Optimal Bandwidth Sharing in Grid Environments

Loris Marchal, Pascale Vicat-Blanc Primet, Yves Robert and Jingdi Zeng
Laboratoire de l'Informatique du Parallélisme, UMR CNRS-ENS Lyon-INRIA-UCB Lyon 5668
École Normale Supérieure de Lyon, France

Abstract

We consider the problem of bulk data transfers and bandwidth sharing in the context of grid infrastructures. Grid computing empowers high-performance computing in a large-scale distributed environment. Network bandwidth, which makes the expensive computational and storage resources work in concert, plays an active role on carrying grid applications traffic. Due to specific traffic patterns and application scenarios, grid network resource management encounters new challenges. From the bandwidth sharing perspective, this article looks at network bandwidth shared among computing and storage elements. Referred to as short-lived, grid data requests with transmission window and volume are scheduled in the network. By manipulating the transmission window, the request accept rate and network resource utilization are to be optimized. The formulated optimization problem is proven NP-complete. Associated with proposed heuristics, simulations are carried out to illustrate the pros and cons of each bandwidth sharing strategy and its application scenarios. A tuning factor, that allows for adapting performance objective, is introduced to adjust network infrastructure and workload.

1. Introduction

Moving one step further from processor clusters, grid computing is a promising technology that brings together large collection of geographically distributed resources (e.g., computing, visualization, storage, information, etc.) to build a very high-performance computing environment for data-intensive or computing-intensive applications [10]. Grid applications involve multi-domain, very long distance heterogeneous networks, complex sets of network services, and local area networks belonging to independent organizations. Heterogeneous traffic flows of grids have impact on grid performance, resource utilization, and performance of individual applications.

The data volume of data grids is in the order of Terabytes and will likely reach Petabytes in the near future. Transporting such enormous quantities of data among grid networks poses specific challenges on the transport protocol and control mechanisms. Data transfer protocols [1, 2], which extend the standard FTP protocol, include features of various existing grid storage systems. They provide security and support parallel, striped, partial, and third-party transfers. But these tools inherit the major issue of the underlying transport protocol, namely the TCP protocol, that reacts poorly to bulk transfer when links present large bandwidth delay product and very strong bottlenecks [21].

How to allocate bandwidth to flows is a central and classical issue in networking. This issue has been well studied [18] within the TCP/IP context. The main assumption in the Internet, that makes the TCP congestion control algorithm robust and meaningful, is that the source access rates are generally much smaller ($c = 2Mbit/s$ for DSL lines) than the bottleneck capacity ($C = 2,5Gbit/s$, say); the link is not a bottleneck until demand attains around 99% of link capacity. In such an environment, the *Max-Min* fairness algorithm [4], that is giving all flows the opportunity to make use of all the available capacity in a "fair" way, is the goal of statistical bandwidth sharing strategies. But it has been shown, in novel Internet usage scenarios [19], that in overloaded networks, performance deteriorates rapidly. Moreover, it is also not uncommon for the transfers to fail entirely, because the TCP connections time out due to packet losses. The TCP protocol is intensively studied to work better in this context. But the proposed alternatives to the congestion control algorithm still keep the bandwidth sharing philosophy and objective function of the Internet, and have great difficulty in dealing with

1-4244-0307-3/06/$20.00 ©2006 IEEE.

high dynamic heavy traffic in large rate-delay product contexts and in presence of deep bottlenecks. Pro-active admission control is the only approach likely to preserve performance [9].

In other respects, bandwidth sharing and transfers scheduling in grids have to be coupled with the management of other types of grid resources. The effective scheduling of jobs in large scale distributed systems is a complex task, and network bandwidth has been identified as one of the primary parameters that affect the performance [17]. To simplify, we can say that the completion time of typical datagrid applications is given by the sum of the execution time and of the time taken to transfer the data they need. In most cases, data transfer time often dominates completion time. For bulk data grid applications, moving terabytes (and sometimes even petabytes) of data in the shortest and most predictable possible amount of time is then of great interest. Network bandwidth sharing then surfaces as a part of the grid resource management.

This paper proposes original solutions to control bandwidth sharing considering the specificity of the DataGrid context. Grid network resources are managed to ensure bandwidth reservation of the dataset movements. Three goals are pursued: (1) improving data transfer time predictability; (2) enhancing transfer reliability; and finally (3) improving transfer performance. A lightweight and easy-to-deploy control plane that is complementary to the data plane is introduced, based on an overlay network approach. In other words, we propose to reconsider the bandwidth sharing optimisation objective in the grid context, where the network model is specific and where transfers are not as unpredictable as in the Internet, and have to be tackled in a global infrastructure perspective.

In the rest of the paper, it is assumed that moving data is easier than distributing and deploying application codes. We consider scheduling algorithms that allocate computing (i.e., CPUs) and storage (i.e., memory and disks) resources first, and then generate output as data transfer requests, where each request has a specified time-window and volume. The proposed system model hides the packet-level traffic and transport-level dynamics inside discrete data transfer requests. The bandwidth sharing problem is considered at the session level, thus greatly reducing the complexity of the system, and proposing an efficient alternative solution to the TCP dynamic issue in large delay product environments.

Associated with the Grid 5000 project [7], an experimental grid platform gathering 5000 processors over eight geographically distributed sites in France, this article centers on network resource sharing in grids.

The rest of the article is organized as follows. Section 2 gives the system model and defines optimization problems of bandwidth sharing. Problem complexity is discussed in Section 3. Heuristics and simulation results for short-lived rigid requests and short-lived flexible requests are given in Section 4 and 5 respectively. Section6 presents related work. The article concludes in Section 7.

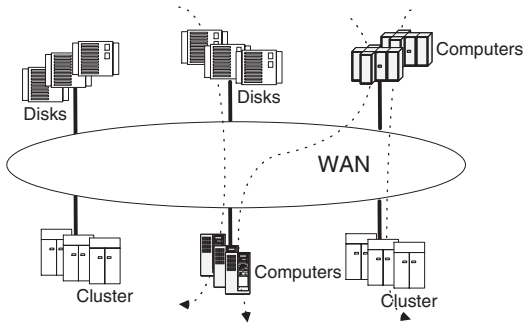

Figure 1. Ingress and egress points of the network.

2. System model and problem definition

The system is a collection of grid sites interconnected over a well-provisioned wide-area network. From the resilient overlay network (RON) to other related architectures, the overlay infrastructure is adopted to provide more control and functional flexibility to the network. The edge routers of the grid network, referred to as *grid overlay routers* in this article, are assumed to form a fully-meshed overlay. Grid network middleware, residing in these routers, controls the resource sharing and the transport of grid data. The network core is assumed to be lossless and queuing delay-free. There are M overlay routers, with N connections per site, as depicted in Figure 1. Bulk data transfers between two points are not symmetrical, hence up to $2N(M-1)$ bi-directional links can be attached to an overlay router. The number of connections among routers increases in the order of $O(MN)$.

The network core is assumed to have ample communication resources [20]. The aggregated capacity of a site is larger than the capacity of its access point (i.e.,

the router), the capacity of each node is in the same order as the access point, and the capacity of the network core is larger than the aggregated capacity of all access points.

Given a set of transmission requests, an ingress point is where the traffic requires to enter the network, and an egress point is where the traffic requires to leave the network. These points, as depicted in Figure 1, are where potential resource bottlenecks present.

2.1. Resource requests

Resource requests, corresponding to different application scenarios, can be long-lived or short-lived. Long-lived requests correspond to indefinite flows between grid users, while short-lived requests represent discrete data transfer tasks. The problem of long-lived request has been studied in [13]. Contrarily to the classical concept of flows that last indefinite time, in this article we deal with flows that represent finite-size large data transfers. We also use the term *short-lived requests* to denote such finite transfers. Short-lived requests have specified time-windows, and the scheduler must enforce induced constraints. The scheduling of short-lived requests can be difficult, due to their flexible time-windows and thus flexible bandwidth assignments.

Flows arrive at the network edge according to a Poisson distribution, and each flow is associated with a source and a destination. Flows are unidirectional, given the fact that grid traffic volume between two grid entities (storage and computing elements) is often asymmetrical.

We use the following notations:

- a set of requests $\mathcal{R} = \{r_1, r_2, \ldots, r_K\}$.
- 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 $e \in \mathcal{E}$.
- each request has a required transmission window of $[t_s(r), t_f(r)]$, and an assigned transmission window of $[\sigma(r), \tau(r)]$ when accepted.
- each request has its volume $vol(r)$ specified either in Bytes or other meaningful units.
- each request has the transmission limit of its attached host $MaxRate(r)$.
- each request, if accepted, has an assigned bandwidth $bw(r)$.

If request r is accepted at time $\sigma(r) = t$, both points $ingress(r)$ and $egress(r)$ devote a fraction of their capacity, that is, $bw(r)$, to request r from time t to time $\tau(t) = t + \frac{vol(r)}{bw(r)}$.

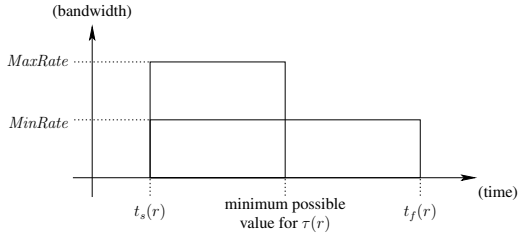

Figure 2. The flexible bandwidth assignment.

The flexibility of bandwidth assignment is illustrated as in Figure 2. For the sake of simplicity, whe choose not to changed the requested transmission starting time: in other words, the assigned staring time is $\sigma(r) = t_s(r)$. Provided with a manipulatable finishing time $t_f(r)$, the assigned bandwidth $bw(r)$ lies in the interval of $[MinRate(r), MaxRate(r)]$. Here, $MinRate(r)$ is determined by the requested time window:

$$MinRate(r) = \frac{vol(r)}{t_f(r) - t_s(r)}.$$

Obviously, the assigned finishing time $\tau(r)$ should not exceed the value of the requested finishing time $t_f(r)$. Accordingly, we have

$$\tau(r) = \sigma(r) + \frac{vol(r)}{bw(r)} = t_s(r) + \frac{vol(r)}{bw(r)} \leqslant t_f(r)$$

and

$$bw(r) \geqslant MinRate(r)$$

Moreover, the capacity of ingress or egress points implicates a limit on the number of scheduled requests. The resource sharing constraints are then stated as the following:

$$\begin{aligned}
&\forall t, \quad \forall i \in \mathcal{I}, \sum_{\substack{r\in\mathcal{R},\ ingress(r)=i,\\ \sigma(r)\leqslant t<\tau(r)}} bw(r) \leqslant B_{in}(i)\\
&\forall t, \quad \forall e \in \mathcal{E}, \sum_{\substack{r\in\mathcal{R},\ egress(r)=e,\\ \sigma(r)\leqslant t<\tau(r)}} bw(r) \leqslant B_{out}(e),\\
&\forall r, \quad MinRate(r) \leqslant bw(r) \leqslant MaxRate(r) \qquad (1)
\end{aligned}$$

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

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. Using the notations provided in Subsection 2.1, we define the following first two optimization objectives:

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

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

RESOURCE-UTIL Under the same constraints, one may aim at maximizing 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 "scale" the utilization value between 0 and 1.

Furthermore, we have defined

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

and

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

$B_{in}^{scaled}(i)$ 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.

2.3. Another optimization objective for flexible requests

When a request r is such that $MinRate = MaxRate$, there is no choice to assign its bandwidth: when accepting r, we need to enforce $bw(r) = MinRate(r) = MaxRate(r)$. In such a case, we say that the request is *rigid*. However, there are situations where we have more freedom: if the request time-window is large enough, we can decide between a whole range of admissible values of $bw(r)$ in the interval $[MinRate(r), MaxRate(r)]$. In such a case, we say that the request is *flexible*.

When we aim at optimizing for MAX-REQUESTS, the first optimization objective, accepted requests are very likely to be granted the minimum bandwidth $MinRate(r)$. However, grid computing applications may bring new elements into the decision making procedure. To fulfill a grid computing task, the CPU, storage, and network bandwidth resources have to be considered simultaneously. If a transmission task gets served faster than what it originally requests, it implies the earlier release of computing and storage resources. These resources will be returned to the available resource pool and can be used for other application requests. The application scenario of grid computing, therefore, suggests that assigning to flexible requests more bandwidth than the authorized minimum will benefit grid applications. Nevertheless, given the same amount of network resource, assigning more bandwidth will perhaps lessen the accept rate. What is the relationship between increased assigned bandwidth and decreased accept rate? What is the trade-of regarding performance gain?

Instead of assigning the requested bandwidth of $MinRate(r)$ to an accepted request, one may grant a fraction of the maximum bandwidth $MaxRate(r)$ that a grid user can utilize. Let f denote this fraction, then the value $f = 0.8$ guarantees 80% of $MaxRate(r)$ for each accepted request. The number of accepted requests, given the factor f, is to be maximized as follows:

$$\#guaranteed = \max\{r \in \mathcal{R}, \\ bw(r) \geqslant \max(f \times MaxRate(r), MinRate(r))\}$$

The factor f may be adopted as a single value that gives a reference on how much faster all requests are transmitted through the network. From a customer and service provider relationship perspective, customers now have two choices: they can stick with their requested network resource, that is, $MinRate$ (r), and have a better chance of being accepted at this time. They can also stand the risk of being rejected and try later, but take the advantage of being transmitted more quickly. Obviously, how much more quickly a request gets transmitted depends on the global grid system configuration and load.

3. Problem complexity

Scheduling problems are known to be difficult, and the one addressed in this paper is no exception. Schedul-

ing long-lived and short-lived rigid requests has already been proven NP-hard [13, 14], even in the case of an uniform network (same bandwidth for each ingress and egress port). However, it was also shown in [14] that the optimal solution for scheduling uniform long-lived requests ($bw(r) = b$ for all $r \in \mathcal{R}$) can be computed in a polynomial time.

In this section, we show that scheduling uniform short-lived flexible requests is NP-complete. This clearly shows the combinatorial nature of the problem, and the intrinsic complexity added by the possibility to route a request at different time-steps. We state the decision problem formally:

Definition 1 (MAX-REQUESTS-DEC). *Consider a problem-platform pair $(\mathcal{R}, \mathcal{I}, \mathcal{E})$ with uniform (unit-size) requests:*

$$\forall r \in \mathcal{R},\ bw(r) = MinRate(r) = MaxRate(r) = 1$$

For each request $r \in \mathcal{R}$, the transmission window $[t_s(r), t_f(r)]$ is known at the beginning (off-line problem). Given a bound K on the number of requests to satisfy, is there a feasible solution to such that at least K requests are accepted?

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

It is worth noting that if the platform reduces to a single ingress-egress pair, the problem is polynomial (a greedy algorithm is optimal).

Proof. Clearly, MAX-REQUESTS-DEC belongs to NP; we prove its completeness by reduction from 3-DM (3-Dimensional Matching), a well-known NP-complete problem [12]. Consider an instance B_1 of 3-DM: given $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_n\}$, and $Z = \{z_1, z_2, \ldots, z_n\}$ three disjoint sets of same cardinal n, and given a set of triples $T \subseteq X \times Y \times T$, does T contain a matching T', i.e. a set of n triples such that no two elements of T' agree in any coordinate? We build the following instance B_2 of MAX-REQUESTS-DEC:

- There are $M = n+1$ ingress points and $N = n+1$ egress points. For ingress points we let $B_{in}(i) = 1$ if $1 \leqslant i \leqslant n$ and $B_{in}(n+1) = n-1$. For egress points, we let $B_{out}(e) = 1$ if $1 \leqslant e \leqslant n$ and $B_{in}(n+1) = n-1$. Both points $B_{in}(n+1)$ and $B_{out}(n+1)$ are called *special*, while the $2n$ other points are called *regular*.
- There are $|T| + 2n(n-1)$ requests in $\mathcal{R}$, and $bw(r) = 1$ for all $r \in R$. Each of the first $|T|$ requests is called *regular* and is associated to a triple in T, while the remaining $2n(n-1)$ requests are called *special*.
- For each triple $(x_i, y_j, z_k) \in T$, we define a regular request r as follows: we let $B_{in}(r) = i$, $B_{out}(r) = j$ and $[t_s(r), ft(r)] = [k, k+1]$. In other words, there is no flexibility for regular requests; if accepted, r must be scheduled at time $\sigma(r) = k$.
- Special requests involve one special point and are flexible. More precisely, for each ingress point i we define $n-1$ identical requests r such that $B_{in}(r) = i$, $B_{out}(r) = n+1$ and $[t_s(r), ft(r)] = [1, n+1]$. Similarly, for each ingress point e we define $n-1$ identical requests r such that $B_{in}(r) = n+1$, $B_{out}(r) = e$ and $[t_s(r), ft(r)] = [1, n+1]$. Therefore, a special request can be scheduled at any time-step between 1 and n.
- Finally, we let $K = n + 2n(n-1)$.

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 T' be the matching. For each time-step k, $1 \leqslant k \leqslant n$, there is a single triple in T' whose last coordinate is z_k. Let (x_i, y_j, z_k) be this triple, and r its associated regular request. Together with r, we route $2(n-1)$ special requests at step k, one from each ingress point except i, and one to each egress point except j. Because T' realizes a permutation in the first and second coordinates of its triples, each regular point is active exactly $n-1$ times during the n scheduling steps. All regular points will thus accept all their $n-1$ special requests. Together with the n regular request (one per step), we have accepted K requests, hence a solution to B_2.

Conversely, assume now that B_2 has a solution. $K = n+2n(n-1)$ request are accepted during the n scheduling steps. But at a given step, no more than $2n-1$ requests can be accepted, and this is only feasible if $2n-2$ of them are special requests. Therefore, at each step exactly $2(n-1)$ special requests and one regular request are accepted. Let T' be the set of the triples associated to these n regular requests, we claim that T' is the desired matching. By construction, no two triples of T' agree in the third coordinate. Assume that two triples would share the same first coordinate, say x_i. This means that ingress point i is activated at two different time-steps for two regular requests. At most $n-2$ special requests with ingress i will be accepted. But this is a contradiction, because all special requests are accepted (there are $2n(n-1)$ of them, and $2(n-1)$ are accepted at each step). For the second coordinate the reasoning is identical. We have found a solution to B_1. □

4. Polynomial heuristics and simulations for short-lived rigid requests

As proved in Section 3, the optimization problem formulated in Section 2 is NP-complete. Heuristics are then needed to solve the problem. 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.

4.1. FCFS

Scheduling requests in a "first come first serve" manner, the $FCFS$ 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.

4.2. Time window decomposition

To make the problem more tractable, we would like to be able to *slice* the whole scheduling problem into time-intervals such that no requests starts or stop during one interval. Pre-defined starting and finishing times are used as reference points for resource scheduling (see Figure 3. Given intervals $[t_0, t_1], [t_1, t_2], \ldots, [t_{i-1}, i_N]$, for each t_i, there exists a request r such that $t_s(r) = t_i$ or $t_f(r) = t_i$. The FCFS strategy is then applied to each time-interval, with two situations explained in the following paragraphs.

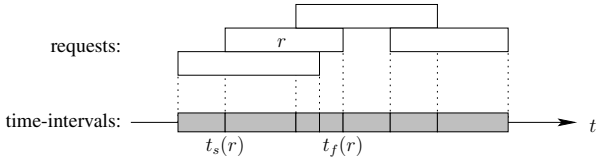

Figure 3. Decomposition of requests with time windows.

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 represent the importance of a scheduling request r on a given time-interval. Assume requests in time-intervals $[t_0, t_1]$, $[t_1, t_2]$, $\ldots$, $[t_{i-1}, t_i]$ have been scheduled, For interval $[t_i, t_{i+1}]$, 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 $(t_{i+1} - t_i)$ over the total request duration, that is,

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

To each request we associate a *cost* factor defined 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 is detailed in Algorithm 1, where $ali(i)$ denotes the amount of bandwidth which is currently allocated for ingress $i \in \mathcal{I}$, and which should never exceed $B_{in}(i)$ (and similarly $ale(e)$ for $e \in \mathcal{E}$).

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.

4.3. Simulation settings

There are 10 ingress and 10 egress points, respectively, with a capacity of $1GB/s$. Requests may occur between any pair of different points, and their volumes are randomly chosen from a set of values: {100G, 200GB, ..., 90GB, 100GB, 200GB, ..., 900GB, 1TB}. Following the Poisson distribution, the request arrival rate determines the system load. The number of requests is determined by the system load, which is defined as the ratio of the sum of demanded bandwidth and of the sum of available bandwidth in the system:

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

CUMULATED-SLOTS ($\mathcal{R}, \mathcal{I}, \mathcal{E}$)
  $TimeIntervals \leftarrow \{t_s(r), t_f(r)$ for some $r \in \mathcal{R}\}$
  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 $\mathcal{I}$ **do** $ali(i) \leftarrow 0$
    for each egress e in $\mathcal{E}$ **do** $ale(i) \leftarrow 0$
    $ActiveRequests \leftarrow \{r \in \mathcal{R},$
        $t_s(r) \leqslant t_1$ and $t_f(r) \geqslant t_2\}$
    sort $ActiveRequests$ by non-decreasing $cost$
    for each request r in $ActiveRequests$ **do**
      if $ali(ingress(r)) + bw(r) \leqslant B_{in}(ingress(r))$ **and** $ale(egress(r)) + bw(r) \leqslant B_{out}(egress(r))$ **then**
        allocate request r on interval $[t_1, t_2]$
        $ali(ingress(r)) \leftarrow ali(ingress(r)) + bw(r)$
        $ale(egress(r)) \leftarrow ale(egress(r)) + bw(r)$
      else
        remove request r from all previous intervals
        remove request r from $\mathcal{R}$

Algorithm 1: The time-window heuristic for rigid requests.

4.4. Simulation results and discussion

As illustrated in Figure 4, first, FIFO shows very poor performance on both accept rate (10%) and utilization ratio (under 20%). 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.

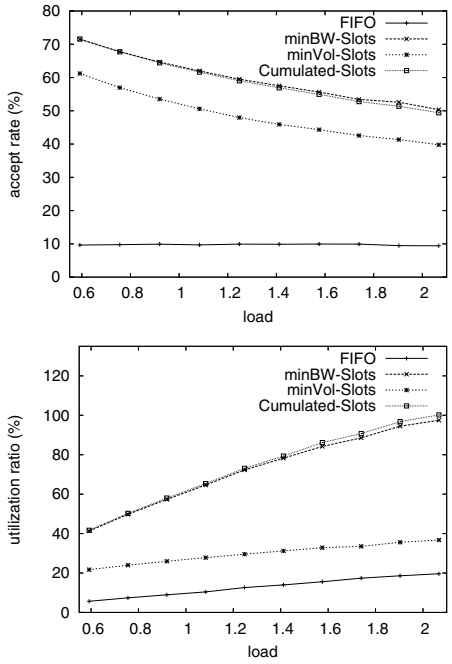

Figure 4. Comparison of the heuristics according to two different metrics

5. Polynomial heuristics and simulations for short-lived flexible requests

In this case, we consider that requests have flexible time-windows. It is then possible to start scheduling a request some time after its arrival, while matching the deadline. Given that the starting time and finishing time are not violated, that is, $t_f(r) > t_s(r) + \frac{vol(r)}{bw(r)}$, the constraint on the assigned bandwidth is stated as $bw(r) \geqslant MinRate(r)$. Algorithms that adopt different bandwidth assignment policies, either granting $MinRate(r)$ to each accepted request, or ensuring $\max(MinRate(r), f \times MaxRate(r))$ for a prescribed tuning factor f, are introduced in the following sections. One major characteristic of all our proposed heuristics here is that they are *on-line*. We take decisions either on the fly (on a pure greedy basis) or after a short delay (scheduling within each time interval). There is no need for the *a priori* knowledge of the whole set of requests.

The heuristics can be classified according to the decision procedure:

FCFS Requests are accepted or rejected on a first-come first-serve basis. If two requests have the same arrival time $t_s(r)$, we schedule the one with the smallest $MinRate(r)$.

Interval-based Requests are accepted or rejected within consecutive time intervals. These intervals have the same length. The scheduler considers all the requests whose arrival time lies within the current interval. More requests are expected to be processed in longer intervals; this leaves more space for optimization, at the price of a longer response time for grid users.

5.1. FCFS heuristics

Both proposed greedy heuristics schedule requests as soon as they arrive. However, they assign different bandwidth to each accepted request r: one is the minimum rate $MinRate(r)$ as originally requested by the user, the other one is a prescribed fraction $f \times MaxRate(r)$. The parameter f varies in experiments. To simplify the notation, we use

$$bw(r) \leftarrow \text{BANDWIDTHASSIGNALG}(r),$$

where BANDWIDTHASSIGNALG denotes any of the previous bandwidth assignment strategies.

The pseudo code of FCFS heuristics is shown in Algorithm 2, where t_{begin} and t_{end} denote the times at which execution begins and ends. $\mathcal{A}$ is the set of accepted requests.

```
GREEDY(R, I, E)
  A ← ∅ for each ingress i in I do ali(i) ← 0 for
  each egress e in E do ale(i) ← 0
  for t = t_begin to t = t_end do
    if t = t_f(r) for some request r ∈ A then
      {reclaim bandwidth allocated to r}
      ali(ingress(r)) ← ali(ingress(r)) − bw(r)
      ale(egress(r)) ← ale(egress(r)) − bw(r)
    if t = t_s(r) for some request r ∈ A then
      {try to schedule r}
      bw(r) ← BANDWIDTHASSIGNALG(r)
      i ← ingress(r)
      e ← egress(r)
      if (ali(i) + bw(r) ⩽ B_in(i)) and (ale(e) +
      bw(r) ⩽ B_out(e) then
        A ← A ∪ {r}
        ali(i) ← ali(i) + bw(r)
        ale(e) ← ale(e) + bw(r)
  return A
```

Algorithm 2: FCFS heuristics for flexible requests.

5.2. Interval-based heuristics

Interval-based heuristics do not schedule requests as soon as they arrive. Instead, they take decisions every time step t_{step}. The execution is thus divided into time intervals of length t_{step}. At the end of each interval $[t, t + t_{step}[$, scheduling decisions are taken for all *candidate* requests, i.e. request r whose arrival time lie in the interval: $t \leqslant t_s(r) < t + t_{step}$. As for the FCFS heuristics, we keep track of the bandwidth $ali(i)$ and $ale(e)$ already allocated on each ingress and egress ports. The first thing to do is to reclaim the bandwidth assigned to accepted requests r whose execution is finished in the previous interval, i.e. requests satisfying $t - t_{step} \leqslant t_f(r) < t + t_{step}$. Then we compute a *cost* associated with each candidate request. The intuition is to balance the resource assignments among access points. A request of high cost is likely to saturate its ingress or egress point, thereby hindering the possibility of scheduling more requests later.

The cost of a request r is computed as the following. Let $i = ingress(r)$, $e = egress(r)$, and $bw(r)$ the bandwidth assigned to r if accepted. As before, $bw(r)$ will be either the minimum rate $MinRate(r)$ or a prescribed fraction $f \times MaxRate(r)$. If r is accepted, the utilization rate of its ingress point i becomes $\frac{ali(i)+bw(r)}{B_{in}(i)}$, and that of its egress point e becomes $\frac{ale(e)+bw(r)}{B_{out}(e)}$. We define the cost of r as the maximum value of these two quantities, that is,

$$cost(r) = \max\left(\frac{ali(i) + bw(r)}{B_{in}(i)}, \frac{ale(e) + bw(r)}{B_{out}(e)}\right)$$

The candidate request with minimum cost will be accepted. The pseudo code of the interval-based heuristics is shown in Algorithm 3.

5.3. Simulations and discussions for short-lived flexible requests

In this section, simulations are carried out to illustrate and compare the performance of the heuristics given in the previous section. The simulated grid network and load is similar to the framework for short-lived rigid requests. The transmission time varies from a couple of minutes to about one day, by randomly generating bandwidth requests between 10MB/s and 1GB/s. As described in Section 2.2, the optimization objective is the accept rate, that is, the number of accepted requests over the number of total requests.

A heavy loaded scenario is illustrated as in Figure 5. The average arrival time of requests vary from 0.1 to

WINDOW($\mathcal{R}, \mathcal{I}, \mathcal{E}$)
$\mathcal{A} \leftarrow \emptyset$ **for** each ingress i in $\mathcal{I}$ **do** $ali(i) \leftarrow 0$ **for** each egress e in $\mathcal{E}$ **do** $ale(i) \leftarrow 0$
for $t = t_{begin}$ to $t = t_{end}$ by step t_{step} **do**
{reclaim bandwidth of finished requests}
for each request $r \in \mathcal{A}$ s.t. $t - t_{step} \leqslant t_f(r) < t$ **do**
$ali(ingress(r)) \leftarrow ali(ingress(r)) - bw(r)$
$ale(egress(r)) \leftarrow ale(egress(r)) - bw(r)$
{determine set of candidate requests}
$\mathcal{C} \leftarrow \emptyset$
for each request $r \in \mathcal{R}$ s.t. $t \leqslant t_s(r) < t + t_{step}$ **do**
$\mathcal{C} \leftarrow \mathcal{C} \cup \{r\}$
$bw(r) \leftarrow$ BANDWIDTHASSIGNALG(r)
{schedule candidate requests}
$continue \leftarrow true$
while ($\mathcal{C} \neq \emptyset$) and $continue$ **do**
select r_{min} such that $cost(r_{min}) \leqslant cost(r)$ for all $r \in \mathcal{C}$
if ($cost(r_{min}) > 1$) **then**
$continue \leftarrow false$
else
$\mathcal{C} \leftarrow \mathcal{C} \setminus \{r\}$
$\mathcal{A} \leftarrow \mathcal{A} \cup \{r\}$
$ali(ingress(r)) \leftarrow ali(ingress(r)) + bw(r)$
$ale(egress(r)) \leftarrow ale(egress(r)) + bw(r)$
return $\mathcal{A}$

Algorithm 3: Interval-based heuristics for flexible requests.

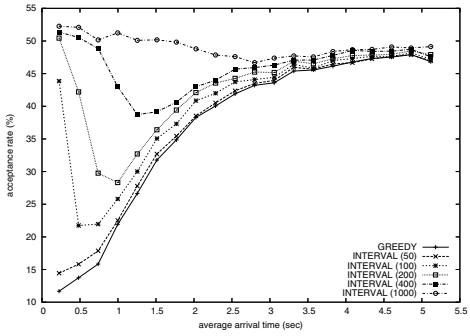

Figure 5. Comparison of FCFS and Interval-based (with different window lengths) heuristics in heavy loaded context

5 seconds. The bandwidth assignment policy assigns $f \times MaxRate(r)$ with $f = 1$. The simulation results show that in a very loaded network, the interval-based heuristics achieves a better accept rate than FCFS. And for the interval-based algorithms, the larger the time interval, the better the request accept rate.

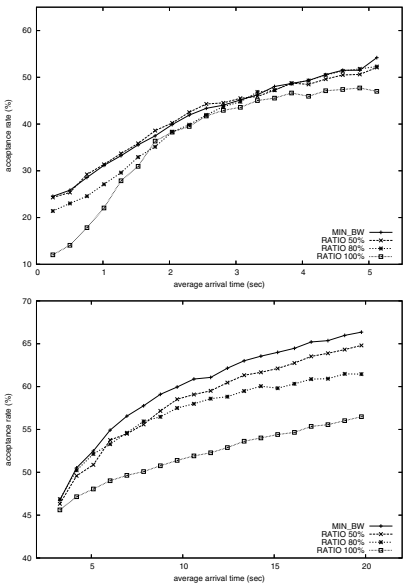

Figure 6. Performance of the FCFS heuristic with different bandwidth allocation policies (f factor) under heavy loaded conditions (left) or underloaded conditions (right)

Figure 6 presents the performance of the FCFS heuristic. The network is less loaded (average arrival time goes from 3 to 20 seconds). Different bandwidth assignment policies apply: either the minimum bandwidth to every accepted request (MIN_BW), or a fixed ratio f of the maximum transmission rate $MaxRate$. As expected, a smaller bandwidth to each request results in more accepted requests, especially when the network is not too much loaded. This is no longer true, however, for heavy loaded networks. For example, assigning a request the maximum rate of the its user leads to a smaller transmission time, thus the corresponding ingress/egress points are freed to other requests more quickly. The same set of simulation is run for the interval-based heuristics, and the results are depicted in Figure 7. The same conclusions as for the greedy heuristics hold, except that we obtain slightly better results for small values of the average arrival time.

These simulations show that greedy and interval

based heuristics have similar performance when the network is not heavily loaded. Results show an average of an acceptance rate of 50% (ie. with bandwidth guarantee) for both strategies. The advance knowledge of requests does not improve the system. In a busy network as the one discussed in this article, interval-based approach improves a lot the accept rate while greedy strategy have an acceptance rate less than 20%. Furthermore, we show that the longer the interval, the better the accept rate. Longer intervals imply chances of having a better knowledge and more requests to schedule; it thus provides more room for the scheduling algorithm to optimize the bandwidth sharing. With large scheduling windows, more than 50% of bandwidth requests for bulk data transfers will be guaranteed, while it has been observed in such an overloaded context that concurrent high speed TCP flows have great difficulties in obtaining bandwidth, and that the largest flows suffer denial of service. In these conditions, bulk transfers often fails before ending. The proposed control may improve transfer reliability (goal 2) while insuring predictability of transfer time (goal 1).

We study the behavior of the tuning factor f through simulations. Between the value of 0 and 1, the tuning factor supplies requests an opportunity of actually obtaining more bandwidth for improving transfer duration (goal 3) and then application completion time. We understand that by pushing requests out of the network at an earlier time, the network may accommodate more requests in the future. Moreover, assigning more bandwidth to each request will certainly decrease the number of accepted requests. Simulation results illustrate the dynamic between the tuning factor and the refined request accept rate. Both greedy and interval-based strategy take advantage of this factor under very underloaded conditions. In these conditions, allocating 80% of the maximum bandwidth improves the acceptance rate by a factor $0,2$ (linear with $(1-f)$) while a value of $f = 0,5$ gives only a gain of $0,3$. This tuning factor enables the grid manager to adjust the global system with its own characteristics and the actual workload without modifying the bandwidth allocation strategy. See [15] for further details.

5.4. Implementation considerations

From the implementation point of view, this bandwidth sharing approach can reutilize most of the RSVP protocol features (client side and RSVP request format). The main difference lies in how the reservation requests are routed and processed. Our requests are routed within the grid overlay network from a client to his access router and then broadcast to all grid access routers implied in the request (egress points) rather within the flat IP network, according to the IP routing algorithm for the classical RSVP approach. Secondly, the decision is made by the local ingress access router, that returns directly a scheduled time window and allocated rate to the client. To enforce the allocation policy, lightweight mechanisms are studied: local bandwidth control on the client side (token bucket based) and high performance data flow control at access point level. We have experimented an hardware assist solution, based on INTEL IXP2400 network processors within the Grid5000 project. This control ensures that the bulk data flows are conform to the scheduling, and, if not, that they are automatically dropped so as not to hurt other well behaving TCP flows. Our first experiments show that new high speed TCP protocols, like BIC protocol, but also well tuned RENO protocols, in such controlled environments, are performing much better than [16]. Ensuring a stable bandwidth by an independent control plane, enables well tuned TCP flows to fully utilize their allocated capacity, to offer a predictable transfer delay to end applications, and to increase reliability of all transfers.

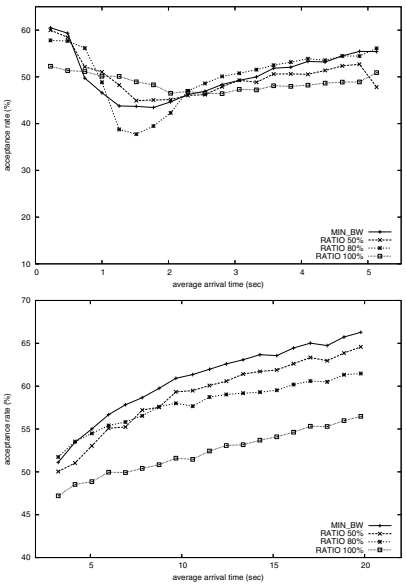

Figure 7. Performance of the Windows heuristic (with length 400) with different bandwidth allocation policies (f factor) under heavy loaded conditions (left) or underloaded conditions (right).

6. Related Work

Admission control and reservation in the Internet have been studied for real-time traffic and generally with immediate reservation, that is, QoS takes effect immediately and remains in effect for an indefinite duration. Consequently, traditional admission control and reservation algorithms [5] adopt greedy strategies. The RSVP protocol enables individual routers to optimize locally the usage of their reserved resources. In a grid context, QoS guarantees apply to TCP-dominated traffic, and a certain degree of isolation is required between connections in order to support performance guarantees without precluding multiplexing. Grid context presents also a great difference in terms of transfer durations (hours, days...) that are bounded, even though they are several orders of magnitude greater than those of the Internet. Flow transfer reliability is a very important issue, as other grid resources (CPUs, disks) have been scheduled, and a large amount of resources could be wasted when long transfer failure occurs. Finally, the grid network exhibits a specific topology, that is, heterogeneous and highly hierarchical. Ingress/egress links act both as natural aggregation points and constitute expected overloaded points as their capacity is in the same order of the access rate of the sources. Similar topology and bandwidth sharing problem are analyzed at different rate scales, in radio access networks. But in Grids we have to consider both sides of the network and the load matrix is given.

The fairness issue between short and long TCP flows [3], that is, between mice and elephants, gained wide attention. Besides, the sharing is closely coupled with routing path search. The work in this article, however, assumes that grid bulk data are separated from the rest of the traffic (mice). TCP/IP protocols [22] have been adapted to carry high-volume grid data applications over long distance. The reliability, RTT fairness, TCP-friendliness issues are in the center of the investigation. These TCP enhancements for large bandwidth-delay product paths focus on Internet context (max-min fairness). However, we consider the work on end-to-end protocol improvement could be of great interest in this controlled context. The routing path search has also been integrated into the picture. This article looks at grid network access points where the traffic enters and leaves the network. The requests have a predefined route from source to destination in this topology. The performance predictability is of more interest here.

While computational/storage resource sharing has been intensively investigated [8], the idea of incorporating network/communication resource management into grid environments has gained attention. For instance, network resource reservation [11] was proposed to be studied within the grid scope. The Globus Architecture for Reservation and Allocation (GARA) introduced the idea of advance reservations and end-to-end management for QoS on different types of resources (bandwidth, storage, and computing). Herein, grid middleware itself takes the network resource into account, if not from a perspective that is totally different from other applications. As proposed by the Grid High-Performance Networking (GHPN) group of Global Grid Forum (GGF), grid network service is defined based on the term of grid service; it integrates the network layer operations into grid applications. In line with this direction, this article further explores the optimization of bandwidth sharing given the specified grid network topology and traffic pattern. A similar bandwidth sharing problem has been investigated in [6]. Although this article also focus on resource requests with transmission time windows, it tackles optimal resource sharing, instead of investigating the impact of the percentage of book-ahead periods and that of malleable reservations on the system. The physical characteristics of optical medium makes it an excellent candidate for supporting grid bulk data applications [20]. Existing work centers on the feasible network architectures. Assuming a system model that complies to optical domain topology and conditions, however, the bandwidth sharing mechanisms of this article can be deployed as part of the optical burst switching intelligent management system.

7. Conclusions

Network bandwidth sharing in data grids is investigated in this article. With bottlenecks at the network edge, requests are scheduled based on the concept of what enters the network shall be able to leave the network. Referred to as short-lived requests, data transfer requests of grid applications are scheduled with respect to optimizing the request accept rate. The problem is proven NP-complete, and polynomial heuristics are investigated. These algorithms are studied and compared by simulations. We have demonstrated that such an approach may contribute to solve the problem of network resource allocation and performance optimisation in a high performance computing and data management context.

Along with other protocols and interfaces, the bandwidth sharing strategies studied here are going to

be integrated in the grid network middleware of the Grid 5000 project. They may work closely with the scheduling of other resources, such as computational and storage but also with optimized transport protocol and services. Heuristics and optimization objectives will be refined so as to take the specificity of different grid infrastructures and usages into account. Future work will be continued in the direction of relieving tentative hot spots in the network, that is, ingress/egress points that are heavily demanded, and in the direction of real-time resource reservation. We will also investigate further the implementation issues and will consider fully distributed allocation algorithms to study the scalability of the approach.

8. Acknowledgment

This work has been funded by the French Ministry of Education and Research, INRIA, and CNRS, via ACI GRID's Grid5000 project.

References

[1] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data management and transfer in high performance computational grid environments. *Parallel Computing*, 28:749–771, May 2002.

[2] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. The globus striped gridftp framework and server. In *Proceedings of the ACM/IEEE Symposium on Supercomputing (SC'05)*. IEEE Computer Society Press, 2005.

[3] K. Avrachenkov, U. Ayesta, P. Brown, and E. Nyberg. Differentiation between short and long TCP flows: Predictability of the response time. In *INFOCOM*. IEEE Communications Society Press, 2004.

[4] D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, 1987.

[5] R. Braden. Resource reservation protocol (rsvp) version 1 functional specification, 1997.

[6] 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)*, pages 82–90, Nov. 2003.

[7] F. Cappello, F. Desprez, M. Dayde, E. Jeannot, Y. Jegou, S. Lanteri, N. Melab, R. Namyst, P. Primet, O. Richard, E. Caron, J. Leduc, and G. Mornet. Grid'5000: A large scale, reconfigurable, controlable and monitorable grid platform. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing (Grid'2005)*. IEEE Computer Society Press, 2005.

[8] K. Czajowski, I. Foster, and C. Kesselman. Resource co-allocation in computational grids. In *Proc. IEEE 8th International Symposium on High Performance Distributed Computing (HPDC)*, pages 219–228. IEEE Computer Society Press, 1999.

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

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

[11] I. T. Foster, M. Fidler, A. Roy, V. Sander, and L. Winkler. End-to-end quality of service for high-end applications. *Computer Communications*, 27(14):1375–1388, 2004.

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

[13] L. Marchal, P. V. blanc Primet, Y. Robert, and J. Zeng. Optimizing network resource sharing in grids. In *IEEE Global Telecommunications Conference GlobeCom'2005*. IEEE Communications Society Press, 2005.

[14] 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.

[15] L. Marchal, P. Primet, Y. Robert, and J. Zeng. Scheduling network request with transmission window. Research Report 2005-31, LIP, ENS Lyon, France, July 2005.

[16] P. V.-B. Primet, T. Takano, Y. Kodama, T. Kudoh, O. Gluck, and C. Otal. Large scale gigabit emulated testbed for grid transport evaluation. In *In Proceedings of the International workshop on Protocols for Very Long Distance Networks (PFLDNET 2006)*, NARA, Japan, February 2006.

[17] 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)*, pages 352–358, July 2002.

[18] J. W. Roberts. A survey on statistical bandwidth sharing. *Computer Networks: The International Journal of Computer and Telecommunications Networking archive*, 45:319–332, June 2004.

[19] R. Sherwood, R. Braud and B. Bhattacharjee. Slurpie: A co-operative bulk data transfer protocol. In *INFOCOM*. IEEE Communications Society Press, 2004.

[20] L. L. Smarr, A. A. Chien, T. Defanti, J. Leigh, and P. M. Papadopoulos. The optiputer. *Communications of the ACM (special issue: blueprint for the future of high-performance networking)*, 46:58–67, Nov. 2003.

[21] M. Weltz, E. He, P. Vicat-Blanc Primet, and al. Survey of protocols other than TCP. Technical report, Global Grid Forum documents GFD 55, April 2005.

[22] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion control (BIC) for fast long-distance networks. In *INFOCOM*. IEEE Communications Society Press, 2004.