Scheduling communication requests traversing a switch : complexity and algorithms

Matthieu Gallet, Yves Robert, Frédéric Vivien

Scheduling communication requests traversing a switch: complexity and algorithms

Matthieu Gallet,
Yves Robert,
Frédéric Vivien

June 2006

Research Report N^o 2006-25

Scheduling communication requests traversing a switch: complexity and algorithms

Matthieu Gallet, Yves Robert, Frédéric Vivien

June 2006

Abstract

In this paper, we study the problem of scheduling file transfers through a switch. This problem is at the heart of a model often used for large grid computations, where the switch represents the core of the network interconnecting the various clusters that compose the grid. We establish several complexity results, and we introduce and analyze various algorithms, from both a theoretical and a practical perspective.

Keywords: Grid computing, networks, scheduling file transfers, switch, complexity.

Résumé

Dans ce rapport, nous étudions l'ordonnancement de l'envoi de fichiers à travers un switch. Ce problème est souvent utilisé pour modéliser le réseau utilisé par une grille de calcul, où le switch représente le coeur du réseau qui relie les différents clusters composant la grille. Nous établissons différents résultats de complexité, avant d'étudier plusieurs algorithmes, tant d'un point de vue théorique que d'un point de vue pratique.

Mots-clés: Ordonnancement de transfert de fichiers, grille de calcul, réseaux, complexité, commutateur.

1 Introduction

Computation grids are more and more used to follow the tremendous growth of the need in computation power. Since computation nodes can be anywhere in the world, there are many issues involving the scheduling of both communications and computations. Even when communication links are dedicated, the problems of maximizing the number of files that can be transmitted, and of allocating the correct bandwidth to each file, turn out to be very difficult.

In this paper, we study the problem of scheduling file transmissions through a classical network switch. This problem could appear somewhat simplistic, but it is often used as a model for more general instances, where the switch is an Internet backbone, and where the different links to and from the switch are the bottlenecks; this model has already been studied in several papers [19, 20, 7, 4].

The bandwidth allocation is an important problem in distributed computating, and constraints are quite different from those applying to the general Internet. Indeed, for a standard DSL line (2Mb/s), the bottleneck is the line itself, and the link between the provider and the backbone (often rated at 2.5Gb/S) is sufficient. On the contrary, grid sites have links which can be rated at 2.5Gb/s, and large bulk transfers between sites can take up to several days. Moreover, performance quickly decreases in case of congestion, since many packets can be dropped by the TCP/IP protocol.

From the user point of view, any aborted transfer is a useless waste of (potentially scarce) computational and storage resources, which are reserved for a finite duration. Thus, we have to choose which requests have to be accepted to ensure the best possible usage of the network. But, even if constraints are stronger for a computation grid than for the Internet, a better scheduling can be considered when all transfers can be forecast.

Like many scheduling problems, most problems in this model are $\mathcal{NP}$-complete, and we have to search for heuristics and approximation algorithms. The idea of reservation was already studied, by example by L. Marchal et al [19], whose model consists of some ingress or egress links interconnected over a well-provisioned WAN. Requests (i.e., files to send from an ingress link to an egress link) have to be chosen and scheduled. During the transfer of a file, the allocated bandwidth remains constant. However, there is no practical reason to enforce this constraint, and we should allow the bandwidth to change several times during the transfer, so as to give more flexibility to the scheduling algorithms.

The rest of the paper is organized as follows. In section 2, the model and notations are detailed. Then we outline some interesting properties in Section 3. Complexity results are the core of Section 4. Algorithms and heuristics are given in Section 5, and experimental results are provided in Section 6. Finally, we give some conclusions in Section 7.

2 Model and problem definition

We describe here the model and notations used in this work. Basically, our goal is to send some files (or requests) from ingress links (or sources) to egress links (or destinations) through a central switch, in order to maximize one of the two studied objective functions.

- PLATFORM:
  We consider a switch, with a capacity C_{tot}, linked to p ingress links and p' egress links. The j-th ingress link has a capacity C_j (with $1 \leq j \leq p$), and the j-th egress link has a capacity C'_j (with $1 \leq j \leq p'$).

- REQUESTS:
  We have a set of n requests to schedule. The i-th request, $1 \leq i \leq n$, arrives in the system a time $r_i \in \mathbb{Q}^+$ (its release date), it should be completed before time $d_i \in \mathbb{Q}^+, d_i > r_i$ (its deadline). It has to be sent from the ingress link $\mathrm{src}(i) \in \{1, \dots, p\}$ to the egress link $\mathrm{dest}(i) \in \{1, \dots, p'\}$. This request has a size (or surface) $S_i \in \mathbb{Q}^+$ and a weight $w_i \in \mathbb{Q}^+$.

- CONSTRAINTS:
  A schedule has to respect some constraints to be valid:

  - a request is either processed, or discarded: $\forall i \in \{1,\dots,n\}, x_i \in \{0,1\}$. Any valid schedule has to choose if the request i is processed ($x_i = 1$) or not ($x_i = 0$), and then allocates an instantaneous bandwidth $b_i : \mathbb{R}^+ \to \mathbb{Q}^+$. Some requests may not be scheduled at all, while some other requests may only be partially completed (in which case we will consider that they were not processed at all).
  - b_i is a function, which is integrable over $\mathbb{R}^+$,
  - bandwidth functions are non-negative functions: $\forall t \geq 0, \forall i \in \{1,\dots,n\}, b_i(t) \geq 0$,
  - deadlines are strictly enforced: $\forall t \geq d_i, \forall i \in \{1,\dots,n\}, b_i(t) = 0$,
  - a request cannot be processed before its release date: $\forall t < r_i, \forall i \in \{1,\dots,n\}, b_i(t) = 0$,
  - we cannot exceed the capacity of an ingress link: $\forall t \geq 0, \forall j \in \{1,\dots,p\}, \sum_{i,\mathrm{src}(i)=j} b_i(t) \leq C_j$,
  - we cannot exceed the capacity of an egress link: $\forall t \geq 0, \forall j' \in \{1,\dots,p'\}, \sum_{i,\mathrm{dest}(i)=j'} b_i(t) \leq C'_{j'}$,
  - we cannot exceed the switch capacity: $\forall t \geq 0, \sum_{i=1}^{n} b_i(t) \leq C_{tot}$,
  - any chosen request has to be entirely processed:

    $$\forall i \in \{1,\dots,n\}, x_i \int_0^{+\infty} b_i(t)dt = x_i S_i.$$

    This formulation allows a request to begin without being finished. Such a scenario has no interest in off-line scheduling strategies, but should be considered for online algorithms, where a new request can be preferred to one currently being processed. Note that if we don't want to allow such scenarios, we simply write

    $$\int_0^{+\infty} b_i(t)dt = x_i S_i.$$

- OBJECTIVE FUNCTIONS:
  Two different objective functions are studied:

  - the number of requests that are processed:

    $$\sum_{i=1}^{n} x_i;$$

  - the profit generated by processed requests:

    $$\sum_{i=1}^{n} x_i w_i.$$

3 Some problem properties

In this section, we prove some simplifying lemmas, which allow to focus on certain types of schedules, thereby reducing the solution space.

3.1 We can use only step functions for the bandwidth allocation functions.

Sofar, we have not specified any constraint on the form of the b_i functions (except their integrability). Now, we show that we can suppose, without any loss of generality, that the b_is are step functions, with a small number of steps.

Lemma 1. *We consider a platform with a capacity C_{tot}, p ingress links (with respective capacities C_j) and p' egress links (with respective capacities C'_j). Let $(b_i)_{1\leq i\leq n}$ be any schedule. Then we can build a schedule $(b'_i)_{1\leq i\leq n}$, which realizes the same objective, and where the b'_i are step functions with at most $2n$ steps: the bandwidths only change when a request joins the system (release date) or when one is completed.*

Proof. We want to show that if we have any schedule $(b_i)_{1\leq i\leq n}$ (cf. Figure 1), then we can build a schedule $(b'_i)_{1\leq i\leq n}$, having the same objective, but only using particular step functions (cf. Figure 2).

We note $\{t_1, \ldots, t_q\}$ the set of release dates and deadlines $\{r_1, \ldots, r_n\} \cup \{d_1, \ldots, d_n\}$ with $t_1 < t_2 < \ldots < t_{q-1} < t_q$.

Now we show that we can build a schedule, such that $\forall i \in \{1, \ldots, n\}, \forall k \in \{2, \ldots, q\}$, $b_i(t) = constant$ over $[t_{k-1}, t_k[$.
We set b'_i over $[t_{k-1}, t_k[$ to the average value of b_i over this same interval. By definition, b'_i is a step function, with at most $q \leq 2n-1$ steps.
Formally, let us consider any valid schedule $\mathcal{O}$ $((b_i)_i,(x_i)_i)$. It respects the following constraints:

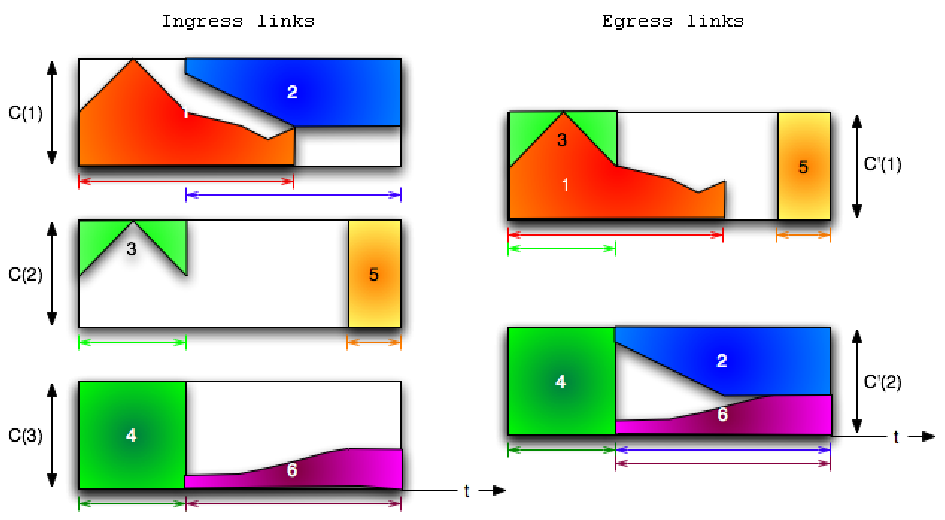
- $\forall t \geq 0, \forall i \in [1, n], b_i(t) \geq 0$;
- $\forall t \geq d_i, \forall i \in [1, n], b_i(t) = 0$;
- $\forall t < r_i, \forall i \in [1, n], b_i(t) = 0$;
- $\forall j \in \{1, \ldots, p\}, \forall t \geq 0, \sum_{i\in\{1,\ldots,n\},\mathrm{src}(i)=j} b_i(t) \leq C_j$;
- $\forall j' \in \{1, \ldots, p'\}, \forall t \geq 0, \sum_{i\in\{1,\ldots,n\},\mathrm{dest}(i)=j'} b_i(t) \leq C'_{j'}$;
- $\forall t \geq 0, \sum_{i\in\{1,\ldots,n\}} b_i(t) \leq C_{tot}$;
- $\forall i \in \{1, \ldots, n\}, x_i \int_{r_i}^{d_i} b_i(t)dt = x_i S_i$.

We build $\mathcal{O}'$ $((b'_i)_i,(x'_i)_i)$ using the following rules:

- $\forall i \in \{1, \ldots, n\}, x'_i = x_i$;
- $\forall k \in \{2, \ldots, q\}, \forall i \in \{1, \ldots, n\}, \forall t \in [t_{k-1}, t_k[, b'_i(t) = \dfrac{\int_{t_{k-1}}^{t_k} b_i(t)dt}{t_k - t_{k-1}}$.

Now, we will show that we have a valid schedule, with the same result as the previous schedule:

- $\forall i \in \{1, \ldots, n\}, \forall t \geq 0, b_i(t) \geq 0$ then $\forall k \in \{2, \ldots, q\}, \int_{t_{k-1}}^{t_k} b_i(t)dt \geq 0$, that show that $\forall t \leq 0, b'_i(t) \geq 0$;
- $\forall t > d_i, \forall i \in [1, n], b'_i(t) = 0$ since $b_i(t) = 0$,
- $\forall t \leq r_i, \forall i \in [1, n], b'_i(t) = 0$ since $b_i(t) = 0$,
- $\forall i \in \{1, \ldots, n\}$,



1. $\mathrm{src}(1) = 1$, $\mathrm{dest}(1) = 1$, $S_1 = 5$, $r_1 = 0$, $d_1 = 4$
2. $\mathrm{src}(2) = 1$, $\mathrm{dest}(2) = 2$, $S_2 = 4$, $r_2 = 2$, $d_2 = 6$
3. $\mathrm{src}(3) = 2$, $\mathrm{dest}(3) = 1$, $S_3 = 1$, $r_3 = 0$, $d_3 = 2$
4. $\mathrm{src}(4) = 3$, $\mathrm{dest}(4) = 2$, $S_4 = 4$, $r_4 = 0$, $d_4 = 2$
5. $\mathrm{src}(5) = 2$, $\mathrm{dest}(5) = 1$, $S_5 = 2$, $r_5 = 5$, $d_5 = 6$
6. $\mathrm{src}(6) = 3$, $\mathrm{dest}(6) = 2$, $S_6 = 2$, $r_6 = 2$, $d_6 = 4$

$C_j = C'_{j'} = 2$ with $j = 1, 2, 3$ et $j' = 1, 2$

Figure 1: A simple schedule example.

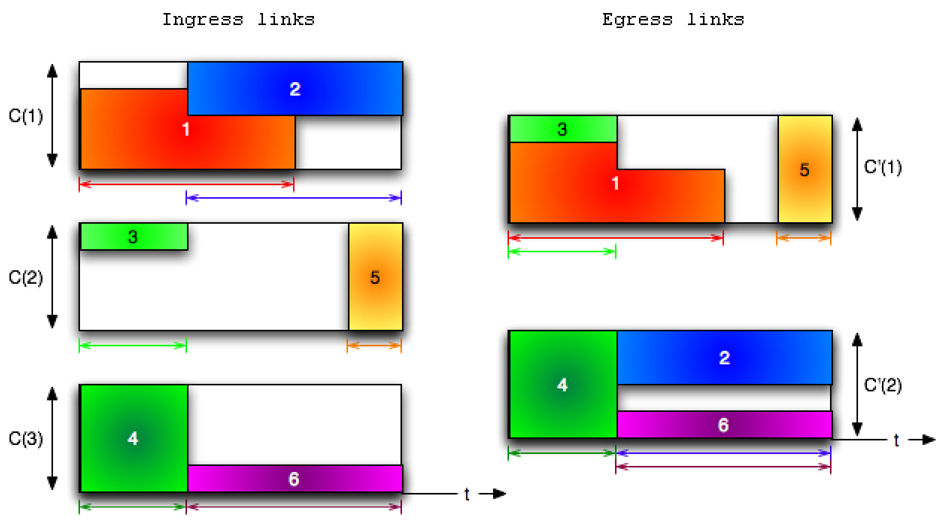


Figure 2: Schedule with step functions.

$$
\begin{aligned}
\int_0^{\infty} b'_i(t)dt &= \int_{t_1}^{t_q} b'_i(t)dt \\
&= \sum_{k=2}^{q} \left(\int_{t_{k-1}}^{t_k} b'_i(t)dt \right) \\
&= \sum_{k=2}^{q} \left(\int_{t_{k-1}}^{t_k} \frac{\int_{t_{k-1}}^{t_k} b_i(s)ds}{t_k - t_{k-1}} dt \right) \\
&= \sum_{k=2}^{q} \left(\int_{t_{k-1}}^{t_k} b_i(s)ds \right) \\
&= \int_{t_1}^{t_q} b_i(s)ds \\
&= \int_0^{+\infty} b_i(s)ds
\end{aligned}
$$

Each request is as processed in the new schedule as in the original schedule.

- $\forall i \in \{1, \ldots, n\}, x_i \int_{r_i}^{d_i} b'_i(t)dt = x_i S_i$
  According to the previous line, this property is true.

- Let $t \geq 0$ and $j \in \{1, \ldots, p\}$. If $t \geq t_q$, then $\forall i \in \{1, \ldots, n\}, b'_i(t) = 0$ and then $\sum_{S(j)} b'_i(t) \leq C_j$ is true.
  Let $S(j) = \{i \in \{1, \ldots, n\}, \mathrm{src}(i) = j\}$.
  If $t < t_q$, there exists $k \in \{2, \ldots, q\}$ such that $t_{k-1} \leq t < t_k$.

$$
\begin{aligned}
\sum_{S(j)} b'_i(t) &= \sum_{S(j)} \frac{1}{t_k - t_{k-1}} \int_{t_{k-1}}^{t_k} b_i(s)ds = \frac{1}{t_k - t_{k-1}} \int_{t_{k-1}}^{t_k} \sum_{S(j)} b_i(s)ds \\
&\leq \frac{1}{t_k - t_{k-1}} \int_{t_{k-1}}^{t_k} C_j ds \\
&\leq C_j.
\end{aligned}
$$

- $\forall j' \in \{1, \ldots, p'\}, \forall t \geq 0, \sum_{i \in \{1,\ldots,n\}, \mathrm{dest}(i)=j'} b'_i(t) \leq C'_{j'}$: we prove this property as we proved the previous one.

□

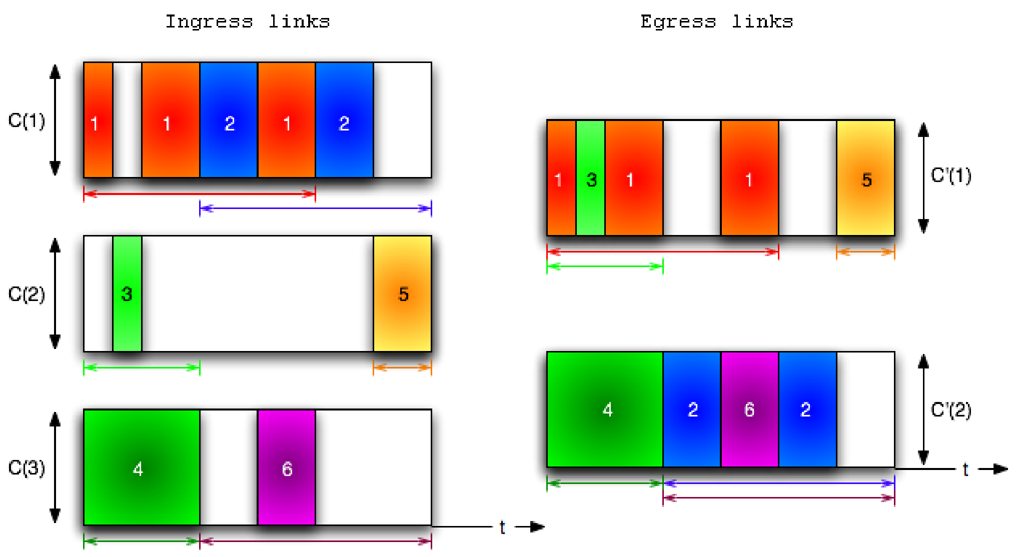


Figure 3: Schedule with $b_i(t) \in \{0, C\}$.

3.2 If all ingress and egress links have the same bandwidth, we can suppose that at any time at most one request is scheduled in a link.

Now, we want to specialize the b_i functions in another way: a bandwidth can only take two different values, 0 or C. So, only one request can be sent in a given link, at a given time.

Lemma 2. *Consider a platform with p ingress links and p' egress links, of same capacity C. Consider an unspecified schedule $(b_i, x_i)_{1\leq i\leq n}$. Then there exists a schedule $(b'_i)_{1\leq i\leq n}$ of same objective, such that, at any time t and for every request i, $b_i(t) \in \{0, C\}$.*

Proof. Let us consider a schedule $\mathcal{O} = (b_i, x_i)_{1\leq i\leq n}$ processing n requests (without loss of generality, we will assume that all requests are processed). Request i has a size $S_i = \frac{s_i}{q}$, where $q = \gcd_{1\leq i\leq n}(S_i)$.

First of all, we can suppose, without any loss of generality, that we have, for all i, b_i is constant over on $[t_1, t_2[$ (t_1 . So, we will work only on the interval $[t_1, t_2[$ and we will suppose that $S_i = \int_{t_1}^{t_2} b_i(t)dt$ (cf. Figure 3.1).

We want to show that we can exhibit a schedule $\mathcal{O}' = (b_i, x_i)_{1\leq i\leq n}$, such that, for all i, $b'_i(t)$ is in $\{0, C\}$ and having the same objective value as the original schedule $\mathcal{O}$.

Let us consider the following weighted bipartite graph:

- the first set of vertices, $V_1 = \{1, \dots, p\}$, represents the set of ingress links,
- the second one, $V_2 = \{1, \dots, p'\}$, represents the set of egress links,
- $E = \{e_1, \dots, e_n\}$ is the set of edges, defined by $e_i = (\mathrm{src(i)}, \mathrm{dest(i)})$,
- a weight $c : E \to \mathbb{Z}+$ defined by $c(e_i) = s_i$.

We know (Egerváry [22, vol. A chap. 20]) that we can find a family of matchings $(M_1, \dots, M_K)$ in a time $O(n^2)$, such that the edge e is in $c(e)$ distinct matchings. Moreover, we have $K =$

$\max_{v\in V_1\cup V_2} c(\delta(v))$ (we recall that $\delta(v) = \{e_i = (u, u') \in E$, s.t. $u' = v\}$). In our case, we have

$$K = \max\left(\max_{1\le j\le p} \sum_{i,\mathrm{src}(i)=j} s_i, \max_{1\le j'\le p'} \sum_{i,\mathrm{dest}(i)=j'} s_i\right).$$

With this family, we can define a schedule $\mathcal{O}'$ by: $\forall k, \forall i, \forall t \in [t_1 + (k-1)\frac{1}{qC}; t_1 + k\frac{1}{qC}[$,

$$b_i'(t) = \begin{cases} C & \text{if} \quad e_i \in M_k, \\ 0 & \text{oherwise} \end{cases}$$

We note $t_2' = t_1 + K\frac{1}{qC}$ the end of $\mathcal{O}'$.
This schedule is valid:

- constraints on capacity are respected, since at most one request is being sent in a link at any given time,
- $\forall i, \int_{t_1}^{t_2'} b_i'(t)dt = C\times\frac{1}{qC}c(e_i)$, since a request uses a bandwidth exactly equal to C during $c(e_i)$ intervals of a length $\frac{1}{qC}$. Since $C \times \frac{1}{qC}c(e_i) = \frac{s_i}{q} = S_i$, all requests are entirely processed;
- no request is sent before its release date;
- bandwidth are all non-negative;

Now, we show that deadlines are respected, i.e., $t_2' \le t_2 \Leftrightarrow t_2 - t_1 \ge K\frac{1}{qC}$ (since $\frac{K}{qC} = t_2' - t_1$).
By definition of K, we have $K = \max_{v\in V_1\cup V_2} c(\delta(v))$. Let j be the vertex (i.e., the egress or ingress link) which realizes this maximum. If we consider any request i which is sent through this link j, we have $b_i(t) = \frac{S_i}{t_2-t_1}$ (since b_i is constant and $\int_{t_1}^{t_2} b_i(t)dt = S_i$) and $\sum_i b_i(t) \le C$, we have $\sum_i S_i\frac{1}{C} \le t_2 - t_1$. Since $s_i = c(e_i)$, we have $(\sum_i c(e_i))\frac{1}{qC} \le t_2 - t_1$, which is exactly the wanted result as, by definition, $c(\delta(v)) = \sum_i c(e_i) = K$.

□

3.3 Choice of requests to schedule

Lemma 3. *Consider any platform and a given subset of n requests. Then we can determine, in polynomial time in n, p, and p' (where p is the number of ingress links and p' the number of egress links), whether there exists a schedule which can process all these requests. If such a schedule exists, we can find one in polynomial time too.*

Proof. Let $\{t_1, \dots, t_q\}$ be equal to $\bigcup_{i=1}^n \{r_i, d_i\}$, with $0 \le t_1 < \dots < t_q$ (we have $q \le 2n$). We define $t_0 = 0$ and $t_{q+1} = +\infty$. By using Lemma 3.1, we know that if there exists a schedule which completely processes these n requests, there exists a schedule such that every bandwidth function is constant over $[t_{k-1}; t_k[$ interval. We will exhibit a schedule which verifies this property.
Now we define:

- $b_{i,1}, \dots, b_{i,q}$ the q different values of each b_i function. We have $b_i(t) = b_{i,u}$ for $t \in [t_{u-1}, t_u[$.
- $\alpha_1, \dots, \alpha_q$ defined by $\alpha_u = t_u - t_{u-1}$.
- $x_{i,1}, \dots, x_{i,q}$ defined by $x_{i,u} = \begin{cases} 1 & \text{if} \quad r_i \le t_{u-1} < t_u \le d_i \\ 0 & \text{else} \end{cases}$
- $y_{i,1}, \dots, y_{i,p}$ defined by $y_{i,j} = \begin{cases} 1 & \text{if} \quad \mathrm{src}(i) = j \\ 0 & \text{else} \end{cases}$
- $y'_{i,1}, \dots, y'_{i,p'}$ defined by $y'_{i,j'} = \begin{cases} 1 & \text{if} \quad \mathrm{dest}(i) = j' \\ 0 & \text{else} \end{cases}$

Now, we write with these new notations the constraints, which are satisfied by any valid schedule:

- a request is processed or not: $x_i = 1$ since all requests are processed,
- since the b_i's are step functions, they are integrable over a finite time,
- bandwidth functions have to be non-negative: $\forall i, \forall u, b_{i,u} \geq 0$,
- we change the constraint "we cannot process a request after its deadline" by the constraint: "any work done after the deadline will not be taken into account",
- we change the constraint "we cannot process a request before its arrival" by the constraint: "any work done before the release date will not be taken into account",
- we cannot exceed the capacity of an ingress link: $\forall u, \forall j, \sum_{i=1}^{n} y_{i,j} b_{i,u} \leq C_j$,
- we cannot exceed the capacity of an egress link: $\forall u, \forall j', \sum_{i=1}^{n} y'_{i,j'} b_{i,u} \leq C'_{j'}$,
- we cannot exceed the capacity of the central switch: $\forall u, \sum_{i=1}^{n} b_{i,u} \leq C_{tot}$,
- every chosen request has to be completely processed: $\forall i, \sum_{u=1}^{q} x_{i,u}\alpha_{i,u} b_{i,u} = S_i$ (we recall that $x_{i,u}\alpha_{i,u}$ is constant).

So, we want to solve the following linear program:

$$\begin{cases} \forall i, \sum_{u=1}^{q} x_{i,u}\alpha_{i,u} b_{i,u} & = S_i \\ \forall i, \forall u, b_{i,u} & \geq 0 \\ \forall u, \sum_{i=1}^{n} b_{i,u} & \leq C_{tot} \\ \forall u, \forall j, \sum_{i=1}^{n} y_{i,j} b_{i,u} & \leq C_j \\ \forall u, \forall j', \sum_{i=1}^{n} y'_{i,j'} b_{i,u} & \leq C'_{j'} \end{cases}$$

which is a linear system with at most $2n^2$ variables and at most $(1+2n+2+2p+2p')n$ equations. We know how to solve such a problem in a time

$$O\left(n^7 \log\left(n + p + p'\right)\right)$$

(with the Karmarkar's algorithm [14]), since we are looking for a rational solution and not for an integer solution. So, we know how to build a schedule using step functions if, and only if, such a schedule exists. By using Lemma 3.1, we can say that we know how to build a schedule processing the n requests if, and only if, such a schedule exists. □

3.4 An upper bound on the optimal solution.

By using the previous result, we can deduce an upper bound on the number of processed requests in an optimal solution, which will be useful to compare heuristics.

Lemma 4. *Let us consider any platform and a set of n possible requests. Then we can find an upper bound on the number of requests which can be integrally processed, in polynomial time in n, p and p', by solving a rational linear program.*

Proof. We will use the same notations as in the previous lemma. Let x_i be equal to 1 if the request i is choosen, or equal to 0 in the other case. Now, we can write again the complete linear program:

$$\max\left(\sum_{i=1}^{n} x_i\right) \text{ such; that } \begin{cases} \forall i, x_i & \in \{0,1\} \\ \forall i, \sum_{u=1}^{q} x_{i,u}\alpha_{i,u} b_{i,u} & = x_i S_i \\ \forall i, \forall u, b_{i,u} & \geq 0 \\ \forall u, \sum_{i=1}^{n} b_{i,u} & \leq C_{tot} \\ \forall u, \forall j, \sum_{i=1}^{n} y_{i,j} b_{i,u} & \leq C_j \\ \forall u, \forall j', \sum_{i=1}^{n} y'_{i,j'} b_{i,u} & \leq C'_{j'} \end{cases}$$

This is a semi-integral linear program, and if we relax the constraint $\forall i, x_i \in \{0,1\}$ by the constraint $\forall i, 0 \leq x_i \leq 1$, we obtain, in polynomial time, an upper bound of the result of the optimal solution. □

We will use this upper bound in Section 6 to assess the quality of our heuristics.

4 Complexity results

In this section, we study the difficulty of the scheduling problem by fixing or, on the contrary, by relaxing, some parameters, in order to see why our problem is difficult, and which variants are $\mathcal{NP}$-complete or polynomial.

4.1 Off-line model

In this section, we assume that we know in advance when a request will be submitted and what its characteristics will be. We are therefore using the off-line model.

4.1.1 Case where the switch has a total capacity $C_{tot} \leq C_j, C'_j$

Here, we suppose that the total switch capacity is inferior to the capacity of the links. This assumption allows us to reuse many results on schedules for a single processor with preemption. We consider the off-line model, i.e., when the problem is entirely known before the execution of the algorithm.

Lemma 5. *Consider a platform with a switch capacity inferior to the capacity of each ingress and egress links. Then, the maximization of the number of completely processed requests ($\sum x_i$) or the maximization of the profit ($\sum w_i x_i$) are identical to the maximization of the number of processed tasks on a single processor, with preemption. Baptiste has found an $O(n^4)$ algorithm for $1|r_i;pmtn|\sum x_i$ [5] and an $O(n^{10})$ one for $1|r_i; S_i = S; pmtn|\sum x_i w_i$ [6]. Lawler has shown that the $1|r_i;pmtn|\sum x_i w_i$ case was pseudo-polynomial [17].*

Proof. We want to maximize $\left(\sum_{i=1}^{n} x_i w_i\right)$ under the following constraints:

- $\forall t \geq 0, \forall j \in \{1,\ldots,p\}, \sum_{i,\mathrm{src}(i)=j} b_i(t) \leq C_j$,
- $\forall t \geq 0, \forall j' \in \{1,\ldots,p'\}, \sum_{i,\mathrm{dest}(i)=j'} b_i(t) \leq C'_{j'}$,
- $\forall t \geq 0, \sum_{i=1}^{n} b_i(t) \leq C_{tot}$.

The third condition implies the previous two ones, since $\forall j, \{i, \mathrm{src}(i) = j\} \subseteq \{1,\ldots,n\}, \forall j, \{i, \mathrm{dest}(i) = j\} \subseteq \{1,\ldots,n\}$, and $C_{tot} \leq C_j, C'_{j'}$. Now, we show that we can transform this problem into a classical task scheduling problem on one processor with preemption. We can suppose, without any loss of generality, that $C_j = C'_{j'} = C_{tot}$. So, we can build (by following the method described in Lemma 3.2) a schedule, such that we have $\forall i, \forall t, b_i(t) \in \{0, C_{tot}\}$. At any time, there is at most one request in the switch, so we have the problem of task scheduling on one processor (the switch) with a speed of C_{tot} and with preemption. □

4.1.2 Switch of unbounded capacity and homogeneous ingress and egress links

Here we remove the constraint of the limited bandwidth of the switch.

If all requests have same size, arrive at the same time and have the same deadline, the problem is polynomial.

Lemma 6. *Consider a platform with an infinite total capacity, and with links of the same capacity C. If all requests have same size, same release date, and same deadline, then the maximization of the number of integrally processed requests ($\sum x_i$) is a polynomial problem (in n), and there exists a $O(n^2)$ algorithm to solve it.*

Proof. First at all, we can suppose, without any loss of generality, that we have $r_i = 0$ and $d_i = 1$, for all i. We want to maximize the number of scheduled requests $\sum i = 1^n x_i$. By using Lemma 3.1, we assume that the bandwidth allocated to any scheduled request is equal to $\frac{S}{r_i - d_i} = S$.

The difference between two requests can only be the ingress link, or the egress link, and we can sort all submitted requests into $p'p$ differents species. Then we can solve the problem by choosing how many requests from each species will be scheduled. We introduce the following definitions:

- $y_{j,j'}$ is the number of submitted requests coming from the ingress link j to the egress link j',
- $x_{j,j'}$ is the number of scheduled requests coming from the ingress link j to the egress link j'.

So, we want to maximize $\sum_{1 \le j \le p, 1 \le j' \le p'} x_{j,j'}$ under the following constraints:

- $S \sum_{1 \le j \le p, 1 \le j' \le p'} x_{j,j'} \le C_{tot}$,
- $\forall j \in \{1, \ldots, p\}, S \sum_{1 \le j' \le p'} x_{j,j'} \le C_j$,
- $\forall j' \in \{1, \ldots, p'\}, S \sum_{1 \le j \le p} x_{j,j'} \le C'_{j'}$,
- $\forall j, \forall j', x_{j,j'} \le y_{j,j'}$.

So, we have to solve a linear program with $p'p$ variables and $p'p + p + p' + 1$ constraints. We know how to do that in polynomial time ([14]). If all requests have same size, same release date, same weight and same deadline, we can maximize the number of scheduled requests in polynomial time.

$\square$

If all requests have same release date and deadline, then the problem is $\mathcal{NP}$-complete, even with one ingress link and one egress link.

Lemma 7 ($C_{tot} = +\infty$, $C_j = C'_j = C$, S_i, w_i is $\mathcal{NP}$-complete). *Consider a platform with an infinite capacity, and with links with the same capacity C. If the n requests have unspecified sizes and weights, the decision problem associated to the problem of maximization of the weighted number of processed requests ($\sum w_i x_i$) is $\mathcal{NP}$-complete, even when there are only one ingress link and one egress link, and when all requests have the same release date and the same deadline.*

Proof. We will use a simple reduction from the well-known $\mathcal{NP}$-complete Knapsack problem.

We consider a single ingress link and a single egress link, such that $C'_1 = C_1 = 1$, and n requests, with sizes $S_i \le 1$, dates $r_i = 0$, $d_i = 1$ and weights $w_i \ge 0$. So, we want to maximize $\sum_{i=1}^n x_i w_i$, such that $\forall t \in [0,1], \sum_{i=1}^n b_i(t) \le 1$.

By using Lemma 3.1, we know that we can take $b_i(t)$ constant and equal to S_i on $[0,1]$ if $x_i = 1$, and $b_i(t) = 0$ otherwise. So, our problem is to maximize $\sum_{i=1}^n x_i w_i$ with $\sum_{i=1}^n x_i S_i \le 1$. This new problem is exactly the Knapsack problem, which is known to be $\mathcal{NP}$-complete [11]. $\square$

The previous two complexity results show that request sizes and weights are both responsible for the $\mathcal{NP}$-completeness of our problem.

If all requests have same release date, same deadline, and same weight, then the problem remains $\mathcal{NP}$-complete.

Lemma 8. *Consider a platform with an infinite total capacity and with links of the same capacity C, and a set of requests of unspecified sizes, but with the same release date $r_i = 0$ and the same deadline $d_i = 1$. Then the decision problem associated to the maximization problem of the number of processed requests ($\sum x_i$) is $\mathcal{NP}$-complete.*

Proof. We will show that this problem is $\mathcal{NP}$-complete, by reducing it to the $\mathcal{NP}$-complete $2-\text{PARTITION}-\text{EQ}$ problem. [15]. The decision problem associated to our problem is "If we have n requests, can we scheduled at least K of them ?". We know that our problem is in $\mathcal{NP}$, since we can check in polynomial time whether there exists a schedule for a given set of requests (cf. Lemma 3.3). Now, let us consider an instance A of n integers $(a_1,\dots,a_n)$ of the $2-\text{PARTITION}-\text{EQ}$ problem, with $n \geq 5$ and $\forall i \in \{1,\dots,n\}$, $a_i \leq \left(\sum_{i=1}^n a_i\right)/2$. We know that the decision problem "can we find a subset I of $\{1,\dots,n\}$ with $\frac{n}{2}$ elements, such that $\sum_{i\in I} a_i = \left(\sum_{i=1}^n a_i\right)/2$" is $\mathcal{NP}$-complete.

First, let us define some new notations:

- $C = \sum_{i=1}^n a_i$
- $\forall i \in \{1,\dots,n\}, a'_i = a_i + C$
- $C' = \frac{1}{2}\left(\sum_{i=1}^n a'_i\right) = \frac{1}{2}(n+1)C$
- $\forall i \in \{1,\dots,n\}, \lambda_i = \frac{a'_i}{4}$
- $\forall i \in \{1,\dots,n\}, k_i = \lfloor \frac{4C'}{a'_i} - 5 \rfloor$
- $T = \sum_{i=1}^n k_i$

From the instance A, we build an instance B of our problem as follows (see Fig. 4):

- $n+2$ ingress links, $1,\dots,n+2$, each with a capacity equal to C',
- n egress links, $1,\dots,n$, each with the same capacity C',
- a set $\mathcal{A}$ of n requests, $1,\dots,n$ with $\forall i \in \{1,\dots,n\}, S_i = a'_i, \text{src}(i) = 1, \text{dest}(i) = i$
- a set $\mathcal{B}$ of n requests, $n+1,\dots,2n$ with $\forall i \in \{1,\dots,n\}, S_{n+i} = a'_i, \text{src}(n+i) = 2, \text{dest}(n+i) = i$
- a set $\mathcal{C}$ of T requests such that, $\forall i \in \{1,\dots,n\}$, we have k_i identical requests (i,j) ($1 \leq j \leq k_i$), with $S_{i,j} = \lambda_i, \text{src}(i,j) = i+2, \text{dest}(i,j) = i$.

Using $a_i + C$ instead of a_i ensures that $k_i > 2$:

$$k_i > \frac{4C'}{a'_i} - 6 = \frac{2(n+1)C}{a_i + C} - 6.$$

We have

$$n \geq 5 \Leftrightarrow (n-3) \geq 2$$

and

$$a_i \leq \frac{C}{2} \Leftrightarrow C \geq 2a_i.$$

So, we have:

$$(n-3)C \geq 4a_i \Leftrightarrow (n+1)C \geq 4(a_i + C)$$
$$\frac{2(n+1)C}{a_i + C} \geq 8.$$

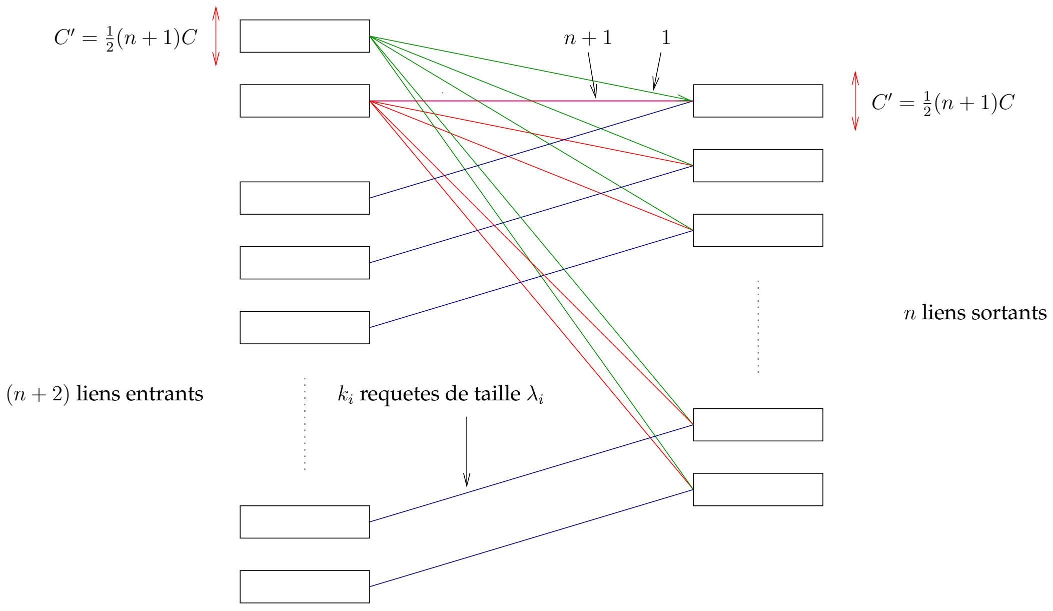


Figure 4: Platform used for the reduction from $2-\mathrm{PARTITION}-\mathrm{EQ}$

Eventually, we have:

$$k_i > \frac{4C'}{a'_i} - 6 = \frac{2(n+1)C}{a_i + C} - 6 \geq 2$$

which is the desired result. Now, we show that we have a polynomial number (in n) of requests.

We have $k_i \leq \frac{4C'}{a'_i} - 4 \leq 2\frac{(n+1)C}{a_i+C} \leq 2\frac{(n+1)C}{C} \leq 2(n+1)$. So, $T = \sum_{i=1}^{n} k_i \leq 2n(n+1)$, and the total number of requests is inferior to $2n(n+1) + 2n$, then we can say that the instance B is polynomial in n.

So, we have $T + 2n$ requests, and we want to know if we can schedule at least $K = T + n$ ones. We will show that at most $T + n$ requests can be scheduled, and that these $T + n$ requests can be scheduled only if the set I of the $2-\mathrm{PARTITION}-\mathrm{EQ}$ problem exists.

- At most $T + n$ requests can be scheduled:
  Let us consider the egress link i ($1 \leq i \leq n$), without taking into account the ingress links. The different cases are:
  - It accepts the requests $i \in \mathcal{A}$ and $n+i \in \mathcal{B}$, and exactly n_i of the k_i requests $(i,j)_{1\leq j\leq k_i}$. $n_i + 2 \leq k_i + 2$ requests are scheduled, but we can show that $n_i < k_i - 2$ (we have $k_i > 2$). Indeed, the bandwdidth needed to schedule i, $n+i$ and at least $k_i - 2$ requests is at least equal to $2a'_i + (k_i - 2)\lambda_i$. By definition of k_i and λ_i, we have:

$$\begin{aligned} 2a'_i + (k_i - 2)\lambda_i &= 2a'_i + \left(\left\lfloor \frac{4C'}{a'_i} - 5 \right\rfloor - 2\right)\frac{a'_i}{4} \\ &> 2a'_i + \left(\frac{4C'}{a'_i} - 8\right)\frac{a'_i}{4} \\ &= 2a'_i + C' - 2a'_i \\ &= C' \end{aligned}$$

    So, no schedule can accept more than $k_i - 3$ requests if it accepts both $i \in \mathcal{A}$ and $i + n \in \mathcal{B}$.

- It accepts a single request between $i \in \mathcal{A}$ and $n+i \in \mathcal{B}$, and n_i requests of the k_i requests (i,j). $n_i + 1 \leq k_i + 1$ requests are scheduled, and we will show that we can have $n_i = k_i$, since we have $a'_i + k_i\lambda_i \leq C'$. By definition of k_i and λ_i, we have:

$$\begin{aligned} a'_i + k_i\lambda_i &= a'_i + \left\lfloor \frac{4C'}{a'_i} - 5 \right\rfloor \frac{a'_i}{4} \\ &< a'_i + \left(\frac{4C'}{a'_i} - 4 \right) \frac{a'_i}{4} \\ &= C' \end{aligned}$$

  An egress link can accept $k_i + 1$ requests, with $k_i > 2$.
- It only accepts n_i of the k_i requests (i,j). $n_i \leq k_i$ requests are scheduled.

So, the total number of scheduled requests is equal to $\alpha + 2\beta + \gamma$, where α is the number of egress links with a single request between $i \in \mathcal{A}$ and $n+i \in \mathcal{B}$, β is the number of egress links with both $i \in \mathcal{A}$ and $n+i \in \mathcal{B}$, and γ is the number of the scheduled requests $(i,j) \in \mathcal{C}$.

We have $\gamma \leq T$ (by definition), but we also have $\gamma \leq T - 3\beta$ (since $n_i < k_i - 2$ when a link accepts both i and $i+n$), and we have $\alpha \leq n$. So, we have $\alpha + 2\beta + \gamma \leq n + 2\beta + T - 3\beta = n + T - \beta$.

Then, $\alpha + 2\beta + \gamma = T + n$ implies $\alpha = n$, $\beta = 0$, $\gamma = T$, i.e., each egress link has exactly one of the two requests between i and $n+i$ and all its requests (i,j).

- $T + n$ requests are scheduled if, and only if, the set I (for the problem A) exists:

  We have seen that the bound $T+n$ can only be reached when on every egress link exactly one request between i and $i+n$ is scheduled. So, to reach this bound implies that for all $i \in \{1,\ldots,n\}$, i or $n+i$ has to be scheduled. Let I be the set of scheduled requests from the ingress link 1.

  We have the following result: we can schedule $n+T$ requests if, and only if, $\sum_{i\in I} a'_i \leq C'$ and $\sum_{i \notin I} a'_i \leq C'$. By using the definition of C', we then have the following equality:

$$\sum_{i\in I} a'_i = \sum_{i\notin I} a'_i.$$

  If $|I| \geq \frac{n}{2} + 1$, then $\sum_{i\in I} a'_i > \sum_{i\in I} C = |I| \times C \geq \left(\frac{n}{2}+1\right) C = C'$. It cannot be true, since it exceeds the bandwidth of an ingress link. By the same way, , we cannot have $|I| \leq \frac{n}{2} - 1$ (by using $\{1,\ldots,n\} - I$), so, we have $|I| = \frac{n}{2}$. The we can write:

$$\begin{aligned} \sum_{i\in I} a'_i &= \sum_{i\notin I} a'_i \\ \Leftrightarrow \left(\sum_{i\in I} a_i\right) + \frac{n}{2}C &= \left(\sum_{i\notin I} a_i\right) + \frac{n}{2}C \\ \Leftrightarrow \sum_{i\in I} a_i &= \sum_{i\notin I} a_i, \end{aligned}$$

  This problem is exactly the instance A of the $2-\text{PARTITION}-\text{EQ}$ problem. To conclude, we can say, that our problem is $\mathcal{NP}$-complete.

$\square$

We do not know the complexity of the case where all requests have the same size and the same weight, but different release dates and deadlines. However, we conjecture that it is $\mathcal{NP}$-complete too.

4.2 Online model

Here we suppose that the scheduler does not know in advance the characteristics of the requests, which are submitted during the execution of the scheduling algorithm. So, the scheduler has to choose requests and to allocate the correct bandwidths without knowing the future, and it cannot change already taken decisions. As can be expected, this new framework is more challenging, and decreases the quality of potential algorithms. If we note $\sum_{i\leq n} x_i^*$ the number of processed requests by an optimal algorithm, and $\sum_{i\leq n} x_i$ the number of processed requests by an algorithm A, we recall that A performs a competivity factor of ρ (we say that A is ρ-competitive) if, and only if, for every instance of the problem we have the following inequality:

$$\sum_{i\leq n} x_i^* \leq \rho \sum_{i\leq n} x_i$$

Lemma 9. *Consider a platform with an infinite capacity, and with links with the same capacity C.*

- *If all requests have same size and same weight, no online algorithm has a competivity factor strictly better than 2.*
- *If all requests have same weight, no online algorithm has a competivity factor strictly better than 3.*
- *In the general case (if weights and sizes are unspecified), no online algorithm has constant competivity factor.*

First and foremost, we prove the case where all requests have the same size and the same weight.

Proof. Let us consider 2 ingress links and 2 egress links. Then we submit to the algorithm the request 1 ($S_1 = 1$, $r_1 = 0$, $d_1 = 1$, $\mathrm{src}(1) = 1$, $\mathrm{dest}(1) = 1$).

We have two cases to consider:

- 1 is accepted : At $t = 1 - \epsilon$, we submit a second request 2 ($S_2 = 1$, $r_2 = 1 - 2\epsilon$, $d_2 = 2 - 2\epsilon$, $\mathrm{src}(2) = 1$, $\mathrm{dest}(2) = 2$). Once again, we have to consider two cases:
  - request 2 is accepted : request 1 is interrupted. Then we submit 4 ($S_4 = 1$, $r_4 = 1$, $d_4 = 2$, $\mathrm{src}(4) = 2$, $\mathrm{dest}(4) = 2$)
    - 4 is accepted and request 2 is interrupted. Both 1 and 4 could have been processed, but only 4 is processed.
    - 4 is rejected. Both 1 and 4 could have been processed, but only 2 is processed.
  - request 2 is rejected and we submit 3 ($S_3 = 1$, $r_3 = 1 - \epsilon$, $d_3 = 2 - \epsilon$, $\mathrm{src}(3) = 2$, $\mathrm{dest}(3) = 1$)
    - 3 is accepted and 1 is interrupted. Both 2 and 3 could have been processed, but only 3 is processed.
    - 3 is rejected. Both 2 and 3 could have been processed, but only 1 is processed.
- request request 1 is rejected, and we stop here the experiment. 1 could have been processed, and no request is processed.

In all cases, only a single request is processed, instead of 2. So, no online algorithm has a competivity factor strictly better than 2. □

Now, we will show that no online algorithm has a competivity factor strictly better thant 3 when requests can have different sizes.

Proof. Consider the requests defined in the following table:

i	S_i	r_i	d_i	src(i)	dest(i)
1	3	0	3	1	1
2	1	$1-8\epsilon$	$2-8\epsilon$	2	1
4	1	$2-8\epsilon$	$3-8\epsilon$	2	1
5	1	$3-8\epsilon$	$4-8\epsilon$	2	1
6	1	$2-5\epsilon$	$3-5\epsilon$	2	2
7	1	2	3	3	1
8	1	$3-8\epsilon$	$4-8\epsilon$	3	2
9	$1-3\epsilon$	$1-7\epsilon$	$2-10\epsilon$	2	3
10	1	$2-10\epsilon$	$3-10\epsilon$	2	3
11	1	$1-5\epsilon$	$2-5\epsilon$	2	2
12	1	$2-10\epsilon$	$3-10\epsilon$	3	2
13	1	1	2	3	3

We consider that request 1 was accepted (otherwise we stop here the experiment) and we submit 2.

- 2 is rejected, we submit 4
  - 4 is rejected, we submit 5
    - 5 is rejected : only 1 is processed, instead of 2, 4 and 5,
    - 5 is accepted : only 5 is processed, instead of 2, 4 and 5,
  - 4 is accepted, 1 is stopped and we submit 6
    - 6 is rejected, we submit 7
      - 7 is rejected : only 4 is processed, instead of 2, 6 and 7,
      - 7 is accepted : only 7 is processed, instead of 2, 6 and 7.
    - 6 is accepted, we submit 8
      - 8 is rejected : only 6 is processed, instead of 2, 4 and 8,
      - 8 is accepted : only 8 is processed, instead of 2, 4 and 8,
- 2 is accepted, 1 is stopped, we submit 9
  - 9 is rejected, we submit 10
    - 10 is rejected : only 2 is processed, instead of 1, 9 and 10,
    - 10 is accepted : only 10 is processed, instead of 1, 9 and 10,
  - 9 is accepted, we submit 11
    - 11 is rejected, we submit 13
      - 13 is rejected : only 9 is processed, instead of 1, 13 and 11,
      - 13 is accepted : only 13 is processed, instead of 1, 13 and 11,
    - 11 is accepted, we submit 12
      - 12 is rejected : only 11 is processed, instead of 12, 9 and 1,
      - 12 is accepted : only 12 is processed, instead of 12, 9 and 1.

In all cases, any *online* algorithm can at most process only one request, when three requests could have been processed by an optimal *off-line* algorithm. So, the competivity factor of any online algorithm cannot be strictly better than 3. □

Now, we will show that there exists no online algorithm with a constant competivity factor , when requests can have different sizes and weights.

Proof. Let us consider any online algorithm A, with a competivity factor better than an integer f. First, we submit request 1 ($S_1 = f^2$, $w_1 = f^2$, $r_1 = 0$, $d_1 = f^2$).

- request 1 is accepted. Then we successively submit f^2 requests $2, \ldots, f^2+1$ defined by: ($S_i = 1$, $w_i = f$, $r_i = i-2$, $d_1 = i-1$). If A accepts at least one of these requests, we stop submitting requests, so it cannot reach a gain better than f. So, there are two different cases:
  - A has not accepted any request among $2, \ldots, f^2+1$, so it reaches a gain equal to f^2 (with the request 1) instead of f^3 (by processing the f^2 requests with a weight f).
  - A has accepted at least one (and then exactly one) request among $2, \ldots, f^2+1$, so, it has a gain equal to f instead of f^2 (with the first request).
- request 1 is rejected. A has a gain equal to 0 instead of f^2.

In all cases, the online algorithm cannot reach a competivity factor better than f, for any given f. So, no online algorithm has a constant competivity factor. $\square$

5 Study of several algorithms

5.1 Off-line algorithms

5.1.1 Study of Earliest Deadline First in use without priorities.

With underloaded system, only one ingress link or one egress link. We recall that a underloaded system is a system in which all proposed requests can be correctly processed. In the particular case in which we have only one ingress link or one egress link ($p = 1$ or $p' = 1$), the Earliest Deadline First algorithm (EDF) is optimal [13]. EDF is less expensive than the approach based on a linear program.

EDF is not an optimal algorithm for an underloaded system with at least 2 ingress links and 2 egress links. In other words, we show that this classical algorithm is not even an approximation algorithm in this case, thereby exhibiting the combinatorial complexity induced by the many links..

Consider any $\epsilon > 0$. We can build an instance, such that $\sum_i x_i \leq \epsilon \sum_i x_i^*$, where $(x_i^*)_{1 \leq i \leq b}$ is an optimal solution.

Indeed, since we have $\lim_{n\to\infty} \frac{2}{2+n} = 0$, we can find a n such that $\frac{2}{2+n} \leq \epsilon$. Then we define the following instance:

i	S_i	r_i	d_i	src(i)	dest(i)
1	$2n$	n	$3n$	2	2
2	n	0	$2n$	1	1
3	1	0	$3n+1/2$	2	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$n+2$	1	0	$3n+1/2$	2	1

EDF will begin to schedule the request 2 from $t = 0$ to $t = n$, and then the request 1 from $t = n$ to $t = 3n$. So, the remaining n requests are not processed. On the contrary, an optimal algorithm will begin by processing the n requests $3, \ldots, n+2$ from $t = 0$ to $t = n$ then it will schedule the requests 2 and 1 in parallel from $t = n$. EDF will process only 2 requests over the $n+2$ available ones, which can all be scheduled, so it has an approximation factor worse than $\frac{2}{n+2} \leq \epsilon$.

5.1.2 A $\min(p, p')$-approximation if $C_j = C'_{j'} = C$, $C_{tot} = +\infty$, $w_i = 1$, and r_i, d_i, S_i are unspecified

Let $R = \sum_{i=1}^{n} x_i^*$ the optimal number of requests which can be completely processed.

Now, we only consider a single ingress link j. Since all egress links have the same capacity as the single ingress link, we can simplify the constraints of a valid schedule, and we can only

keep the constraint on the capacity of the ingress link. We know that we can transform any valid schedule into a schedule which uses bandwithes equal to 0 or C (i.e., $\forall i, \forall t, b_i(t) \in \{0, C\}$, see section 3.2). This problem then is exactly the same problem of scheduling tasks on a single processor. So, we note R_j the total number of requests which can be processed in this problem. We know that this problem is polynomial, since a $O(n^4)$ algorithm exists [5].
Of course, we have $R \leq \sum_{j=1}^{p} R_j$, and then $R \leq p \times \max_{1 \leq j \leq p}(R_j)$. If we decide to only process the $\max_{1 \leq j \leq p} R_j$ requests (by only processing the requests coming from a link which realizes this maximum), we have a p-approximation of the original problem.

With the same idea, but using the egress links and not the ingress links, we can have a $\min(p, p')$-approximation.

5.1.3 Greedy algorithms

Greedy algorithms are very natural for our problem. We build three algorithms using different criteria.

1. We sort all the requests by increasing $\frac{S_i}{d_i - r_i}$ (this formula corresponds to the minimal average allocated bandwidth neeeded to process the request) . If the request i can be processed with all the already chosen requests (among the $i - 1$ previous ones) - we can check this in polynomial time according to the lemma 3.3 -, e add it to the set of processed requests, otherwise we reject it.

2. We sort all requests by increasing size S_i. Then we proceed like in the previous algorithm for scheduling the requests.

3. We try to process all the requests, and as long as it is not possible, we remove the request which "obstructs" the largest number of other requests. This notion of obstruction can be more formally defined by "two differents requests obstruct each other if their respective intervals $[r_i; d_i[$ intersect each other, and if they have at least one common link (ingress, or egress link)". Like in previous greedy algorithms, we use a linear program to determine if we can process the set of chosen requests.

5.1.4 Linear programs

We know that the optimal solution could be computed by a semi-integral linear program, but the computation time of such a linear program is very large. We can use a rational relaxation of the integral constraint (on the x_i variables) in order to have an approximated solution. If x_i is the integer variable (equal either to 0 or 1), indicating whether we have to process or not the request i, then x_i^* will be a rational variable *between* 0 and 1.

- *"Naive" approximation.* A very common method to approximate a linear program is to round the rational values to the nearest integer: if $x_i^* < 1/2$, we let $x_i = 0$, otherwise we let $x_i = 1$. Nonetheless, after having chosen a first set of requests, we have to check whether this set is feasible, otherwise we have to remove some requests as long as the chosen set of requests is not correct.

- *"Naive" approximation, and greedy completion.* The idea is only to add a greedy step (by example by sorting remaining requests in increasing $S_i/(r_i - d_i)$ order), to try to add some rejected requests.

- *Randomized linear program.* This idea is taken from Coudert and Rivano [10], and consists in an n step process. In each step, a variable x_i is randomly chosen, and set to 0 with a probability x_i^*, and otherwise to 1. The following step allows to check whether the set of chosen requests is always feasible; if not, we have to set the last chosen variable to 0.

5.2 Online algorithms

5.2.1 FCFS

The First Come, First Served algorithm (FCFS) is a very simple algorithm, easy to implement, and which can have quite good results. One of its good points is that it can work as well in an off-line context as in an online context. In our problem, requests are sorted by increasing release date (r_i), and the request i is scheduled if we can complete it before its deadline (while completing the previous scheduled requests), and we allocate to it the maximum available bandwidth.

5.2.2 Load balancing

In this method, we have to keep a list of ready requests (i.e., whose release date is anterior to the current time, and whose deadline is such that we can still completely process it). When a request is released or finishes, we choose the requests which will be scheduled in the different links. We assign a priority to the requests: a request will be more important if it does not increase too much the load of its links, or if it was already begun (to increase the number of completely processed requests). The main drawback of this online method is that it can interrupt some already started requests.

6 Experiments

The different heuristics (described above) were compared by simulations. The simulated platform has 10 ingress links and 10 egress links, all with a capacity of 1GB/s. The used requests have a randomly chosen size, between 100GB and 1TB. Their starting time was Poisson distributed, the parameter of this Poisson distribution is the average arrival time of the requests (between 0.1 and 5 seconds). The value of the parameter varies to obtain heavy loaded scenarios and less loaded cases. The average bandwidth needed to send requests are between 10MB/s and 1GB/s, so we can fix the corresponding deadlines. The simulation was coded in C++, and an external library, LP_solve [2], was used to solve the different linear programs used by the algorithms.

6.1 Increasing the speed of algorithms

As said before, many heuristics are based on the use of linear programs. The size of these linear programs quickly increases with the number of chosen requests ($O(n^2)$ variables and constraints). So, we cannot think to use these heuristics with a large number of requests. By example, with only 200 requests, one computation was not finished after 60 hours. To allow tests with a large number of requests, two simplifications were implemented:

- Limitation of the number of bandwidth variations in the linear programming problems .

  We have already seen in Section 3.3 that if we have selected n requests, and if only we allow each bandwidth function b_i to change $2n$ times, we can solve exactly the problem. In order to speed up the algorithm, we will not allow anymore such a large number of variations. We arbitrarily fix this number to be quite small (for example 10 or 20, instead of $2n$). So the linear program becomes smaller, and then the computation is by far quicker.

  Of course, we cannot expect to always obtain the best solution anymore, so we have to assess the impact of this simplification on the quality of the produced solutions (cf. Fig. 6).

- Temporal slicing of the original problem into small successive sub-problems.

  The previous method has, as main advantage, the property that it does not decrease too much the quality of the results. However, with this method, the number of variables in linear programs can still become really too large. So, some further optimization is needed. A naive idea is to cut the large interval $[min(r_i); max(d_i)[$ in k small intervals $[t_j, t_{j+1}[$. If a request i can be processed in more than one interval, we arbitrarily decide to force that it is

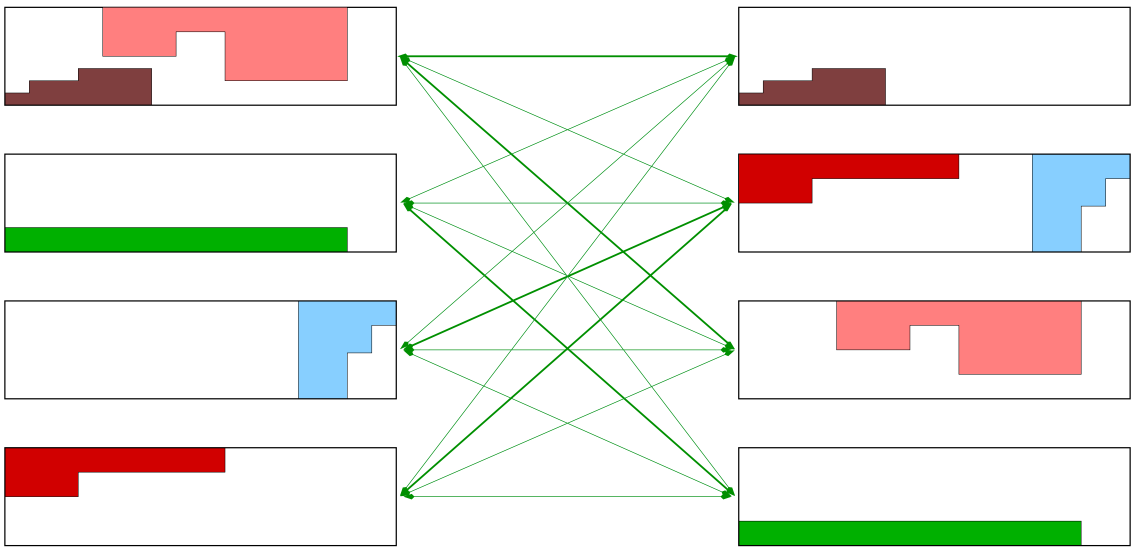


Figure 5: A platform example, with several requests.

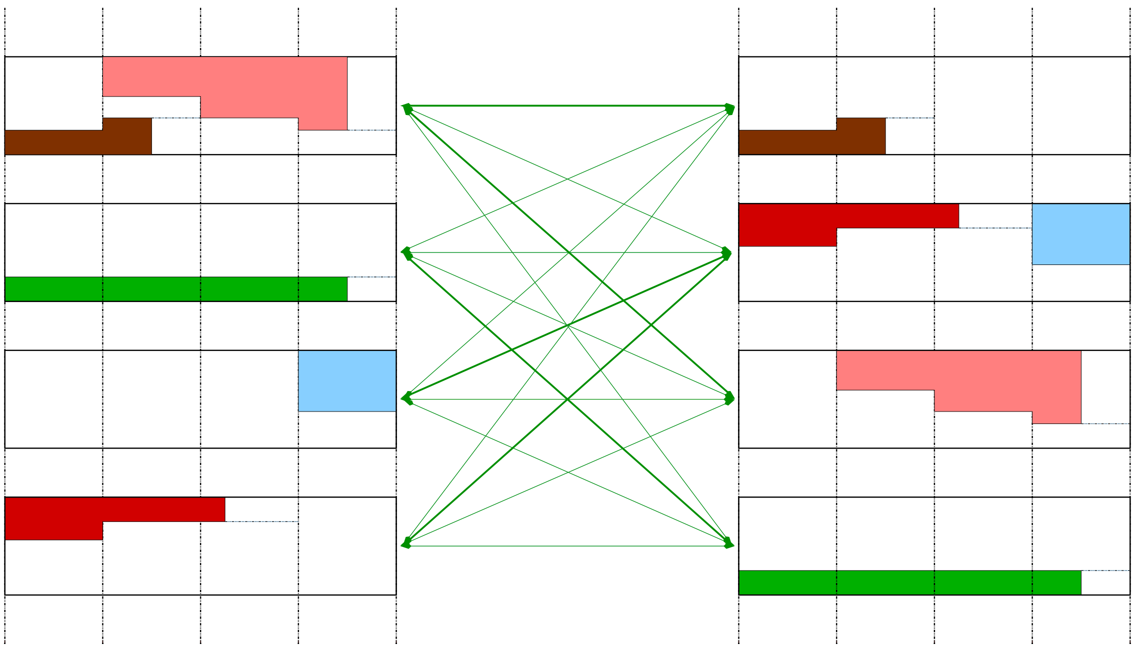


Figure 6: The same example, with $nb\_var = 4$.

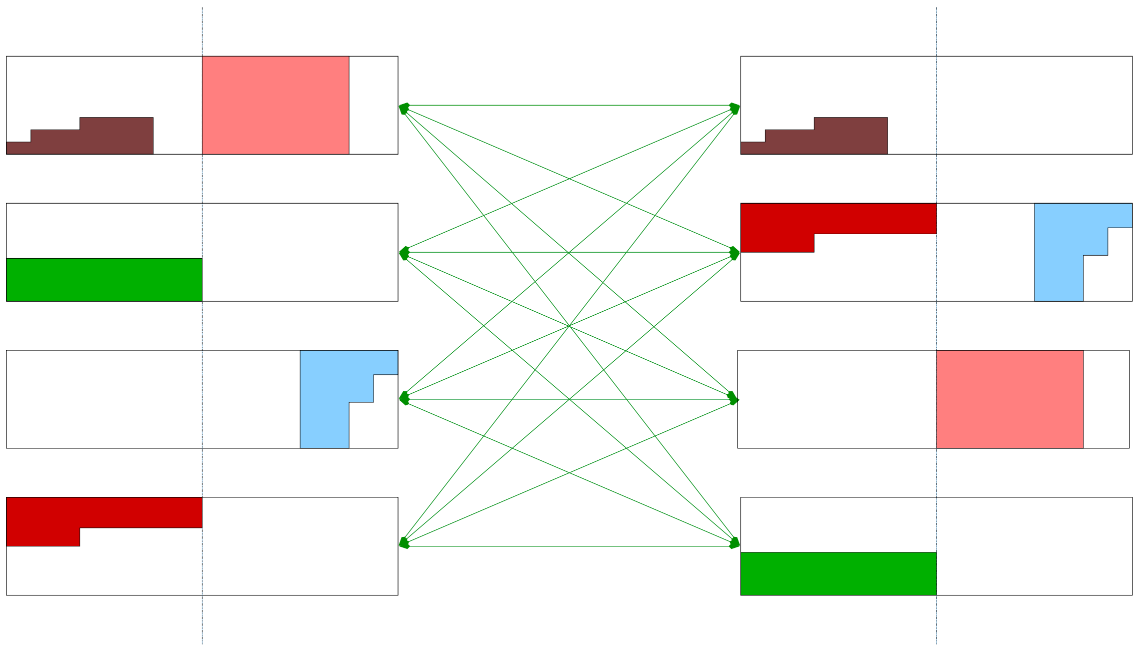


Figure 7: The same example, after temporal cutting

executed in a single interval j. Therefore, we chose $r_i = max(t_j, r_i)$ and $d_i = min(t_{j+1}, d_i)$; So, we force the request i to be completely processed during this interval j (cf. figure 7).

This method still suffers from a drawback: the number of requests in an interval is not constant. To ensure that the computation time will be small enough, it is smarter to use different sizes for the k intervals, but to assign the same number of requests to each interval. So, we have k small linear programs with an almost constant size.

The final problem has stronger constraints, so the number of processed requests will decrease. However,the complexity will be linear in the number of requests for each algorithm. The idea was originally to speed-up linear programs, but, of course, this approach can be used with every heuristic.

So, we can now choose between a better result costing a large amount of computations, and a worse result obtained in a shorter time.

6.2 Some results

In this section, t_{av} will be the average time between the arrival of two successive requests, and it allows to decide if the global system is heavily loaded or not. In the following experiments, t_{av} will take values between 0.1 and 5 seconds.

- Influence of n on the computation time

  As we can see on the three figures 8, 9, 10, with $n \leq 200$, computation time quickly increases, at least with linear programs-based heuristics. Since theses heuristics have good results, accelerating methods are very interesting if they don't decrease too much the quality of the heuristics.

  These figures were created with a small set of instance (about 20 instances for each value of n), but we can see that the computation time can become very large.

- Influence of the number nb_var of steps of bandwidth functions.

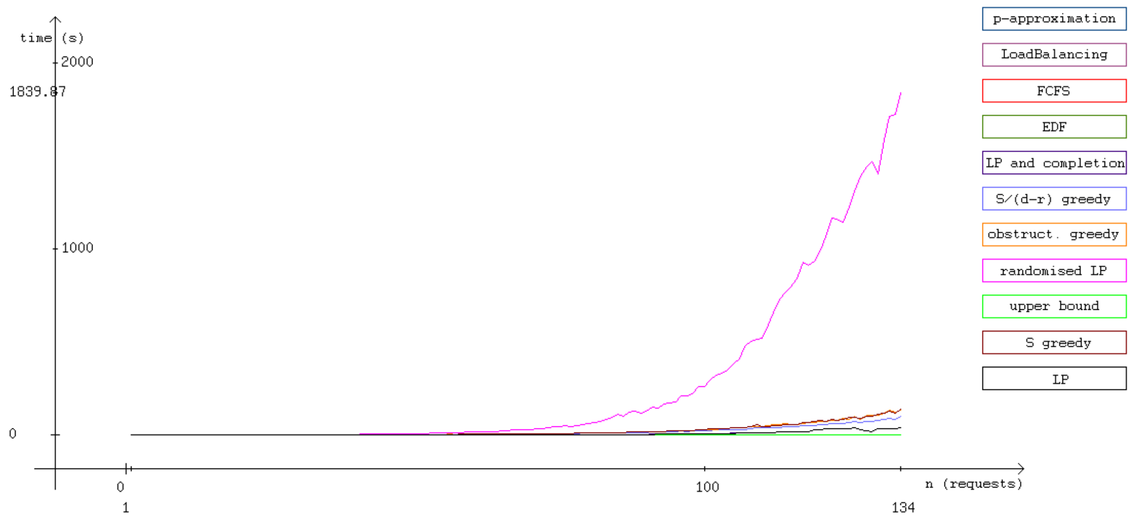


Figure 8: Influence of n on the computation time ($t_{av} = 0.1$, $nb\_var = 0$: no approximation).

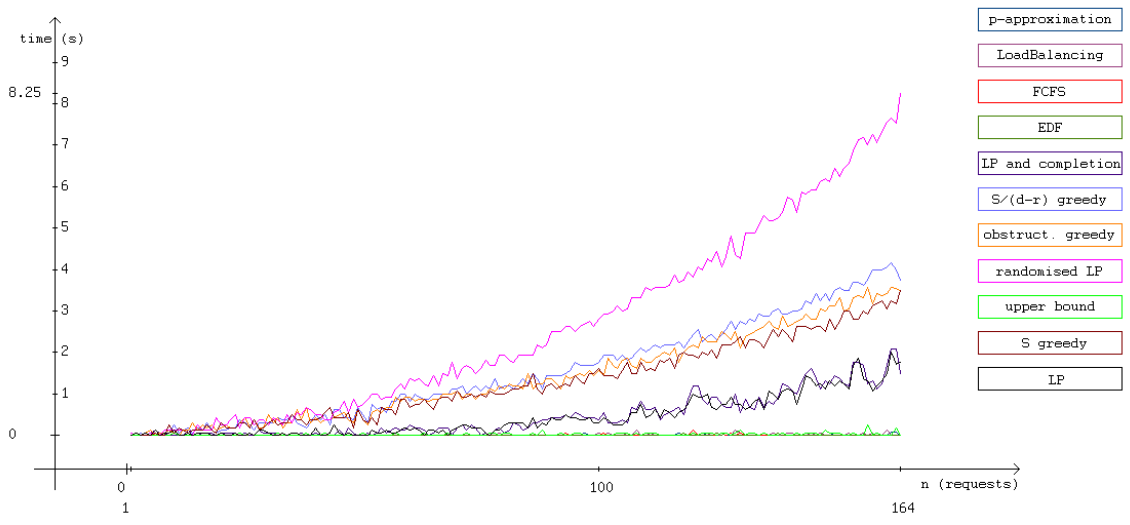


Figure 9: Influence of n on the computation time ($t_{av} = 0.1$, $nb\_var = 10$).

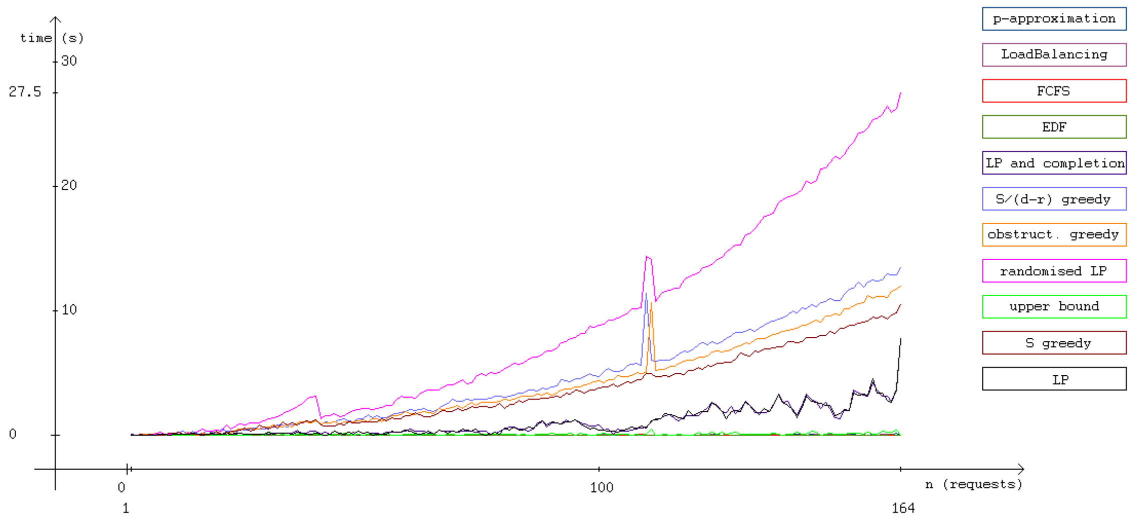


Figure 10: Influence of n on the computation time ($t_{av} = 0.1$, $nb\_var = 40$).

For some practical reasons, a null number of variations corresponds to the original linear program, without any approximation. Excepting for this special value, the greater the value of nb_var, the better approximation.

Figure 11 clearly shows that using this acceleration has a great benefit for the computation time, especially for the randomised linear program.

The computation time is linear in nb_var, here between 10 and 80. We can also see from Figure 12. So this idea is very interesting.

- Influence of the average load of the system

  In order to compare the different heuristics, several instances of the problem were generated, with $12,000$ requests and average arrival times between 0.1 and 5 seconds. All heuristics were not tested, since those which were too slow or not interesting enough (like the p-approximation) were rejected.

  Two different parameters were used for the temporal slicing: by blocks of 200 requests for the three greedy heuristics using linear programs, and by blocks of $1,000$ requests for other heuristics. From now, we call "optimal" solution the upper bound of the optimal solution, computed by blocks of $1,000$ requests.

  Figure 13 shows that the performance iis better if the system is less loaded (which is quite normal). The two greedy algorithms have good results, with around 75% of the optimal soution, if the system is heavily loaded. The three online heuristics (EDF, FCFS and Load balancing) are very similar between each others, and they are around 50% of the optimal solution. This is quite good for online algorithms.

  When the system is less loaded, all heuristics perform well (more than 80% of the optimal solution).

  On Figure 14, we can see the computation times of these heuristics. It is easy to see the main drawback of linear program-based algorithms, since the computation time is not predictible. Since all problems are with $12,000$ requests, one can think that computation time will be almost constant, and it is clearly not the case. Due to the chosen implementation, we can understand that the computation time is greater in case of heavily loaded system, but it is not enough to explain this variation. In fact, this variation is explained by the fact that LP_solve uses the simplex method, which can be exponential is the worst case (even if it is polynomial in average).

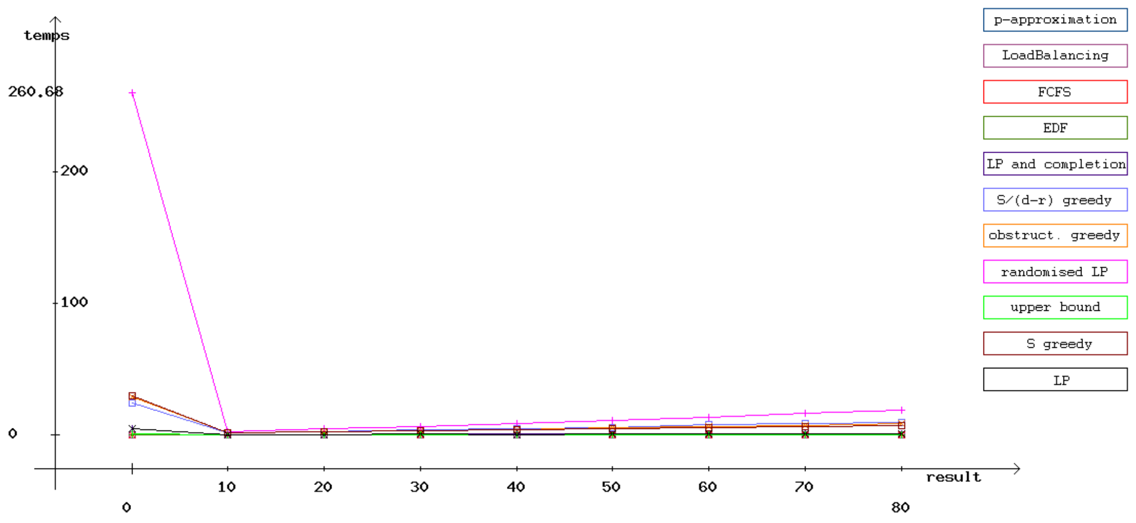


Figure 11: Influence of the number of variations on the computation time ($t_{av} = 0.1$).

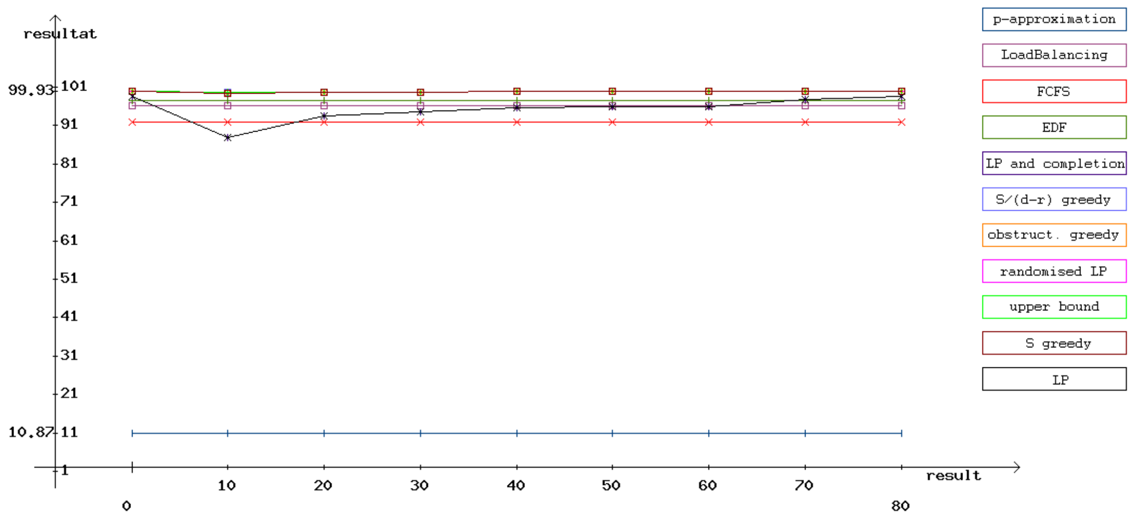


Figure 12: Influence of the number of variations on the objective function ($t_{av} = 0.1$).

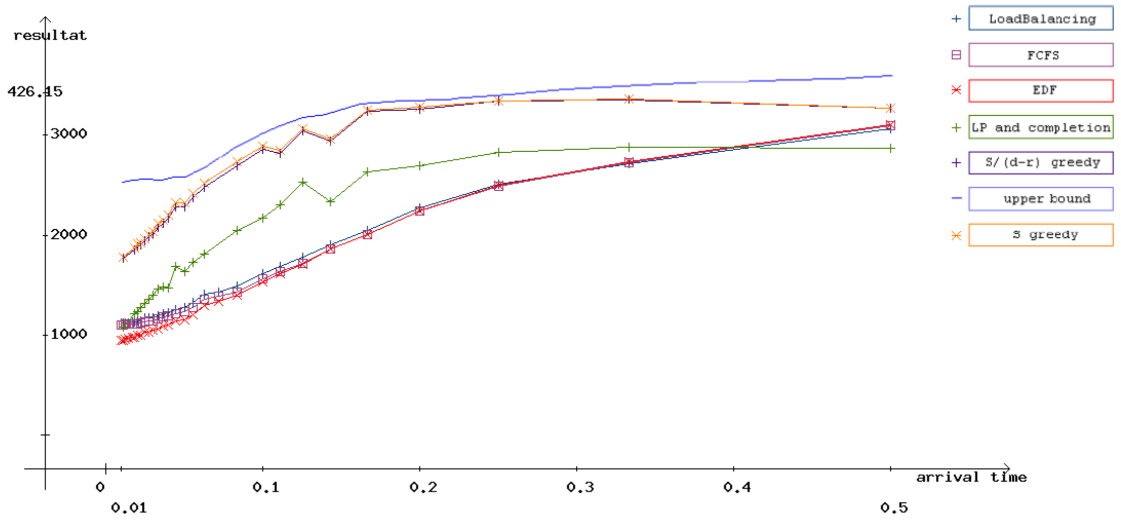


Figure 13: Influence of the average arrival time t_{av} on the scheduled requests number.

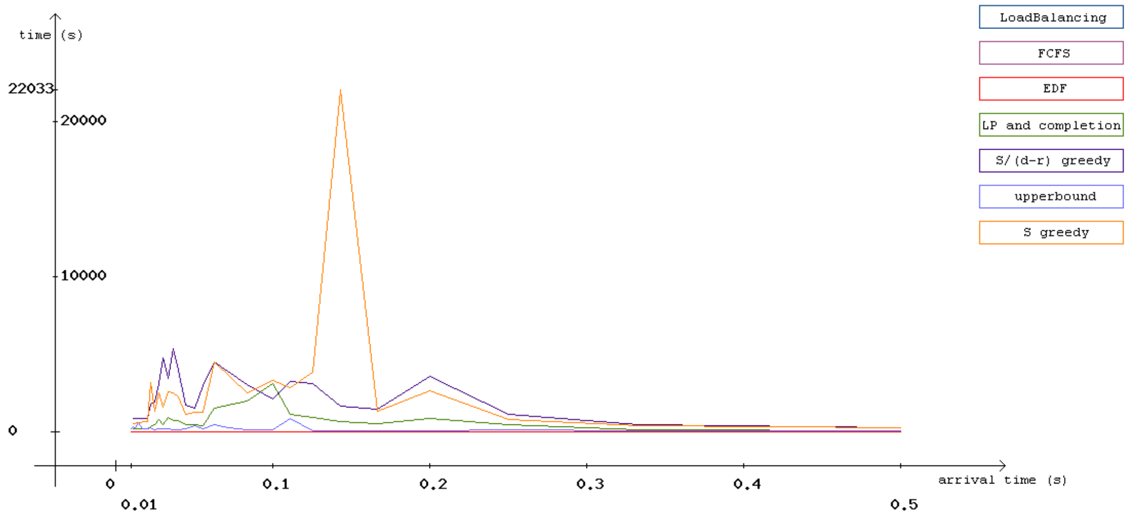


Figure 14: Computation time for several values of t_{av}.

7 Conclusion

This work extend previous work on the optimization of network resource sharing in grids. The major novelty is to allow bandwidth variations during the schedule of a request, thereby providing additional flexibility to the scheduler.

Different variants of the problem were studied, in order to assess the difficulty of the off-line problem and of the online problem. Several algorithms were implemented, tested and compared to an upper bound of the optimal solution.

Obviously, online heuristics have worse performance than off-line heuristics, because the online problem is more difficult. But it could be interesting to see if we can enhance online heuristics to derive better solutions. Moreover, the use of linear programs in the greedy heuristics slows them down, and it would be useful to find another way of determining whether a subset of requests is feasible or not.

In this paper, only a very simple platform (a switch) was studied. Further extensions should deal with more complex platforms, in order to model actual networks more closely. While the combinatorial nature of the scheduling problem would be even greater on such complex platforms, we hope that efficient heuristics could still be introduced and evaluated.

References

[1] The grid 5000 project. http://www.grid5000.org.

[2] Mixed integer programming (mip) solver. http://groups.yahoo.com/group/lp_solve/.

[3] *Optimizing Network Resource Sharing in Grids*. IEEE Communications Society Press, 2005.

[4] F. Baille. *Algorithmes d'approximation pour des problèmes d'ordonnancement bicritères : application à un problème d'accès au réseau*. PhD thesis, 2005.

[5] P. Baptiste. An $o(n^4)$ algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Operations Research Letters*, 24:175–180, 1999.

[6] P. Baptiste. Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times. *Journal of Scheduling*, 2:245–252, 1999.

[7] C. Bin-Bin and P. Vicat-Blanc Primet. A flexible bandwidth reservation framework for bulk data transfers in grid networks. Research Report 2006-20, LIP, ENS Lyon, France, jun 2006.

[8] P. Brucker. *Scheduling Algorithms*. Springer-Verlag, 2004.

[9] Vasek Chvatal. *Linear Programming*. A Series of Books in the Mathematical Sciences. Freeman, 1983. ChVA v 83:1 P-Ex.

[10] D. Coudert and H. Rivano. Lightpath assignment for multifibers WDM optical networks with wavelength translators. In *IEEE Globecom*, volume 3, pages 2686–2690, Taiwan, Nov 2002. OPNT-01-5.

[11] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[12] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[13] J.R. Jackson. Scheduling a production line to minimize maximum tardiness. Research Report 43, UCLA, University of California, 1955.

[14] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, New York, NY, USA, 1984. ACM Press.

[15] R. M. Karp. *Reducibility among combinatorial problems*. Plenum Press, New York, 1972.

[16] T.W. Lam, T.W. Johnny Ngan, and K.K. To. Performance guarantee for edf under overload. *Journal of Algorithms*, 2002.

[17] E.L. Lawler. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Annals of Operations Research*, 26:125–133, 1990.

[18] A. Legrand, A. Su, and F. Vivien. Minimizing the stretch when scheduling flows of biological requests. Research Report LIP 2005-48, Laboratoire de l'informatique de parallélisme (LIP), ENS de Lyon, France, oct 2005. Also available as INRIA research report 5724.

[19] L. Marchal, P. Primet, Y. Robert, and J. Zeng. Optimizing network resource sharing in grids. Research Report 2005-10, LIP, ENS Lyon, France, mar 2005.

[20] L. Marchal, P. Vicat-Blanc Primet, Y. Robert, and J. Zeng. Scheduling network requests with transmission window. Research Report 2005-32, LIP, ENS Lyon, France, jul 2005.

[21] L. Marchal, Y. Yang, H. Casanova, and Y. Robert. A realistic network/application model for scheduling divisible loads on large-scale platforms. Research Report 2004-21, LIP, ENS Lyon, France, apr 2004. Also available as INRIA Research Report RR-5197.

[22] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer-Verlag, 2003.