Scheduling communication requests traversing a switch: complexity and algorithms

Matthieu Gallet Yves Robert Frédéric Vivien

Laboratoire de l'Informatique du Parallélisme
UMR CNRS-ENS Lyon-INRIA-UCBL 5668
Lyon, France
{Matthieu.Gallet, Yves.Robert, Frederic.Vivien}@ens-lyon.fr

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.

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 [11, 12, 6, 3].

The bandwidth allocation is an important problem in distributed computing, 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, P. Primet, Y. Robert and J. Zeng [11], whose model consists of some ingress and 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. Due to lack of space, all proofs are omitted but can be found in the companion research report [8]. Algorithms and heuristics are given in Section 5, and experimental results are provided in Section 6. Finally, we give some con-

15th EUROMICRO International Conference on Parallel, Distributed and Network-Based Processing (PDP'07)
0-7695-2784-1/07 $20.00 © 2007 IEEE

clusions 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 $\text{src}(i) \in \{1,\ldots,p\}$ to the egress link $\text{dest}(i) \in \{1,\ldots,p'\}$. This request has a size (or area) $S_i \in \mathbb{Q}^+$ 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: $\forall i \in \{1,\ldots,n\}, x_i \in \{0,1\}$. 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 completed at all).
- b_i is a function, which is integrable over $\mathbb{R}^+$,
- bandwidth functions are non-negative functions: $\forall t \geq 0, \forall i \in \{1,\ldots,n\}, b_i(t) \geq 0$,
- deadlines are strictly enforced: $\forall t \geq d_i, \forall i \in \{1,\ldots,n\}, b_i(t) = 0$,
- a request cannot be processed before its release date: $\forall t < r_i, \forall i \in \{1,\ldots,n\}, b_i(t) = 0$,
- we cannot exceed the capacity of an ingress link: $\forall t \geq 0$, $\forall j \in \{1,\ldots,p\}$, $\sum_{i,\text{src}(i)=j} b_i(t) \leq C_j$,
- we cannot exceed the capacity of an egress link: $\forall t \geq 0$, $\forall j' \in \{1,\ldots,p'\}$, $\sum_{i,\text{dest}(i)=j'} b_i(t) \leq C'_{j'}$,
- 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: $\forall i \in \{1,\ldots,n\}, x_i \int_0^{+\infty} b_i(t)dt = x_i S_i$.
  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 Step functions are sufficient.

So far, 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.*

3.2 Ingress and egress links of same bandwidth.

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, $b_i(t) \in \{0, C\}$.

3.3 Satisfiability of all requests.

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.*

Let $\{t_1, \ldots, t_q\}$ be equal to $\bigcup_{i=1}^{n}\{r_i, d_i\}$, with $0 \leq t_1 < \cdots < t_q$ (we have $q \leq 2n$). We define $t_0 = 0$ and $t_{q+1} = +\infty$. Now we define:

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

Then our problem is equivalent to solve the following linear program:

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

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.*

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 any link. 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 or egress link. 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* $1|r_i; pmtn|\sum x_i$ *[4] and an* $O(n^{10})$ *one for* $1|r_i; S_i = S; pmtn|\sum x_i w_i$ *[5]. Lawler has shown that the* $1|r_i; pmtn|\sum x_i w_i$ *case was pseudo-polynomial [10].*

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.*

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.*

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.*

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 has a competitive ratio of ρ (we say that A is ρ-competitive) if, and only if, for every instance of the problem we have the following inequality:

$$\sum_{i \leq n} x_i^* \leq \frac{1}{\rho} \sum_{i \leq n} x_i$$

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 competitive ratio strictly better than 2, if all requests have same weight, no online algorithm has a competitive ratio strictly better than 3 and in the general case (if weights and sizes are unspecified), no online algorithm has constant competitive ratio.*

5 Study of several algorithms

5.1 Off-line algorithms

5.1.1 Study of Earliest Deadline First without priorities.

With underloaded system, only one ingress link or one egress link. We recall that an 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 [9]. 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 competitive ratio 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 bandwidths 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 [4].
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 needed 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 -, we 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 different 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 [7], 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

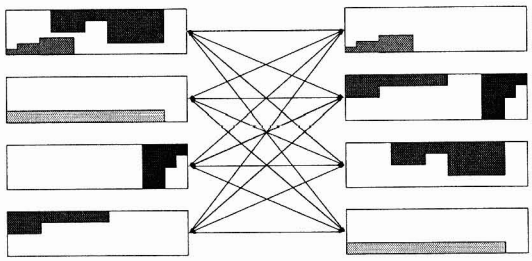

Figure 1. A platform example, with several requests.

process it). When a request is released or finishes, we choose the requests which will be scheduled on 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 on the Grid5000 platform [1]. 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 is 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, like the randomized approximation. 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 problem.

  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 nb_var to be quite small (for example 10 or 20, instead of $2n$, cf. Fig. 2). 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.

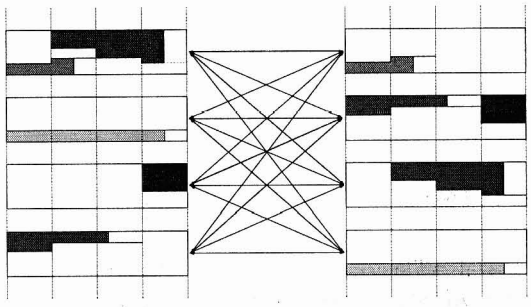

Figure 2. The same example, with $nb\_var = 4$.

- 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 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 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 3).

  This method still suffers from a drawback: the number of requests in an interval is not constant. To ensure that the solving time will be small enough, it is smarter to use different sizes for the k intervals, but to assign the same number of re-

quests 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.

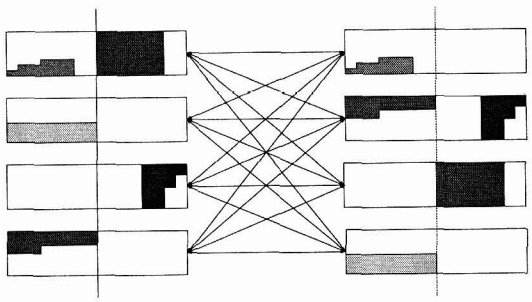

Figure 3. The same example, after temporal slicing

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 the number nb_var of steps of bandwidth functions.

  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 the approximation.

  Figure 4 clearly shows that using this acceleration has a great benefit for the computation time, especially for the randomized linear program. Only three heuristics are shown on the figure, but others are between the greedy one and the randomized one.

  The computation time is linear in nb_var, here between 10 and 80. We can also see from Figure 5. On this figure, some heuristics have almost the same results, so they are grouped with the same symbol. So this optimization is very interesting.

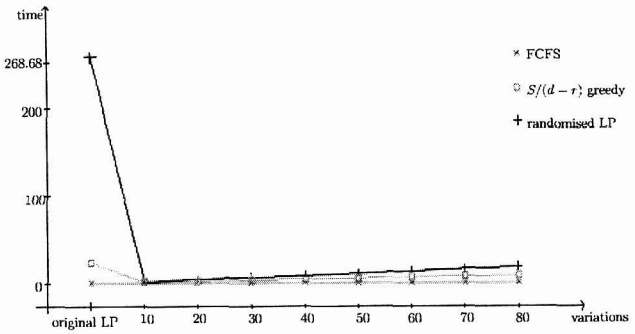

Figure 4. Influence of the number of variations on the computation time ($t_{av} = 0.1$).

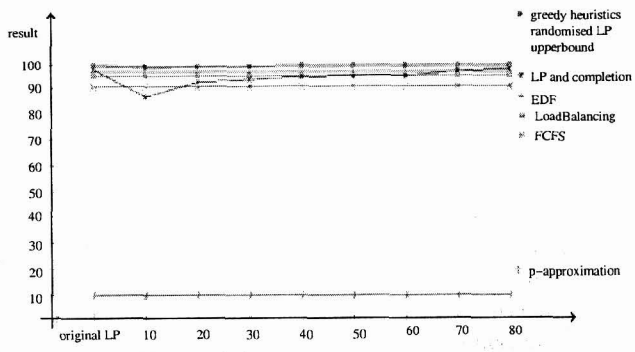

Figure 5. Influence of the number of variations on the objective function ($t_{av} = 0.1$).

- 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, on we call "optimal" over upper bound of the optimal solution, computed by blocks of 1,000 requests.

  Figure 6 shows that the performance is better if the system is less loaded (which is quite normal). The two greedy algorithms have good re-

sults, around 75% of the optimal solution, 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).

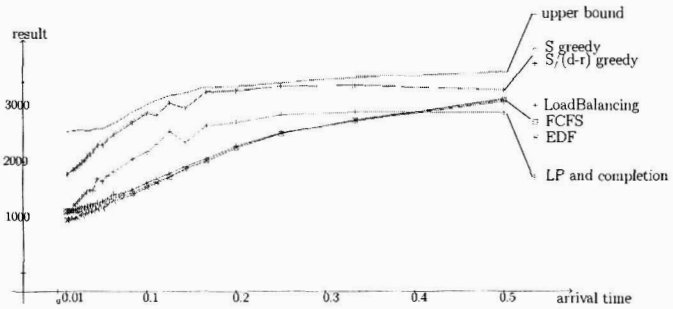

Figure 6. Influence of the average arrival time t_{av} on the scheduled requests number.

7 Conclusion

This work extends 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 and online problems. 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] F. Baille. *Algorithmes d'approximation pour des problèmes d'ordonnancement bicritères : application à un problème d'accès au réseau.* PhD thesis, 2005.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] M. Gallet, Y. Robert, and F. Vivien. Scheduling communication requests traversing a switch: complexity and algorithms. Research Report 2006-25, LIP, ENS Lyon, France, jun 2006.

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

[10] 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.

[11] 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.

[12] 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.