Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n° 5668

Optimizing Network Resource Sharing in Grids

Loris Marchal,
Yves Robert,
Pascale Vicat-Blanc Primet,
Jingdi Zeng

March 2005

Research Report N° 2005-10

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`

Optimizing Network Resource Sharing in Grids

Loris Marchal, Yves Robert, Pascale Vicat-Blanc Primet, Jingdi Zeng

March 2005

Abstract

While grid computing reaches further to geographically separated clusters, data warehouses, and disks, it poses demanding requirements on end-to-end performance guarantee. Its pre-defined destinations and service criteria ease the performance control; however, expensive resources and equipments used by grid applications determine that optimal resource sharing, especially at network access points, is critical. From the resource reservation perspective, this article looks at communication resources shared by grid sites. Two resource request scenarios have been identified, aiming at optimizing the request accept rate and resource utilization. The optimization problems, proven NP-complete, are then solved by heuristic algorithms. Simulation results, aside from showing satisfying results, illustrate the pros and cons of each algorithm.

Keywords: grid computing, communication resource, resource sharing, optimization.

Résumé

Le calcul distribué sur la grille requiert l'utilisation de clusters et de moyens de stockage distribués géographiquement, ce qui rend toute garantie de performance difficile à obtenir entre chacun des ces éléments. L'existence de chemins et de critères de services prédéfinis facilite quelque peu le contrôle de performance ; cependant, le coût d'utilisation des équipements et des ressources utilisés par les applications distribuées sur une telle grille fait que l'optimisation du partage de ressources est essentielle, particulièrement aux points d'accès du réseau. Nous nous intéressons ici au partage des ressources de communication par les différents sites de la grille de calcul, pour permettre la réservation de ressources. Nous isolons deux scénarios pour les requêtes, et cherchons à maximiser le taux d'acceptation des différentes requêtes ainsi que le taux d'utilisation des ressources de communication. Nous montrons que les différents problèmes d'optimisation sont NP-complets, et proposons des heuristiques pour les résoudre. Nous comparons les performances de ces différentes heuristiques par simulation.

Mots-clés: Calcul sur la grille, ressources de communication, partage de ressources, optimisation.

1 Introduction

Grid computing is a promising technology that brings together geographically distributed resources. Grids aggregate a large collection of resources(e.g., computing, communication, storage, information, etc.) to build a very high-performance computing environment for data-intensive or computing-intensive applications [1].

Grid applications, such as distance visualization, bulk data transfer, and high-end collaborative environment, have diverse and demanding performance requirements [2]; for instance, the coordinate management of network, storage, and computing resources, dynamically control over QoS and application behaviors, and advance resource reservation. Analyses [3] have shown that grids demand broad service quality, such as guaranteed delivery of huge data files [4], TCP throughput predictability, and data delivery stability.

The underlying communication infrastructure of grids, moreover, is a complex interconnection of LANs and WANs that introduces potential bottlenecks and varying performance characteristics [5]. For instance, the interface between LAN and WAN, considering grid sites may generate large flows thought their gigabit interfaces, introduces resource sharing bottleneck. Herein, provisioning end-to-end services with known and knowable characteristics of grids, which spans multiple administrative and technological domains, is critical.

An approach to tackle this problem is network resource reservation [6]. While computational/storage resource sharing/scheduling has been intensively investigated for grids [7, 8, 9, 10] during the past years, surfacing, is the idea of incorporating network/communication resource management into grid environments.

Based on the Grid 5000 project [11], an experimental grid platform gathering 5000 processors over eight sites geographically distributed in France, this article centers on network resource sharing. The rest of the article is organized as follows. Section 2 gives the system model and defines optimization problems for network resource sharing. Section 3 proves that the optimization problem is NP-complete. Heuristics and simulation results are given in section 4 and section 5, respectively. Section6 presents related work. Finally, the article concludes in section 7.

2 System Model and problem definition

Derived from physical configuration of the Grid5000 network, the system model is a collection of LANs (that is, grid sites) interconnected over a well-provisioned WAN. They are connected through IP routers. The grid network middleware carries out the network resource reservation task and communicates with grid applications. The network core is assumed to have ample communication resources [**?**]. Here, the aggregated capacity of a LAN is larger than the capacity of its access point (i.e., the router), and the capacity of the network core is larger than the aggregated capacity of all access points.

Given a set of resource requests, one can separate grid sites into ingress and egress points: where the traffic requires to enter the network from, is the ingress point, and where the traffic requires to leave the network from, is the egress point. These points at the network edge, as depicted in Fig. 1, are where resource sharing bottlenecks present.

2.1 Resource requests

Resource requests, corresponding to different application scenarios, can be long-lived or short-lived. The difference is that short-lived requests have time windows specified, as detailed below.

Given the notation as follows:

- a set of requests $\mathcal{R} = \{r_1, r_2, \ldots, r_K\}$, with $bw(r)$ as the bandwidth demanded by request $r \in \mathcal{R}$.
- a set of ingress points $\mathcal{I} = \{i_1, i_2, \ldots, i_M\}$, with $B_{in}(i)$ as the capacity (i.e., bandwidth) of ingress point $i \in \mathcal{I}$.
- a set of egress points $\mathcal{E} = \{e_1, e_2, \ldots, e_N\}$, with $B_{out}(e)$ as the capacity (i.e., bandwidth) of egress point $e \in \mathcal{E}$.

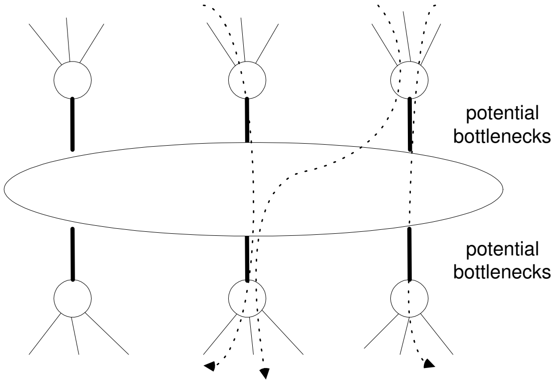


Figure 1: The system model that shows ingress and egress points of a network as potential bottlenecks.

For each request $r \in \mathcal{R}$, resource sharing constraints are stated as:

$$\begin{aligned} \forall i \in \mathcal{I}, \quad & \sum_{r \in \mathcal{R}, ingress(r)=i} bw(r) \quad \leqslant \quad B_{in}(i) \\ \forall e \in \mathcal{E}, \quad & \sum_{r \in \mathcal{R}, egress(r)=e} bw(r) \quad \leqslant \quad B_{out}(e) \end{aligned} \tag{1}$$

where $ingress(r) \in \mathcal{I}$ and $egress(r) \in \mathcal{E}$ are the ingress and egress point of request r, respectively.

For short-lived requests, more parameters are introduced as:

- each request $r \in \mathcal{R}$ has a starting time $t_s(r)$ and a finishing time $t_f(r)$. The time window of request r is then $[t_s(r), t_f(r)]$.
- Each request $r \in \mathcal{R}$ has its volume $vol(r)$ specified either in Bytes or other meaningful units.

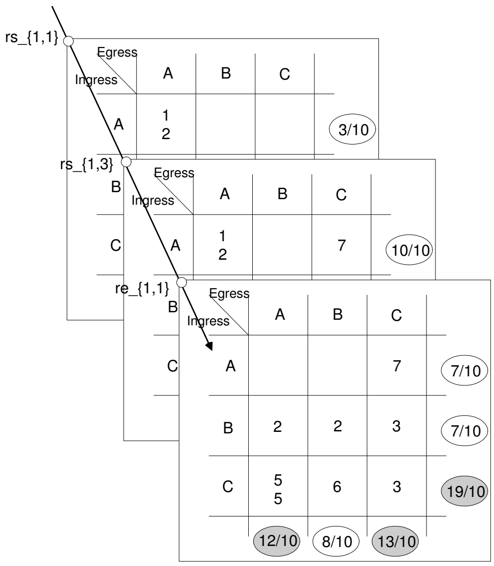


Figure 2: Short-lived requests interleaving with transmission windows.

An example of short-lived requests is depicted as in Fig. 2. It is formed on three dimensions, that is, ingress point, egress point, and time axis. The request starting and finishing times in the time axis are where resource assignment gets adjusted.

If request r is accepted at time $\sigma(r) = t$, both points of $ingress(r)$ and $egress(r)$ devote a fraction of their capacity, that is, $bw(r)$, to request r from time t to time $\tau(t) = t + \frac{vol(r)}{bw(r)}$. Obviously, the scheduled window of $[\sigma(r), \tau(r)]$ must be included in the time window of $[t_s(r), t_f(r)]$ for all requests $r \in \mathcal{R}$, that is,

$$\forall r \in \mathcal{R}, \quad t_s(r) \leqslant \sigma(r) < \tau(r) \leqslant t_f(r)$$

Applying to the short-lived requests with scheduled time window $[\sigma(r), \tau(r)]$, the resource constraints (1) are now restated as:

$$\begin{aligned} &\forall t,\ \forall i \in \mathcal{I}, \sum_{\substack{r\in\mathcal{R},\ ingress(r)=i,\\ \sigma(r)\leqslant t<\tau(r)}} bw(r) \leqslant B_{in}(i) \\ &\forall t,\ \forall e \in \mathcal{E}, \sum_{\substack{r\in\mathcal{R},\ egress(r)=e,\\ \sigma(r)\leqslant t<\tau(r)}} bw(r) \leqslant B_{out}(e) \end{aligned} \tag{2}$$

2.2 Optimization objectives

To formulate the optimization problem, x_k is defined as a boolean variable; it is equal to 1 if and only if request r_k is accepted. Provided with different types of requests and constraints specified in subsection 2.1, two optimization objectives are given as below:

MAX-REQUESTS Under the constraints in (1) or (2), one may maximize the ratio of the number of accepted requests to that of total requests. The objective function, referred to as MAX-REQUESTS, is:

$$\text{MAXIMIZE} \sum_{k=1}^{K} x_k$$

We summarize this into the following linear program:

$$\begin{array}{l} \text{MAXIMIZE } \sum_{k=1}^{K} x_k, \\ \text{UNDER THE CONSTRAINTS} \\ \left\{ \begin{array}{lll} \text{(3a)} & \forall i \in \mathcal{I}, & \sum\limits_{r_k\in\mathcal{R}, ingress(r_k)=i} x_k.bw(r_k) \leqslant B_{in}(i) \\ \text{(3b)} & \forall e \in \mathcal{E}, & \sum\limits_{r_k\in\mathcal{R}, egress(r_k)=e} x_k.bw(r_k) \leqslant B_{out}(e) \end{array} \right. \end{array} \tag{3}$$

RESOURCE-UTIL Under the same constraints, one may maximize the resource utilization ratio, that is, the ratio of granted resources to total resources. The objective function, referred to as RESOURCE-UTIL, is:

$$\text{MAXIMIZE} \frac{\sum_{k=1}^{K} x_k.bw(r_k)}{\frac{1}{2}\left(\sum_{i=1}^{M} B_{in}^{scaled}(i) + \sum_{e=1}^{N} B_{out}^{scaled}(e)\right)},$$

where the numerator $\sum_{k=1}^{K} x_k.bw(r_k)$ is the total bandwidth that has been assigned to requests. Since one bandwidth request is counted twice, that is, at both ingress and egress points, a factor of $1/2$ is used to "stretch" the utilization value to 1.

Furthermore, defined as

$$B_{in}^{scaled}(i) = \min\left(B_{in}(i), \sum_{r\in\mathcal{R}, ingress(r)=i} bw(r)\right)$$

and

$$B_{out}^{scaled}(e) = \min\Big(B_{out}(e), \sum_{r\in\mathcal{R}, egress(r)=e} bw(r)\Big),$$

$B_{in}^{scaled}(i)$ and $B_{out}^{scaled}(e)$ are adopted to rule out the possibility where one access point has no requests at all; thus, the capacity of this point shall be excluded when calculating resource utilization.

3 Problem Complexity

Since the linear program (3) involves integer (boolean) variables there is little hope that an optimal solution could be computed in polynomial time. Indeed, both optimization problems MAX-REQUESTS and RESOURCE-UTIL turn out to be NP-complete, as shown in the rest of the section.

The decision problem associated to the MAX-REQUESTS problem is the following:

Definition 1 (MAX-REQUESTS-DEC). *Given a problem-platform pair $(\mathcal{R}, \mathcal{I}, \mathcal{E})$ and a bound Z on the number of request to satisfy, is there a solution to the linear program 3 such that $\sum_{k=1}^{K} x_k \geqslant Z$?*

Theorem 1. MAX-REQUESTS-DEC *is NP-complete.*

Proof. Clearly, MAX-REQUESTS-DEC belongs to NP; we prove its completeness by reduction from 2-PARTITION, a well-known NP-complete problem [12]. Consider an instance B_1 of 2-PARTITION: given n integers $\{a_1, a_2, \ldots, a_n\}$, is there a subset I of indices such that $\sum_{i\in I} a_i = \sum_{i\notin I} a_i$? Let $S = \sum_{i=1}^{n} a_i$ and assume, without loss of generality, that $1 \leqslant a_i \leqslant S/2$ for $1 \leqslant i \leqslant n$. We build the following instance B_2 of MAX-REQUESTS-DEC:

- There are $K = 2n$ requests in $\mathcal{R}$, and $bw(r_k) = bw(r_{k+n}) = a_k$ for $1 \leqslant k \leqslant n$.
- There are $M = 2$ ingress points and $N = n$ egress points. For ingress points we let $B_{in}(i_1) = B_{in}(i_2) = S/2$. For egress points we let $B_{out}(e_k) = a_k$, $1 \leqslant k \leqslant n$.
- We let $ingress(r_k) = i_1$, $ingress(r_{k+n}) = i_2$, and $egress(r_k) = egress(r_{k+n}) = e_k$ for $1 \leqslant k \leqslant n$.
- Finally, we let $Z = n$. In other words, we aim at satisfying half of the requests.

The size of B_2 is polynomial (and even linear) in the size B_1. We have to show that B_1 has a solution if and only if B_2 has a solution.

Assume first that B_1 has a solution. Let I be the subset of $\{1, 2, \ldots, n\}$ such that $\sum_{i\in I} a_i = \sum_{i\notin I} a_i = S/2$. We claim that we can satisfy the $|I|$ requests $r_k, k \in I$ together with the $n - |I|$ requests $r_{k+n}, k \notin I$, thereby achieving the desired bound $Z = n$. Indeed, we schedule the first $|I|$ request from ingress point i_1, and the remaining $n-|I|$ ones from i_2, without exceeding their capacity $B_{in}(i_1) = B_{in}(i_2) = S/2$. Egress point e_k is used either for request r_k if $k \in I$, or for request r_{k+n} if $k \notin I$; in either case, $B_{out}(e_k) = a_k$ is equal to the requested bandwidth for the request.

Conversely, assume now that B_2 has a solution. Let I be the set if indices k such that r_k is satisfied and $1 \leqslant k \leqslant n$. Similarly, let J be the set of indices such that r_{k+n} is satisfied and $1 \leqslant k \leqslant n$. Because the capacity of egress point e_k is $B_{out}(e_k) = a_k$, I and J must be disjoint: if they shared an index, the capacity of the corresponding egress point would need to be twice larger than it is. Because the bound $Z = n$ is achieved, we have $|I| + |J| \geqslant n$. We deduce that I and J form a partition of $\{1, 2, \ldots, n\}$. We have $\sum_{k\in I} a_k \leqslant S/2$ because the capacity of ingress point i_1 is not exceeded, and $\sum_{k\in J} a_k \leqslant S/2$ because the capacity of ingress point i_2 is not exceeded. But $I \cup J = \{1, 2, \ldots, n\}$ and $\sum_{k=1}^{n} = S$, hence $\sum_{k\in I} a_k = \sum_{k\notin I} a_k = S/2$. We have found a solution to B_1. □

Proposition 1. *The decision problem associated to* RESOURCE-UTIL *is NP-complete.*

Proof. For the sake of brevity, we do not formally state the decision problem associated to RESOURCE-UTIL, but the definition should be obvious. To prove the proposition, we use the previous reduction: it can easily be checked that we achieve a full utilization of each resource (both ingress and egress points) if and only if there is a solution to the 2-PARTITION original instance. □

There are two sources of heterogeneity in the MAX-REQUESTS problem: the capacities $B_{in}(i)$ and $B_{out}(e)$ of the ingress/egress points may be different, as well as the bandwidths $bw(r)$ demanded by the requests. To fully assess the complexity of the problem, it is interesting to ask whether the MAX-REQUESTS-DEC problem remains NP-complete in the case of an uniform network (all ingress/egress capacities are equal)? if yes, does it remain NP-complete for an uniform network and uniform requests (all request bandwidths are equal)? The answers to these questions are given in the following proposition:

Proposition 2. *For an uniform network ($B_{in}(i) = B$ for all $i \in \mathcal{I}$ and $B_{out}(e) = \overline{B}$ for all $e \in \mathcal{E}$),* MAX-REQUESTS-DEC *remains NP-complete. But for an uniform network and uniform requests ($bw(r) = b$ for all $r \in \mathcal{R}$), the optimal solution of* MAX-REQUESTS *can be computed in a polynomial time.*

Proof. For the first part of the proposition, we start by observing that the restriction of MAX-REQUESTS-DEC still belongs to NP. For the completeness, we use a reduction from 2-PARTITION-EQUAL, a well-known NP-complete variation of 2-PARTITION [12]. Consider an instance B_1 of 2-PARTITION-EQUAL: given n integers $\{a_1, a_2, \ldots, a_n\}$, where n is even, is there a subset I of $n/2$ indices such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$? So in this variation of 2-PARTITION, the two subsets with equal sum must have had the same cardinal.

Let $S = \sum_{i=1}^{n} a_i$ and assume (without loss of generality) that $1 \leqslant a_i \leqslant S/2$ for $1 \leqslant i \leqslant n$. We construct an instance B_2 which has some similarities with the one used in the proof of Theorem 1:

1. First we scale the integers a_i as
$$a'_i \leftarrow a_i + S$$
and we compute $S' = \sum_{i=1}^{n} a'_i = (n+1)S$. The rationale behind this scaling is that $\sum_{i \in I} a'_i = S'/2$ can only occur if the set I has cardinal $n/2$.

2. We keep the two ingress points i_1 and i_2 with the same capacity $B = S'/2$. We augment the capacity of the n egress points $e_1, \ldots, e_n$ so that $B_{out}(e_k) = 2S + 1 = \overline{B}$ for $1 \leqslant k \leqslant n$. We keep the same set of $2n$ requests (with the new value a'_k for the bandwidth of r_k and r_{n+k}, $1 \leqslant k \leqslant n$).

3. We add n new ingress points $i_3, \ldots, i_{n+2}$, all of capacity $B = S'/2$, and n new requests r_{2n+k}, $1 \leqslant k \leqslant n$. The intuitive idea is that there will be a new request from each new ingress point to each egress point, which will saturate its bandwidth if accepted together with another old request. Formally,
$$\mathit{ingress}(r_{2n+k}) = i_{k+2},\ \ \mathit{egress}(r_{2n+k}) = e_k,\ \ bw(r_{2n+k}) = 2S + 1 - a'_k$$

Finally we let $Z = 2n$, i.e., we ask whether it is possible to accept $2n$ requests. We now have a uniform network, and we can reproduce the main ideas of the proof of Theorem 1. The main trick is that egress e_k cannot accept both requests r_k and r_{n+k}, because $2a'_k \geqslant 2(1 + S) > \overline{B}$. Hence only one of them can be accepted, and this will be possible only if there is a solution to 2-PARTITION-EQUAL.

For the second part of the proposition, consider an instance of MAX-REQUESTS with an uniform network ($B_{in}(i) = B$ for all $i \in \mathcal{I}$ and $B_{out}(e) = \overline{B}$ for all $e \in \mathcal{E}$) and uniform requests ($bw(r) = b$ for all $r \in \mathcal{R}$). Without loss of generality, we can assume that b evenly divides B and $\overline{B}$, and thus, after proper scaling, that $b = 1$. We claim that the solution of the linear program (3) can be computed in polynomial time. Indeed, the constraints (1) and (1) now write $AX \leqslant C$, where:

- A is a matrix of size $(N + M) \times K$. There are N rows for the ingress points, followed by M rows for the egress points. There is a column for each request $r_k \in \mathcal{R}$. In fact, A is a sub-matrix of the incidence matrix of the complete bipartite graph connecting the set of ingress points to the set of egress points.

- X is a vector of size K, its k-th component is x_k

- C is a vector of size $N+M$, whose first N components are equal to B and whose last M components are equal to $\overline{B}$.

Because the incidence matrix of a bipartite graph is totally unimodular (Theorem 5.24 in [13], the integer linear program (3) can be solved in polynomial time (Theorem 5.19 in [13]). This completes the proof. □

Since the problems have been proven to be NP-complete, heuristics are pursued to solve the problem defined in Section 2. Different approaches are taken, as explained in Section 4 and Section 5, respectively. The simulation results are also given, as a means of studying and comparing the performance of different heuristics.

4 Polynomial heuristics and simulations for long-lived requests

Three polynomial heuristics are proposed for both optimization objectives MAX-REQUESTS and RESOURCE-UTIL.

4.1 Growing the set of accepted requests

Based on classical greedy algorithm where requests are accepted until there are no more available resources, MAXREQ-SIMPLE sorts requests by bandwidth in a non-decreasing order (ties are broken arbitrarily). A request is accepted if and only if its requested bandwidth does not exceed the available capacity of both ingress and egress points. See Algorithm 1, where $\mathcal{A}$ is the set of accepted requests.

```
MAXREQ-SIMPLE (R, I, E)
  SortedRequests ← requests r_k ∈ R sorted by non-decreasing value of bw(r_k) A ← ∅
  for each request r ∈ SortedRequests do
    if bw(r) ≤ min(B_in(ingress(r)), B_out(egress(r))) then
      A ← A ∪ {r}
      B_in(ingress(r)) ← B_in(ingress(r)) − bw(r)
      B_out(egress(r)) ← B_out(egress(r)) − bw(r)
  return A
```

Algorithm 1: The simple greedy algorithm MAXREQ-SIMPLE

MAXREQ-REFINED refines the previous procedure, by accepting the request that leaves the maximum amount of resources to others. Take request r_k as an example. Let $i = ingress(r_k)$, and let $alloc\_ingress(i)$ be bandwidth of point i which has been taken by accepted requests (initially $alloc\_ingress(i) = 0$). By calculating the utilization ratio of ingress point i, that is, $\frac{alloc\_ingress(i)+bw(k)}{B_{in}(i)}$, and that of the corresponding egress point, the request that minimizes this ratio is accepted. See Algorithm 2, where $\mathcal{A}$ is the set of accepted requests.

4.2 Peeling off the set of original requests

Starting from the whole set of requests (i.e., the set of accepted requests $\mathcal{A} = \mathcal{R}$), MAXUSEPEELING "peels off" certain requests until a solution meeting all resource constraints is found. Given the set of requests, an occupancy ratio defined as $ratio(i) = \frac{\sum_{r \in \mathcal{A}, ingress(r)=i} bw(r)}{B_{in}(i)}$ is calculated for all access points. If all ratios are smaller than 1, all requests are accepted. Otherwise, among requests whose ingress and egress points both have their occupancy ratio bigger than 1, the one that helps decrease the ratio the most is peeled off; requests, either of whose ingress or egress points has a ratio bigger than 1, are scanned through in a similar manner. This heuristic is detailed in Algorithm 3.

4.3 Simulation settings

It is assumed that there are 50 ingress and egress points, respectively. The capacity of each point is randomly chosen as either 1Gb/s or 10Gb/s. Requests may occur between any pair of different points, and its bandwidth request is randomly chosen from a set of values: {10MB/s, 20MB/s, . . . , 90MB/s, 100MB/s,

```
MaxReq-Refined (R, I, E)
  A ← ∅
  continue ← true
  for each ingress point i ∈ I do
    alloc_ingress(i) ← 0
  for each egress point e ∈ E do
    alloc_egress(e) ← 0
  while (R ≠ ∅) and continue do
    for each request r ∈ R do
      cost(r) ← max( (alloc_ingress(ingress(r))+bw(k)) / B_in(ingress(r)) , (alloc_egress(egress(r))+bw(k)) / B_out(egress(r)) )
    select r_min such that cost(r_min) ⩽ cost(r) for all r ∈ E
    if (cost(r_min) > 1) then
      continue ← false
    else
      R ← R \ {r}
      A ← A ∪ {r}
      alloc_ingress(ingress(r)) ← alloc_ingress(ingress(r)) + bw(r)
      alloc_egress(egress(r)) ← alloc_egress(egress(r)) + bw(r)
  return A
```

Algorithm 2: The refined greedy algorithm MAXREQ-REFINED

200MB/s, ..., 900MB/s, 1000MB/s$\}$. The number of requests is determined by the system load, which is defined as the ratio of the sum of demanded bandwidth and the sum of available bandwidth in the system:

$$load = \frac{\sum\limits_{r \in \mathcal{R}} bw(r)}{\frac{1}{2}\left(\sum\limits_{i \in \mathcal{I}} B_{in}(i) + \sum\limits_{e \in \mathcal{E}} B_{out}(e)\right)}$$

In the simulation, we consider both over-loaded scenarios, with load close to 200%, and cases where the load is very low (down to 20%).

4.4 Simulation results and discussion

The simulation results for long-lived requests are illustrated in Figure 3.

Obviously, MAXREQ-SIMPLE and MAXREQ-REFINED, aiming at accepting as many requests as possible, outperforms MAXUSEPEELING with respect to the accept rate. And MAXUSEPEELING achieves better utilization ratio because it targets at optimizing the resource utilization. The original purposes of these heuristics have been met.

One may argue that none of the strategies reaches 100% acceptance rate or utilization ratio. The reason is that randomly generated requests in the article are not uniformly distributed among access points. It is not rare that certain point are heavily loaded, and certain points are not. The plotted accept rate and utilization ratio, which are more than 50%, are actually rather satisfying.

5 Polynomial heuristics and simulations for short-lived requests

As illustrated in subsection 2.1, if request r with time window $[t_s(r), t_f(r)]$ is accepted at time $\sigma(r) = t$, a fraction of system capacity, that is, $bw(r)$, is scheduled to request r from time t to time $\tau(t) = t + \frac{vol(r)}{bw(r)}$. Assume that time constraints are rigid, that is, $\sigma(r) = t_s(r)$ and $\tau(r) = t_f(r)$. Requests are then accepted or rejected as they are.

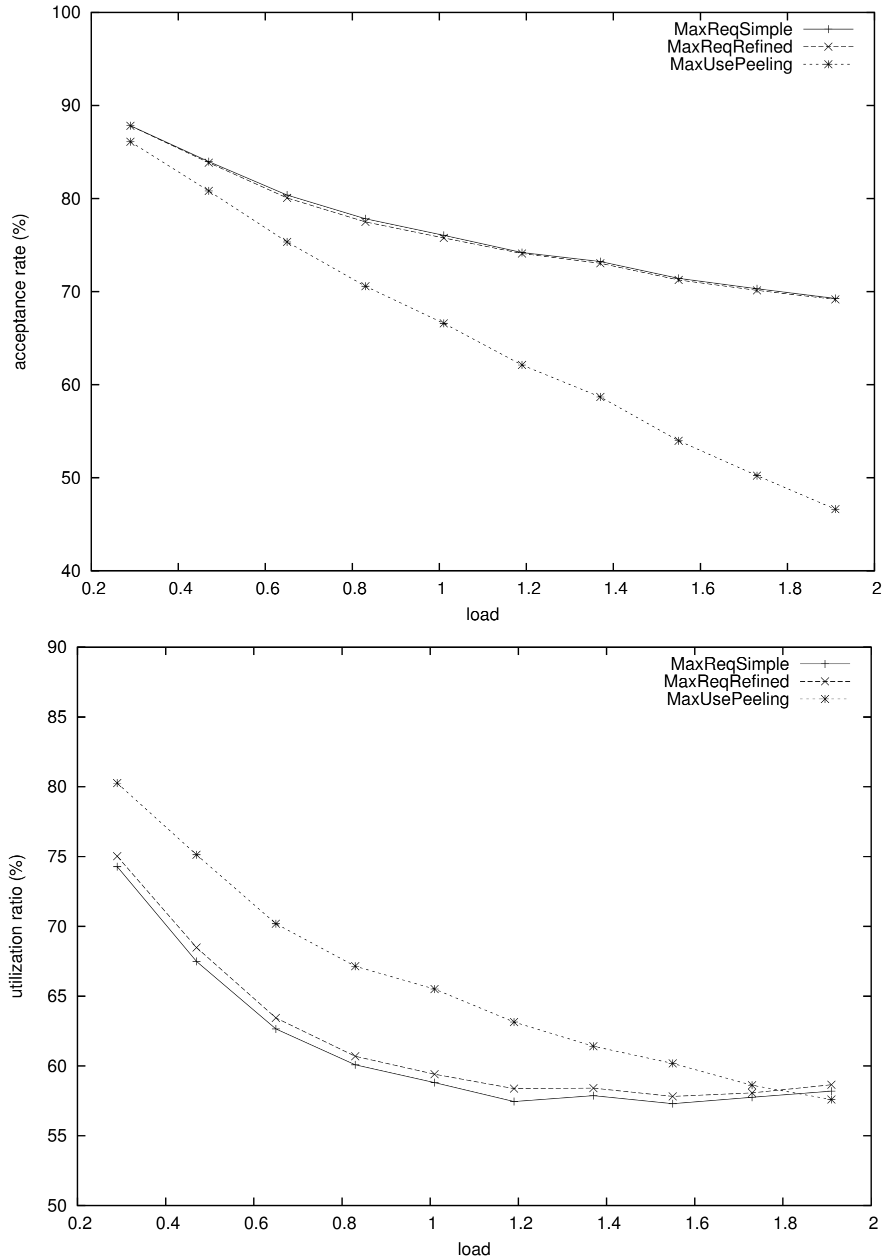


Figure 3: Comparison of the heuristics for long-lived requests.

```
MAXUSEPEELING (R,I,E)
  A ← R
  SaturatedIngresses ← {i ∈ I such that ratio(i) > 1}
  SaturatedEgresses ← {e ∈ E such that ratio(e) > 1}
  while SaturatedIngresses ≠ ∅ or SaturatedEgresses ≠ ∅ do
    {first, look for a request between two saturated points}
    E ← {r ∈ A with ingress(r) ∈ SaturatedIngresses and egress(r) ∈ SaturatedEgresses}
    if E is not empty then
      find the request r_0 in E with the maximum value
      of max{ratio(ingress(r_0)), ratio(egress(r_0))}
    else
      {choose the most saturated (ingress or egress) point}
      find p ∈ I* ∪ E* such that ratio(p) = max{max_{i∈I} ratio(i), max_{e∈E} ratio(e)}
      find the request r_0 such that:
      • ingress(r_0) = p or egress(r_0) = p
      • and bw(r_0) is maximum
    {now suppress this request and update the system}
    suppress the request r_0 from A
    if ingress(r_0) ∈ SaturatedIngresses and ratio(ingress(r_0)) ⩽ 1 then
      suppress ingress(r_0) from SaturatedIngresses
    if egress(r_0) ∈ SaturatedEgresses and ratio(egress(r_0)) ⩽ 1 then
      suppress egress(r_0) from SaturatedEgresses
```

Algorithm 3: The MAXUSEPEELING heuristic. We recall the that *ratio* is the ratio between the demanded bandwidth on a given point over its capacity: $ratio(i) = \frac{\sum_{r\in\mathcal{A},\ ingress(r)=i} bw(r)}{B_{in}(i)}$ for an ingress point i, and $ratio(e) = \frac{\sum_{r\in\mathcal{A},\ egress(r)=e} bw(r)}{B_{out}(e)}$ for an egress point e.

Note that, sharing the same complexity characteristics with long-lived ones, resource sharing optimization for short-lived requests is also NP-complete.

5.1 FIFO

Scheduling requests in a "first come first serve" manner, the FIFO heuristic accepts requests in the order of their starting times. If several requests happen to have the same starting time, the request demanding the smallest bandwidth is scheduled first.

5.2 Time window decomposition

With rigid time windows, pre-defined starting and finishing times are used as reference points for resource scheduling. As depicted in Figure 4, these time points naturally form time intervals within which no request starts or stops; thus heuristics for long-lived requests in Section 4 can be applied. Given intervals $[t_0, t_1], [t_1, t_2], \ldots, [t_{i-1}, i_N]$, therefore, for each t_i, there exists a request r such that $t_s(r) = t_i$ or $t_f(r) = t_i$. The greedy strategies proposed in Section 4 are then applied to each time-interval, with two situations explained in the following paragraphs.

For a request that spreads over multiple time intervals, first, if it gets rejected in its first time interval, it will be discarded permanently; second, if it gets accepted in its first time interval, it shall be granted certain priority when competing with other requests in its future time intervals.

Taking the duration of a request and the scheduling decisions in previous time intervals into consideration, a *priority* factor is used to represents the importance of scheduling request r on a given time-interval. Assume requests in time-intervals $[t_0, t_1]$, $[t_1, t_2]$,..., $[t_{i-1}, t_i]$ have been scheduled, At the interval of $[t_i, t_{i+1}]$, the *priority* factor is defined as the sum of the time already allocated to the request $(t_i - t_s(r))$ and the duration of the current interval $(t_i - t_{i-1})$ over the total request duration, that is,

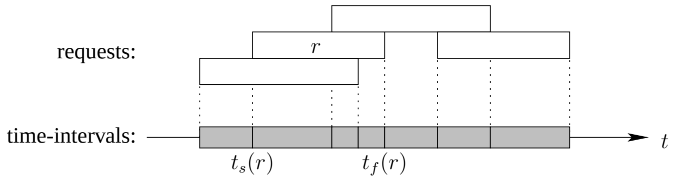


Figure 4: Decomposition of requests with time windows.

$$priority(r, [t_i, t_{i+1}]) = \frac{t_{i+1} - t_s(r)}{t_f(r) - t_s(r)}$$

The cost factor defined in the MAXREQ-REFINED heuristic for long-lived requests, is then refined as follows:

$$cost(r, [t_i, t_{i+1}]) \quad = \quad \frac{bw(r)}{b_{\min} \times priority(r, [t_i, t_{i+1}])}$$

where $b_{\min} = \min \left\{ B_{in}(ingress(r)), B_{out}(egress(r)) \right\}$

By adopting this cost factor, for requests with the same starting time, a higher priority is given to requests with smaller duration; it maximizes the accepted number of requests. For requests within the same time interval, a higher priority is given to requests that have been granted more resources. The complete heuristic CUMULATED-SLOTS is detailed in Algorithm 4.

```
CUMULATED-SLOTS (R, I, E)
  TimeIntervals ← {t_s(r), t_f(r) for some r ∈ R}
  sort TimeIntervals and remove duplicated dates
  take the first element t_1 of TimeIntervals
  while TimeIntervals is not empty do
    take the first element t_2 of TimeIntervals
    {we work on the interval [t_1, t_2]}
    for each ingress i in I do
      alloc_ingress(i) ← 0
    for each ingress e in E do
      alloc_egress(e) ← 0
    ActiveRequests ← {r ∈ R, such that t_s(r) ⩽ t_1 and t_f(r) ⩾ t_2}
    for each request r in ActiveRequests do
      cost(r) ← vol(r) / (min{B_in(ingress(r)), B_out(egress(r))} × (t_{i+1} − t_s(r)))
    sort ActiveRequests by non-decreasing value of cost
    for each request r in ActiveRequests do
      if alloc_ingress(ingress(r)) + bw(r) ⩽ B_in(ingress(r))
      and alloc_egress(egress(r)) + bw(r) ⩽ B_out(egress(r)) then
        allocate request r on interval [t_1, t_2]
        alloc_ingress(ingress(r)) ← alloc_ingress(ingress(r)) +
        bw(r)
        alloc_egress(egress(r)) ← alloc_egress(egress(r)) + bw(r)
      else
        remove request r from all previous intervals
        remove request r from R
```

Algorithm 4: The heuristic for short-lived requests

Following the same time window decomposition technique, two variants of the previous heuristic, that is, MINBW-SLOTS and MINVOL-SLOTS, are proposed with re-defined cost factor $cost(r, [t_i, t_{i+1})] = bw(r)$ and $cost(r, [t_i, t_{i+1})] = vol(r)$, respectively.

5.3 Simulation settings

Again, we are willing to test our heuristics and compare their performances by simulation. The framework of these simulations is close the one for long-lived requests, but have some particularities.

The platform is now composed of 10 ingress and 10 egress point, with the same capacity of 1GB/s. Ingress/egress points and bandwidth of the requests are generated as previously. Some new parameters of the requests still have to be instantiated:

- The volume is randomly chosen between 100GB and 1TB. As the possible bandwidths go from 10MB/s to 1GB/s, the duration of the requests may vary from a couple of minutes to about one day.
- Starting times are chosen according to a Poisson law. To reach a meaningful load of the platform, the parameter of this law can vary from 20 to 80 seconds.

As in the long-lived case, we want to compute the load of the platform. We have to change this definition, so as to take the duration of the requests into account:

$$load = \frac{\displaystyle\sum_{r \in \mathcal{R}} vol(r)}{\left(\displaystyle\sum_{i \in \mathcal{I}} B_{in}(i)\right) \times T}$$

where T is the total duration of the schedule. However, with this definition, the "clean-up" phase has a huge importance in the definition of the load, as shown in Figure 5. In this graph, representing the sum of all requested bandwidth over the time, we can consider two phases: a heavy-load phase where lots of requests are produced, and a "clean-up" phase, where we wait for long requests to terminate. We want to compare the performances of our strategies in the first phase, so we modify the definition of the termination time T used in the computation of the load:

$$T = \max_{r \in \mathcal{R}} t_s(r) - \min_{r \in \mathcal{R}} t_s(r).$$

So the "long tail" of the second phase is not taken into account anymore.

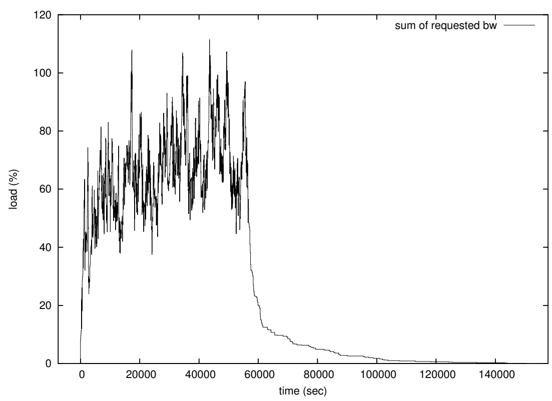


Figure 5: Variable load for short-lived requests.)

5.4 Simulation results and discussion

As illustrated in Figure 6, first, FIFO shows poor performance on both accept rate and utilization ratio. The fact that FIFO lets requests block each other indicates that selectively reject is an important step towards good performance. Second, MINVOL-SLOTS does not perform as well as MINBW-SLOTS and CUMULATED-SLOTS. In fact, accepting a request with the minimum volume may not always be a good decision. If the time window is small, the request will likely take the majority of the bandwidth; this lowers the value of the accept rate and thus the utilization ratio. Last, CUMULATED-SLOTS and MINBW-SLOTS have very close performance. CUMULATED-SLOTS should have good performance because its decision is made based on both demanded bandwidth and resource reservation in the past; it prevents a request from being rejected in the late stage of its time window. MINBW-SLOTS accepts the requests with smaller bandwidth requirements; these requests are unlikely to be rejected later, unless other requests with small bandwidth demand surges at one point. Under some circumstances, MINBW-SLOTS performs as well as CUMULATED-SLOTS, even without resource reservation history.

6 Related Work

Admission control mechanisms in IP networks are well-developed [14]. They have been mostly done at the ingress points of the network edge, or is closely coupled with feasible path search. The work in this article, however, looks at both access points where the traffic enters and leaves the network. Besides, the specific network topology studied in this article does not pose significant requirements on routing.

Studying control mechanisms at network edge, this work is in line with the Internet philosophy of pushing the complexity to the network edge. On the perspective of resource scheduling, it pursues solutions based on the idea of what enters the network shall be able to leave the network, that is, the idea of avoiding potential packet drop within the network. This idea of "globally max-min fair" was investigated in Network Border Patrol [15], a core-stateless congestion avoidance mechanism.

Advance reservation for grids has also been under intensive study. The Globus Architecture for Reservation and Allocation (GARA) provides advance reservations and end-to-end management for QoS on different type of resources (network, storage and computing) [6]. A QoS architecture that combines resource reservation and application adaptation has been proposed. The work in this article fits in this context, but further explores the optimization on network resource sharing, based on a specific topology.

The advance reservation problem has also been defined and investigated in [16]. Although both targeting at resource requests with starting and finishing time limits, the work in this article looks at optimal resource sharing over a network with resource bottlenecks occurring at the edge, rather than investigating on impacts of the percentage of book-ahead periods and that of malleable reservations on the system.

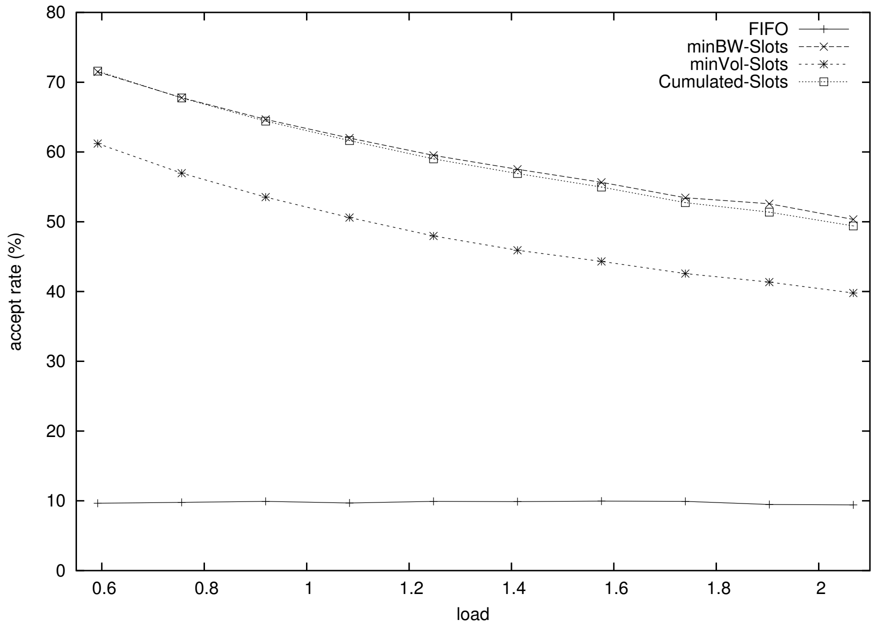
7 Conclusions

Network resource sharing in grids has been investigated in this article. With bottlenecks presented at the network edge, network resources are reserved based on the concept of what enters the network shall be able to leave the network. For both long-lived and short-lived requests, optimization objectives with respect to request accept rate and resource utilization are pursued. Proven to be NP-complete, the optimization problems are solved with heuristics. The heuristic algorithms are studied and compared by simulations.

Resource sharing optimization studied in this article can be extended to other similar environments, for example, community overlay networks. The resource sharing strategies can be carried out either in a centralized or distributed manner, depending on network management implementations. Future work will be continued in the direction of reliving tentative hot spots in the network, that is, ingress/egress points that are heavily demanded, and in the direction of real-time resource reservation.

References

[1] I. Foster, *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2004.



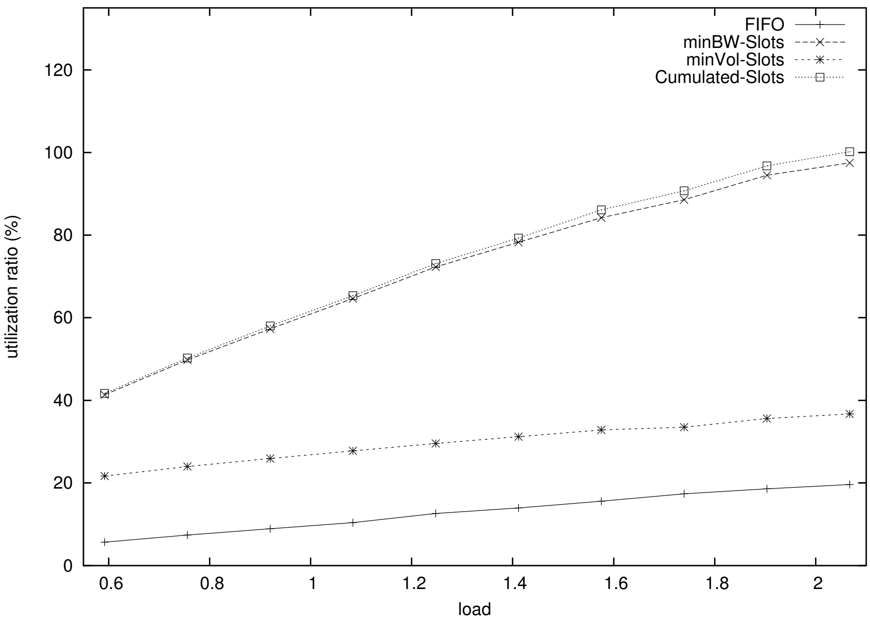


Figure 6: Comparison of the heuristics according to two different metrics

[2] V. Sander, W. Allcock, C. Pham, I. Monga, P. Padala, M. Tana, and F. Travostino. (2003, June) Networking issues of grid infrastructures. Grid working draft, Grid High Performance Networking Research Group (GHPN-RG), Global GRID Forum. [Online]. Available: http://forge.gridforum.org/projects/ghpn-rg/document/draft-ggf-ghpn-netissues-0/en/1/draft-ggf-ghpn-netissues-1.pdf

[3] P. Vicat-Blanc/Primet, “High performance grid networking in the datagrid project,” *special issue Future Generation Computer Systems*, Jan. 2003.

[4] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, “The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets,” *Journal of Network and Computer Applications*, vol. 23, pp. 187–200, 2001.

[5] S. Floyd and V. Jacobson, “Link-sharing and resource management models for packet networks,” *IEEE/ACM Transaction on Networking*, vol. 3, pp. 365–386, Aug. 1995.

[6] I. T. Foster, M. Fidler, A. Roy, V. Sander, and L. Winkler, “End-to-end quality of service for high-end applications,” *Computer Communications*, vol. 27, no. 14, pp. 1375–1388, 2004.

[7] T. Roblitz, F. Schintke, and A. Reinefeld, “From clusters to the fabric: the job management perspective,” in *Proc. IEEE the International Conference on Cluster Computing*, 2003, pp. 468–473.

[8] K. Ranganathan and I. Foster, “Decoupling computation and data scheduling in distributed data-intensive applications,” in *Proc. IEEE the 11th Symposium on High Performance Distributed Computing(HPDC'02)*, July 2002, pp. 352–358.

[9] K. Czajowski, I. Foster, and C. Kesselman, “Resource co-allocation in computational grids,” in *Proc. IEEE the eighth International Symposium on High Performance Distributed Computing*, Aug. 1999, pp. 219–228.

[10] H. Dail, F. Bern, and H. Casanova, “A decoupled scheduling approach for grid application development environment,” *International Journal Parallel and Distributed Systems*, vol. 63, pp. 505–524, 2003.

[11] The grid 5000 project. [Online]. Available: http://www.grid5000.org/

[12] M. R. Garey and D. S. Johnson, *Computers and Intractability, a Guide to the Theory of NP-Completeness*, 1979.

[13] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, ser. Algorithms and Combinatorics 21. Springer-Verlag, 2002, second edition.

[14] V. Firoiu, J. L. Boudec, D. Towsley, and Z. Zhang, “Theories and models for internet quality of service,” *Proceedings of the IEEE*, vol. 90, pp. 1565–1591, Sept. 2002.

[15] C. Albuquerque, B. Vickers, and T. Suda, “Network border patrol: Preventing congestion collapse and promoting fairness in the internet,” *IEEE Transactions on Networking*, vol. 12, pp. 173–186, Feb. 2004.

[16] L. Burchard, H.-U. Heiss, and C. A. F. D. Rose, “Performance issues of bandwidth reservations for grid computing,” in *Proc. IEEE the 15th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'03)*, Nov. 2003, pp. 82–90.