An Optimal Multiprocessor Scheduling Algorithm without Fairness

Geoffrey Nelissen[1], Vandy Berten, Joël Goossens, Dragomir Milojevic
Parallel Architectures for Real-Time Systems (PARTS) Research Center
Université Libre de Bruxelles
Brussels, Belgium

***Abstract*—All known scheduling algorithms that optimally schedule task sets on multiprocessor platforms, are partially or completely based on the notion of *proportionate fairness* introduced by Baruah *et al.* in 1993 [1]. One could therefore think that there is no other solution to guarantee the optimality than to use the proportionate fairness property. We want to prove the opposite and we propose an alternative. In this paper we present an algorithm which is a multiprocessor generalization of EDF and where tasks do not have to follow any fairness property. We conjecture that this algorithm optimally schedules any set of periodic tasks with implicit deadlines and could easily be extended to the schedule of sporadic tasks.**

I. Introduction

Optimal scheduling algorithms for uniprocessor platforms exist from many years [2]. These are generally based on simple priority definitions as in Earliest Deadline First (EDF). However, their generalization to the scheduling on multiprocessor platforms always leaded to the loss of optimality. A new approach named *proportionate fairness* was therefore proposed by Baruah *et al.* in 1993 [1]. In a *Proportionate Fair* (PFair) algorithm, the time is divided in quanta and each task τ_i is scheduled such that, after any quantum q, the amount of quanta executed by τ_i from the start of the schedule to q has been proportionate to its utilization factor. Unfortunately, the optimality of this new class of algorithms is at the cost of numerous preemptions, migrations and scheduling points during the system execution. These drawbacks are partially overcome when applying the *Deadline Partitioning Fairness* (DP-Fair) theory [3], [4]. In this case, the property of fairness has indeed to be ensured only at the deadlines of the jobs executed in the application and not anymore after each quantum of time.

In [5], the authors proposed an hybrid solution named EKG for the schedule of periodic tasks. In its optimal version, the tasks are grouped in what we will call *supertasks*. All supertasks are scheduled under a DP-Fair policy. Whenever a supertask is chosen to be executed, one of its *component tasks* is selected accordingly to the EDF algorithm to be effectively executed on the platform. The technique used in EKG allows to drastically reduce the amount of preemptions during the execution. However, when extended to the schedule of sporadic tasks with implicit deadlines, EKG [6] and its evolution NPS-F [7] have to make a trade-off between the upper-bound on the total utilization of the platform and the amount of preemptions. Therefore, the optimality cannot be guaranteed unless we accept to have an amount of preemptions and scheduling points even greater than what we get with a PFair algorithm.

[1]Supported by the Belgian National Science Foundation (F.N.R.S.) under a F.R.I.A. grant.

Goal of this work: We conjecture that the the fairness property needed by DP-Fair, EKG (and successors) is not mandatory for optimality. We aim to propose an optimal multiprocessor scheduling algorithm based on priorities and relaxing the fairness property.

II. Model

We first tackle the problem of scheduling a set of n independent strictly periodic tasks with implicit deadlines on a platform composed of m identical processors. Each task τ_i has a worst case execution time C_i and a period T_i. That is, each job of τ_i must receive C_i time units before the next job arrival and there are T_i time units between two such job arrivals. We define the utilization factor of τ_i as the quantity $U_i \stackrel{\text{def}}{=} \frac{C_i}{T_i}$. At any instant t, we define the remaining execution time $r_i(t)$ of τ_i as the amount of time units that the current job of τ_i has still to execute before its next absolute deadline $d_i(t)$.

The system state $s(t)$ at time t is completely defined by the remaining execution times $r_i(t)$ and the absolute deadlines $d_i(t)$ of all tasks τ_i in the system.

For a better readability, we will use the notations r_i and d_i instead of $r_i(t)$ and $d_i(t)$ in the remaining of this paper.

III. Algorithm Description

In the following, we assume that for all pairs of tasks τ_k and τ_ℓ, if $k < \ell$ then $d_k \leq d_\ell$ (i.e. the deadline of the current job of τ_k is earlier than the deadline of the job of τ_ℓ).

Our algorithm divides the time in *time slices* extending from one job arrival to the next one. Whenever a job arrives in the system, we execute the two following phases:

Phase 1: The algorithm divides the r_i time units of every task τ_i among the processors. We define $q_{i,j}$ as the amount of work of τ_i that we assign to processor π_j in the interval $[t, d_i]$ (t is the instant of the new job arrival). The $q_{i,j}$ values are determined such that (i) $\sum_j q_{i,j} = r_i$ and (ii) the r_i time units of τ_i can be executed before d_i without intra-job parallelism.

$TaskList$:= list of n tasks sorted by increasing absolute deadlines
t := current time
for all $\tau_i \in TaskList$ **do**
  Temp := r_i
  for $j := 1$ **to** m **do**
    $\rho_{i,j} := \rho_{i-1,j} + q_{i-1,j} + \left[U^i - (j-1)\right]_0^1 (d_i - d_{i-1})$
    $q_{i,j}^{\max} := (d_i - t) - \rho_{i,j} - \sum_{\ell<j} q_{i,\ell}$
    /* Assignment on π_j */
    if $q_{i,j}^{\max} \geq$ Temp **then**
      $q_{i,j}$:= Temp
      break
    else
      $q_{i,j} := q_{i,j}^{\max}$
      Temp := Temp $- q_{i,j}^{\max}$
    end if
  end for
end for

Fig. 1. Assignment pseudo-algorithm.

The tasks are assigned in an *increasing absolute deadline order*. For each task τ_i and each processor π_j, we compute the amount of time units that τ_i can execute within the interval $[t, d_i]$ without parallelism.

Fig. 1 proposes a pseudo-code of the assignment protocol and introduces the two following quantities:

- $\rho_{i,j}$: the amount of time which is already reserved on processor π_j for the execution of the tasks $\{\tau_1, ..., \tau_{i-1}\}$ (i.e. tasks with a deadline at or before d_i).
- $q_{i,j}^{\max}$: the maximum amount of work that τ_i could execute on π_j in the interval $[t, d_i]$ without parallelism.

Explanations on the computation of both these quantities will be given in Section III-A.

When the task τ_i is assigned, we always try to fill up the processors with the smallest indexes. Therefore, we first compute the maximum amount of work $q_{i,1}^{\max}$ that the task τ_i could execute on processor π_1. If $q_{i,1}^{\max}$ is larger than the remaining execution time r_i of τ_i, we can assign all the r_i time units on π_1. That is, $q_{i,1} := r_i$. On the other hand, if $r_i > q_{i,1}^{\max}$ then we cannot assign more than $q_{i,1}^{\max}$ time units of τ_i on processor π_1 (otherwise, from the definition of $q_{i,j}^{\max}$, there is a risk that τ_i does not respect its deadline d_i without parallelism). Therefore, we associate the maximum amount of work of τ_i with π_1 (i.e. $q_{i,1} := q_{i,1}^{\max}$) and we recursively apply the same procedure on processors π_2 to π_m until all the r_i time units of τ_i are dispatched on the processors of the platform.

Phase 2: We schedule the $q_{i,j}$ time units according to *partitioned EDF*. Furthermore, we add the following rule: if a task τ_i is split among several processors (i.e. there exist at least two processors π_k and π_ℓ such that $k < \ell$ and $q_{i,k} > 0$ and $q_{i,\ell} > 0$) then τ_i cannot be executed on the higher indexed processor π_ℓ when executed on the lower indexed one π_k. This mechanism does not affect the schedulability of the task in the future if the $q_{i,j}^{\max}$ values were correctly computed (see Section III-A).

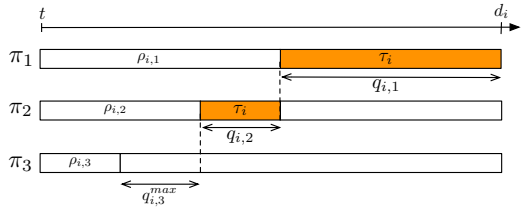

Fig. 2. Computation of $q_{i,j}^{\max}$.

When the first absolute deadline (say d_1) is reached (i.e. a new job arrives in the system), we update the system state $s(t)$ and we repeat the phases 1 and 2 for the next time slice extending from d_1 to the next absolute deadline in the system.

A. Computation Details

We define the maximum amount of work $q_{i,j}^{\max}$ that a task τ_i can execute on a processor π_j without parallelism as

$$q_{i,j}^{\max} \stackrel{\text{def}}{=} (d_i - t) - \rho_{i,j} - \sum_{\ell<j} q_{i,\ell} \quad (1)$$

Indeed, we cannot allocate more than $(d_i - t)$ time units on one processor within the interval $[t, d_i]$. Therefore, if there already are $\rho_{i,j}$ time units reserved on π_j before the assignment of τ_i then we can give $(d_i - t) - \rho_{i,j}$ time units at most to τ_i (i.e. $q_{i,j}^{\max} \leq (d_i - t) - \rho_{i,j}$). Moreover, accordingly to the algorithm presented in the previous section, τ_i could have already received some time $q_{i,\ell}$ on processors with lower indexes (i.e. $\ell < j$). The value of $q_{i,j}^{\max}$ is minimal when these $q_{i,\ell}$ time units are scheduled as in the example depicted on Fig. 2. Since intra-task parallelism is not allowed, the maximum amount of work that τ_i could execute on π_j under this situation is given by Eq. 1.

To compute $q_{i,j}^{\max}$, we therefore need to know the amount of time $\rho_{i,j}$ which is already reserved on this processor for the execution of the tasks $\{\tau_1, ..., \tau_{i-1}\}$.

To compute the value of $\rho_{i,j}$, we assume that for each task τ_k already assigned on the platform:

1) we reserved r_k time units before its first absolute deadline d_k according to the algorithm presented in the previous section.
2) we reserved a proportion of time equal to U_k after the deadline d_k such that all future jobs of τ_k will be able to be executed on the platform. This technique does not differ from the PFair approach [1]. However, since the assignment protocol will be reapplied after the first job deadline d_1 reached in the system, this reserved time will never be really scheduled. But, the anticipation of this demand allows to keep a feasible system after d_1.

Therefore, the amount of time reserved for τ_1 between t and d_i ($i > 1$) is equal to

$$r_1 + U_1(d_i - d_1)$$

Applying this argument to all tasks τ_k such that $k < i$, we get that the amount of processor time ρ_i reserved for their execution on the platform is equal to

$$\rho_i \stackrel{\text{def}}{=} \sum_{k=1}^{i-1} (r_k + U_k(d_i - d_k)) \tag{2}$$

Eq. 2 can be rewritten recursively:

$$\begin{cases} \rho_1 = 0 \\ \rho_i = \rho_{i-1} + r_{i-1} + \sum_{k=1}^{i-1} U_k(d_i - d_{i-1}) \end{cases} \tag{3}$$

Indeed, the amount of time reserved for $\{\tau_1, ..., \tau_{i-1}\}$ within $[t, d_i]$ is equal to the time reserved for $\{\tau_1, ..., \tau_{i-2}\}$ within $[t, d_{i-1}]$ (i.e. ρ_{i-1}) increased by the time reserved for τ_{i-1} before d_{i-1} (i.e. r_{i-1}) and the amount of time reserved for all tasks $\{\tau_1, ..., \tau_{i-1}\}$ in the interval $[d_{i-1}, d_i]$ (i.e. $\sum_{k=1}^{i-1} U_k(d_i - d_{i-1})$).

Eq. 3 gives the amount of time already reserved for tasks $\{\tau_1, ..., \tau_{i-1}\}$ within $[t, d_i]$ on *all* the processors of the platform. To compute the amount of time reserved on *one* specific processor π_j, we use the two following properties:

- According to the algorithm previously presented, the amount of time assigned for the execution of τ_k within $[t, d_k]$ on π_j is equal to $q_{k,j}$ instead of r_k.
- Only $(d_i - d_{i-1})$ time units can be reserved on one processor in the interval $[d_{i-1}, d_i]$. Thereby, if for instance $(d_i - d_{i-1}) = 2$ and the term $\sum_{k=1}^{i-1} U_k(d_i - d_{i-1}) = 5$ then we assume that two time units are reserved on π_1, the two next time units are reserved on π_2 and the last time unit is reserved on π_3. Therefore, the amount of time already reserved on processor π_j in the interval $[d_{i-1}, d_i]$ is equal to $\left[\sum_{k=1}^{i-1} U_k - (j-1)\right]_0^1 (d_i - d_{i-1})$ where $[x]_a^b \stackrel{\text{def}}{=} \max(a, \min(b, x))$.

Using these two properties, we get that the amount of time $\rho_{i,j}$ already reserved on each processor π_j before the assignment of τ_i is given by

$$\rho_{i,j} = \rho_{i-1,j} + q_{i-1,j} + \left[\sum_{k=1}^{i-1} U_k - (j-1)\right]_0^1 (d_i - d_{i-1})$$

and using the notation $U^i \stackrel{\text{def}}{=} \sum_{k=1}^{i-1} U_k$ it yields

$$\begin{cases} \rho_{1,j} \stackrel{\text{def}}{=} 0 \\ \rho_{i,j} \stackrel{\text{def}}{=} \rho_{i-1,j} + q_{i-1,j} + \left[U^i - (j-1)\right]_0^1 (d_i - d_{i-1}) \end{cases}$$

B. Example

Fig. 3 shows an example of the assignment of three tasks on two processors following the algorithm presented in Fig. 1. The parameters of the tasks are as follow: $d_1 = t + 10$, $d_2 = t + 30$, $d_3 = t + 42$, $r_1 = 5$, $r_2 = 15$, $r_3 = 26$, $U_1 = 0.3$, $U_2 = 0.8$ and $U_3 = 0.35$. We first assign the task τ_1 since it has the smallest deadline. By definition of $\rho_{i,j}$, we get that $\rho_{1,1} = 0$ (i.e. there is still no task assigned on the processor)

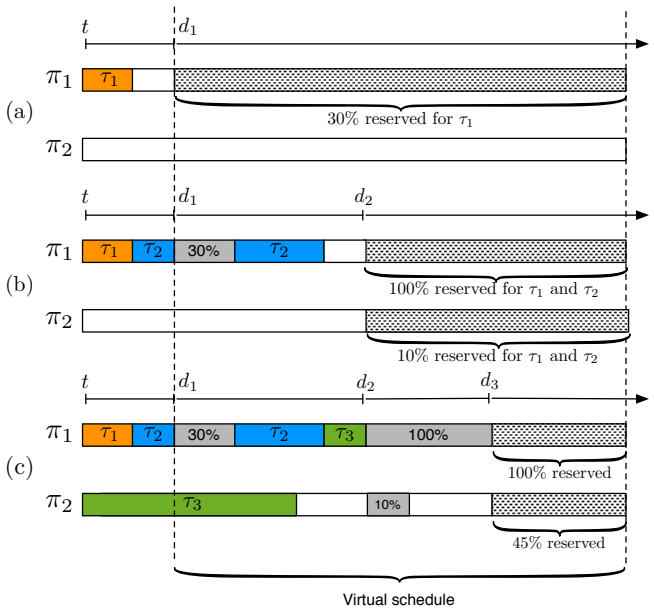

Fig. 3. Tasks assignment example.

implying that $q_{1,1}^{\max} = (d_1 - t) = 10$. We thereby assign all the remaining execution time r_1 of τ_1 to π_1 (i.e. $q_{1,1} = 5$). Moreover, we reserve 30% of time after d_1 to execute τ_1 in the future (i.e. a proportion of time equal to the utilization factor of τ_1) (see Fig. 3(a)). The second task that we must assign is τ_2. Since its deadline d_2 is at time $t + 30$, we have to execute the remaining execution time of τ_2 in the interval $[t, t+30]$. We therefore compute the maximum execution time $q_{2,1}^{\max}$ that τ_2 could execute on π_1 in this time interval. Since τ_1 is executed during $r_1 = 5$ time units between t and d_1 and since 30% of time is reserved to execute τ_1 after d_1, we get that $\rho_{2,1} = r_1 + U_1(d_2 - d_1) = 11$ and therefore $q_{2,1}^{\max} = (d_2 - t) - \rho_{2,1} = 19$. Since $r_2 = 15$, we assign the r_2 time units to the processor π_1 (i.e. $q_{2,1} = 15$). Furthermore, we reserve 80% of time to execute τ_2 after d_2. The total proportion of time reserved on the platform after d_2 is thereby equal to $U_1 + U_2 = 110\%$. Since we cannot reserve more than 100% of one processor, we reserve 100% of π_1 and 10% of π_2 (see Fig. 3(b)). We obtain that $\rho_{3,1} = \rho_{2,1} + q_{2,1} + 1 \cdot (d_3 - d_2) = 38$ and $\rho_{3,2} = \rho_{2,2} + q_{2,2} + 0.1(d_3 - d_2) = 1.2$. It therefore yields $q_{3,1}^{\max} = (d_3 - t) - \rho_{3,1} = 4$ leading to the split of τ_3 among π_1 and π_2. We finally get that $q_{3,1} = q_{3,1}^{\max} = 4$ and $q_{3,2} = 22$ (see Fig. 3(c)).

Note that the assignment protocol will be re-executed after each job arrival. Therefore, the *virtual schedule* build after d_1 (i.e. the first deadline and thereby the first job arrival in the system) on Fig. 3 will never be executed but is useful to keep a feasible task system after d_1.

IV. Algorithm Properties

In this section we will present some interesting properties and conjectures about our algorithm.

A. Time Complexity

Property 1: The time complexity of our algorithm is $O(n \cdot m)$.

Proof: To obtain the time complexity of the complete algorithm, we will compute the complexity of the two phases presented in Section III.

- Phase 1: During the assignment process, the construction of the sorted list (named $TaskList$ in Fig. 1) can be implemented with a complexity of $O(n)$. Indeed if we use a list of the tasks pre-sorted according to their periods, the reconstruction of $TaskList$ after each deadline consists in the merge of two sorted lists. This can be achieved with a linear complexity [8]. Moreover, the assignment of the n tasks on the m processors is performed using two nested loops. The first one has n iterations and the second one has m iterations. All the operations realized in the loops can be done in $O(1)$. Therefore, the time complexity of the assignment procedure is $O(n \cdot m)$.
- Phase 2: After the assignment procedure, the execution of EDF on each processor only needs to manipulate a ready queue. This can be done with a complexity of $O(\log n)$.

The overall complexity is therefore $O(n \cdot m)$. ■

B. Optimality

Conjecture 1: Our algorithm optimally schedules any feasible set of periodic tasks with implicit deadlines such that $\sum_{i=1}^{n} U_i \leq m$.

The idea behind this conjecture is that our algorithm proposes a valid schedule in the first time slice extending from time $t = 0$ to the first deadline d_1 in the system (the proof is not presented here due to space limitation). Moreover, the task system is still feasible after d_1. Then, the algorithm computes a new repartition of the jobs after d_1 such that partitioned EDF can be used without parallelism in the second time slice (through the computation of $q_{i,j}^{\max}$) and such that the system of tasks stays feasible after the second deadline d_2 in the system (through the computation of $\rho_{i,j}$ and $q_{i,j}^{\max}$). Repeating this step after each job arrival (i.e. each deadline in the system), we get that our algorithm is optimal for the schedule of periodic tasks with implicit deadlines.

C. Strict Multiprocessor EDF Generalization

Our algorithm is a strict multiprocessor EDF generalization in the sense that applied on a uniprocessor platform, it behaves exactly like EDF. We therefore get the following property:

Property 2: Our conjectured optimal algorithm is a strict multiprocessor EDF generalization.

Proof: If there is only one processor available in the platform, at each job arrival in the system, our algorithm will always assign all tasks on this processor (phase 1 of the algorithm). Indeed, the available capacity on the processor is always smaller than or equal to the tasks demand if the task system is feasible. Therefore, all tasks running on the platform are scheduled according to EDF between all job arrivals (phase 2 of the algorithm). ■

It results that our algorithm imposes at most one preemption at each job arrival when used to schedule a task set on one processor. In comparison, with some DPFair algorithms such as DP-Wrap [3] or LLREF [9], there are at least n preemptions between two job arrivals. However, we do not say that this property extends when our algorithm is used on more than one processor. Indeed, the upper bound on the amount of preemptions on multiprocessor platforms has still to be studied.

V. Conclusion and Future Works

In this paper, we presented a new multiprocessor scheduling algorithm. We showed that this algorithm is a strict multiprocessor EDF generalization and we conjecture that it optimally schedules periodic tasks with implicit deadlines. Furthermore, to the best of our knowledge, it is the first multiprocessor scheduling algorithm that does not use the fairness property to reach the optimality. With this new approach, we open the way to a new family of optimal algorithms with new tools to impact their performances.

However, some research has still to be carried out to reach our objectives. Therefore, our future works include:

1) formally prove the optimality of our algorithm.
2) extend this algorithm and its optimality proof to the schedule of set of *sporadic* tasks with implicit deadlines.
3) quantify the amount of preemptions during the execution when our algorithm is executed. Furthermore, it would be interesting to compare this quantity with the results obtained with EKG, NPS-F and some DP-Fair algorithms.

References

[1] 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.

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

[3] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, "Dp-fair: A simple model for understanding optimal multiprocessor scheduling," in *Proceedings of the 22nd Euromicro Conference on Real-Time Systems*. IEEE Computer Society, July 2010.

[4] D. Zhu, D. Mossé, and R. Melhem, "Multiple-resource periodic scheduling problem: how much fairness is necessary?" in *Proceedings of the 24th IEEE International Real-Time Systems Symposium*. IEEE Computer Society, 2003, p. 142.

[5] B. Andersson and E. Tovar, "Multiprocessor scheduling with few preemptions," *International Workshop on Real-Time Computing Systems and Applications*, pp. 322–334, 2006.

[6] B. Andersson and K. Bletsas, "Sporadic multiprocessor scheduling with few preemptions," in *Proceedings of the 2008 Euromicro Conference on Real-Time Systems*. IEEE Computer Society, 2008, pp. 243–252.

[7] K. Bletsas and B. Andersson, "Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound," in *Proceedings of the 2009 30th IEEE Real-Time Systems Symposium*. IEEE Computer Society, 2009, pp. 447–456.

[8] D. E. Knuth, *Art of Computer Programming, Volume 3: Sorting and Searching (2nd Edition)*. Addison-Wesley Professional, 1998.

[9] H. Cho, B. Ravindran, and E. D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," in *Proceedings of the 27th IEEE International Real-Time Systems Symposium*. IEEE Computer Society, 2006, pp. 101–110.