a, were found having priority larger or equal than that of τ_a, thus they were allocated.

If the behavior of the algorithm is analyzed at time $\zeta - Q$ (a previous time step), it will result that m tasks were found, different than τ_a, having larger or equal priorities than that of τ_a (otherwise, task τ_a would be allocated and thus finished its execution). By repeating this analysis, going back in time until sd_k, it will be obtained that $RE_{\tau_a}(sd_k) = N_a > 0$, i.e., the discrete schedule does not meet the *fluid* schedule at sd_k, a contradiction.

Case 2: If $N_a = \alpha Q, \alpha > 1$, then at time ζ, $PR_{\tau_a}(\zeta) > PR_{\tau_i}(\zeta)$, for some task $\tau_i \neq \tau_a$. Thus τ_a was allocated at time ζ and $\alpha = \alpha - 1$. Let $\zeta = \zeta - Q$. If sd_k is reached then

$RE_{\tau_a}(sd_k) = N_a > 0$, i.e., the discrete schedule does not meet the *fluid* schedule at sd_k, a contradiction, otherwise if $\alpha - 1 == 1$ then go to Case 1.

Part 2) Sentence 2.

Since the hyper-period H is divided into $I = \frac{H}{Q}$ subintervals and task migrations and preemptions occur at the end of this subintervals then the number of task migrations and preemptions is bounded by I. ■

VII. Example

There exists a set of tasks: $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$, where $\tau_1 = (9, 10, 10)$, $\tau_2 = (9, 10, 10)$, $\tau_3 = (8, 40, 40)$ running on two processors are considered. Fig. 2 shows the $TCPN$ global model for three tasks and two processors. The hyper-period is $H = 40$. Applying the *fluid* controller, the *fluid* schedule, marking in $m_i^{alloc}(\zeta)$, is obtained. Fig. 4 shows the fluid schedule and that the errors E_{τ_i} converge to zero, therefore the *fluid* schedule based on the sliding-mode control meets the time constraints.

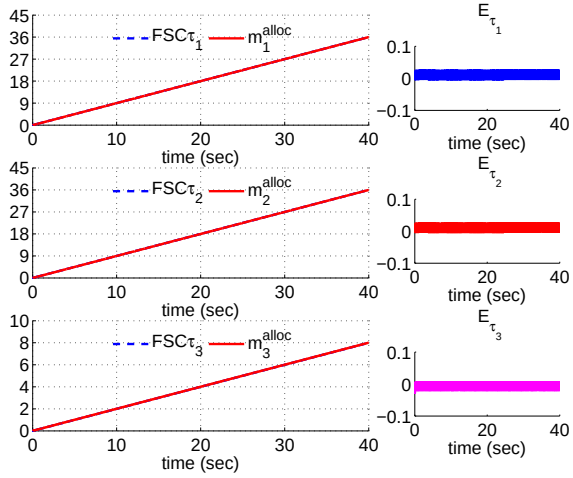

Fig. 4. A feasible *fluid* schedule. The *execution error* E_{τ_i} converges to zero due the control action.

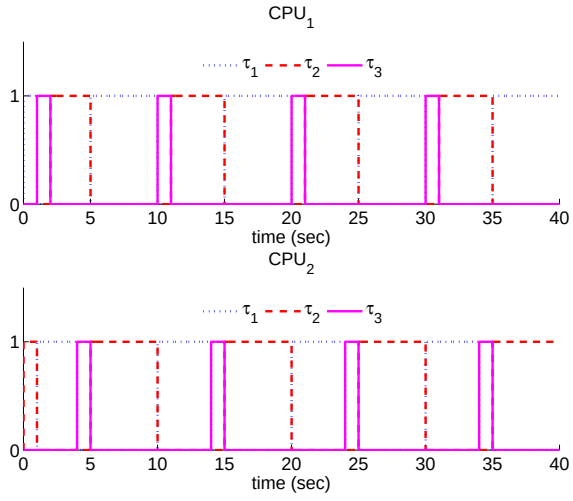

Fig. 5. Discretization of the *fluid* schedule for CPU_1 and CPU_2.

In Algorithm 1 $SD = \{10, 20, 30, 40\}$ and $Q = 1$. The computed discrete schedule is depicted in Fig. 5. Note that at every $sd \in SD$ the accumulated tokens in $m_i^{alloc}(sd)$ meets the fluid schedule requirements. For instance at $sd = 10$, the fluid schedule indicates that task τ_1 requires 9 time units. Analyzing Fig. 5, τ_1 is executed 1 time units in CPU_1 from $\zeta = 0$ to $\zeta = 1$ and 3 time units in CPU_2 from $\zeta = 1$ to $\zeta = 4$ and finally 5 time units in CPU_1 from $\zeta = 5$ to $\zeta = 10$, thus τ_1 is executed 9 time units at $sd = 10$.

VIII. Conclusions

We propose a *fluid* scheduler based on a sliding mode control technique, designed to easily integrate thermal restrictions in the near future. The scheduler is derived from a $TCPN$ model of tasks and $CPUs$, which constitutes itself a modeling methodology for fluid schedulers in real-time systems. Moreover, we present a discrete implementation of the fluid scheduler, which allows reducing task switching and migration. The scheduler here presented shares with P-fair algorithms their ability to exploit CPU time. Furthermore, the scheduler keeps the number of context switches and migrations reasonably low, by solving the control equation only upon the jobs' deadlines, profiting from the qualities of deadline partitioning algorithms. Future work will address the design of fine thermal-aware schedulers using this model.

References

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

[2] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok, "Real time scheduling theory: A historical perspective," *Real-time systems*, 2004.

[3] S. K. Dhall and C. Liu, "On a real-time scheduling problem," *Operations research*, vol. 26, no. 1, pp. 127–140, 1978.

[4] M. L. Dertouzos and A. K.-L. Mok, "Multiprocessor online scheduling of hard-real-time tasks," *Software Engineering, IEEE Transactions on*, vol. 15, no. 12, pp. 1497–1506, 1989.

[5] A. Chandra, M. Adler, and P. Shenoy, "Deadline fair scheduling: bridging the theory and practice of proportionate pair scheduling in multiprocessor systems," in *Real-Time Technology and Applications Symposium, 2001. Proceedings Seventh IEEE*, 2001, pp. 3–14.

[6] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, 1996.

[7] S. K. Baruah, J. E. Gehrke, and C. G. Plaxton, "Fast scheduling of periodic tasks on multiple resources," in *ipps*. IEEE, 1995, p. 280.

[8] J. H. Anderson and A. Srinivasan, "Mixed pfair/erfair scheduling of asynchronous periodic tasks," in *Real-Time Systems, 13th Euromicro Conference on, 2001*. IEEE, 2001, pp. 76–85.

[9] D. Zhu, D. Mossé, and R. Melhem, "Multiple-resource periodic scheduling problem: how much fairness is necessary?" in *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, 2003, pp. 142–151.

[10] V. Utkin, J. Guldner, and J. Shi, *Sliding mode control in electro-mechanical systems*. CRC press, 2009, vol. 34.

[11] G. Desirena-Lopez, C. R. Vázquez, A. Ramírez-Treviño, and D. Gómez-Gutiérrez, "Thermal modelling for temperature control in MPSoC's using fluid Petri nets," in *IEEE Conference on Control Applications part of Multi-conference on Systems and Control*, 2014.

[12] J. Desel and J. Esparza, *Free Choice Petri Nets*. Cambridge Tracts in Theoretical Computer Science 40, 1995.

[13] M. Silva and L. Recalde, "Redes de Petri continuas: Expresividad, análisis y control de una clase de sistemas lineales conmutados," *Revista Iberoamericana de Automática e informática Industrial*, julio 2007.

[14] H. K. Khalil and J. Grizzle, *Nonlinear systems*. Prentice hall New Jersey, 1996, vol. 3.