End-host based mechanisms for implementing Flow Scheduling in GridNetworks

Sebastien Soudan, Romaric Guillier, Pascale Primet
INRIA, Universite de Lyon
ENS-Lyon, 46 allee d'Italie
Lyon, France
ssoudan@ens-lyon.fr

ABSTRACT

In Grids, data transfers and network resources need to be managed in a more deterministic way than in the Internet. New approaches like flow scheduling are proposed and studied as alternatives to traditional QoS and reservation proposals. To enable such flow scheduling approaches, runtime mechanisms controlling flow sending time and rate have to be implemented in the data plane. This paper quantifies and compares such end-host based mechanisms combined with transport protocols to instantiate different scheduling strategies in a range of latency conditions. We show that, a single-rate scheduling strategy implemented by an AIMD-based protocol and a packet pacing mechanism offers predictable performance and is insensitive to latency. This paper also highlights the limits of other strategies and rate limitation mechanisms like token bucket which generates unpredictability and other drawbacks.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: General; C.4 [**Performance of Systems**]: Design studies

Keywords

Grid networks, flow scheduling, bulk data transfers, rate limitation, pacing

1. INTRODUCTION

High end instruments and applications generate huge amount of data which have to be moved between data centers, computing centers or visualization centers in a reliable and time-constrained manner [21]. Today, data are transferred within packet networks through transport services such as GridFTP based on the TCP protocol. TCP enables a fair sharing of the link capacities among contending flows by applying a distributed congestion control algorithm. Such end to end congestion control approach does not fill the transfer time predictability and reliability needs of these applications. Mechanism to control huge movements of data in the time axis seems to be highly required in Grids. Indeed, tight coordination of resource allocation among end points often requires a data mover service to carry out a giant task in a specified time interval.

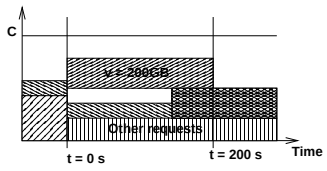


Figure 1: 200 GB scheduled transfer.

Let us consider the following example: 200 GB data produced in a site A need to be moved to a site B for processing. The CPU and disk resources in site B have been reserved in advance from 200 s to 400 s for the CPU, and from 0 s to 400 s for the disk. If the transfer can begin only when disk in site B is allocated, and if there is no pipeline between transfer and computing services, the bulk data transfer task r_1 needs to move 200 GB data from site A to site B in the time interval $[0\,s, 200\,s]$ (Figure 1), to fully use the CPU resources.

To manage transfer jobs and network resource sharing in a more deterministic way, researchers are studying new approaches all based on some resource reservation paradigm like dynamic light path provisioning or flow scheduling [14]. Bandwidth reservation has been studied extensively for real-time applications [2], which are often approximately modeled as reserving a fixed amount of bandwidth from a given start time. In comparison, bulk data transfer tasks are specified in terms of *volume* and *active window* (from arrival time to deadline). The bulk data transfer optimization problem has been formulated in networks in different ways [9, 3]. Given a network model, minimizing the congestion factor appears to be a powerful objective function [1]. In this situation, allocating bandwidth to flows in a fair manner, as it is the case in the Internet, is no more the main objective. Indeed, to complete more tasks before their deadlines, sharing instantaneous bandwidth fairly among all active flows may not be optimal [7]. In some cases, it is beneficial to allow a connection with larger pending volume and earlier deadline to grab more bandwidth, similar to the case of Earliest Deadline First scheduling in real-time systems [19].

Chen and Primet demonstrated in [1] that *spaghetti scheduling* which consists of allocating a single rate over the full time window of each request is an optimal solution for the

GridNets 2007 October 17-19, 2007, Lyon, France.

DOI 10.4108/gridnets.2007.2152

bulk data transfer scheduling (BDTS) problem, when all requests have the same time constraints. The allocated rate is obtained by dividing the volume by the maximal time window of the request. Indeed, this solution gives the minimal congestion factor. The flexibility of choosing *bandwidth allocation profile* can also be exploited to improve system performance, when the time-windows are different. The BDTS problem has an optimal solution with bandwidth reservation profile in the form of the step function with $O(r)$ intervals where r is the number of requests. The implementation of the multi-rate allocation strategy is an extension of the single rate allocation ones.

As bulk data transfer scheduling problem has been formulated and proved to be a valuable solution to optimize both user and network provider utility functions, this paper concentrates on implementation issues of such approach. Indeed, all proposed scheduling algorithms operating at a session abstraction layer rely on an ideal transport protocol able to fully utilize the controlled bandwidth. However it is not very clear how current transport protocols have to be combined with time and rate control mechanisms to approximate the ideal implementation of such approach. The goal of this paper is then to explore end host based mechanisms which can be used to have a better control on the bandwidth sharing in a scalable and simple way. We also study how window-based transport protocols interact with these mechanisms. We aim at evaluating the potential biases introduced at runtime. These biases will have to be integrated in the flow scheduling algorithms outputs.

This paper is organized as follows: Section 2 presents end hosts mechanisms considered to enforce allocations. Scheduling strategies and transport protocol issues are considered in Section 3. Section 4 shows the experimental evaluation of these strategies and end hosts mechanisms combined.

2. END HOSTS MECHANISMS

2.1 Rate limitation mechanisms

In this work, for the sake of scalability, only software end-hosts rate limitation mechanisms are considered. We focus on GNU/Linux mechanisms as it is the most deployed operating system in grid environments.

Timescale used to define rates is of great importance. In real packet networks, flows are not fluid. Packets are sent entirely, one after one, at the wire bit-rate. This acts as an on/off sending process. Then, when considering fixed size packets, the only way to modify data rates over a large period of time is to vary inter-packets intervals.

To calculate these intervals, we have to consider the time source that can be used to enforce the limitation. In an end-host system four different time sources are available:

1. Userland timers;
2. TCP self clocking namely RTT of the transfer's path;
3. OS's kernel timers;
4. Packet-level clocking.

2.1.1 Userland timer based mechanisms

Application level timers can provide up to 8192 Hz timers through RTC[1] under GNU/Linux. RTC timers above 64 Hz are normally not accessible to users. Another solution would be to use POSIX functions like `nanosleep()`. But it has some limitations as the man-page suggests:

> *The current implementation of `nanosleep()` is based on the normal kernel timer mechanism, which has a resolution of 1/HZ s [...] Therefore, `nanosleep()` pauses always for at least the specified time, however it can take up to 10 ms longer than specified until the process becomes runnable again.*

This scheduling issue also applies for processes that use RTC based timers.

Trickle, proposed in [6], is a userland library which "moderate" `send()` and `recv()` calls using `nanosleep()` function in order to limit the rate. UDT [8], a transport protocol using rate-based congestion control, relies on userland `sleep()` function.

These mechanisms provide a coarse grained limitation.

2.1.2 TCP self-clocking based mechanisms

TCP self clocking depends on the round-trip time of the path and is subject to variations and ranges from about 0.01 ms within a LAN to several hundred of millisecond for worldwide connections.

As TCP is self-clocked by its acknowledgment mechanisms [11], the TCP congestion windows based limitation mechanisms are thus clocked on the RTT. TCP can not send more than its congestion window over one RTT period of time. We can thus limit the congestion window value to control the rate at the RTT timescale. This solution requires a precise knowledge of the RTT and only provides a controlled rate at RTT timescale which can slightly vary.

2.1.3 OS kernel timer based mechanisms

OS kernel timers depend on the kernel implementation and can be set up to 1000 Hz under current GNU/Linux kernel version.

GNU/Linux kernel uses an internal time source that rises an interrupt every `HZ`. This value can be set between 100 Hz and 1000 Hz. GNU/Linux provides several `qdisc`[2] to control the rate limitation. These mechanisms use HZ clocking. We note that HTB (Hierarchical Token Bucket) provides a classful limitation mechanisms. This mechanisms is actually implemented as a DRR[3] [5, 17] queuing discipline.

Some transport protocols do also rely on this time source. For example, DCCP [15] (Datagram Congestion Control Protocol) GNU/Linux implementation which use TFRC (TCP Friendly Rate Control) as congestion control mechanism relies on HZ timing.

2.1.4 Packet clocking based mechanisms

Packet-level clocking depends on the underlying link. Gigabit Ethernet carries a single 1500 bytes packet in 12 μs.

OS based timers cannot afford time sources being precise enough to accommodate packet length because OS kernel cannot be continuously handling interrupts. But the most precise solution to enforce bandwidth limitation is to increase packets' departure intervals. This solution is known as pacing. Although hardware solutions were proposed in [12], generic Gigabit Ethernet NICs do not provide pacing.

[1] Real Time Clock

[2] Queue disciplines

[3] Deficit Round Robin

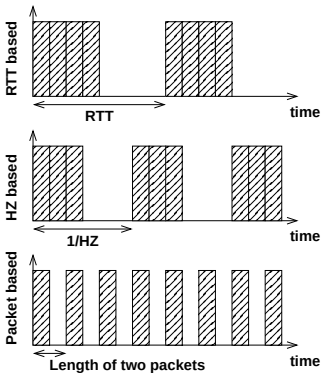


Figure 2: Different sending patterns.

Software implementations, like PSPacer [20], also exist for packet pacing. PSPacer introduces Ethernet PAUSE packets between packets carrying data. These PAUSE packets are discarded at the first switch encountered. They just introduce spacing between real packets, then limiting the effective rate. PSPacer is implemented as a GNU/Linux kernel `qdisc` but does not use any timers at the cost of an higher PCI bus bandwidth utilization. PSPacer actually uses a byte-clock by counting bytes sent and defines the departure date of a given packets in terms of "byte" time.

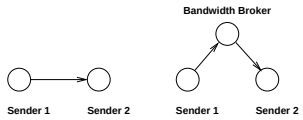
These time sources allow to create different sending patterns as shown on figure 2. The upper figure shows a schematic view of an RTT-based rate limitation, the lower ones packet-level rate limitation and in-between timer based rate limitation. These different rate limitation patterns generate different burst types, which impact global performance. Consequently, bursts of packets can lead to losses even if the rates of two flows are limited to insure the sum does not exceed the available bandwidth. The instantaneous aggregated rate, being larger than the output rate, may overflow the intermediate buffers. These losses generate retransmissions and TCP's congestion window moderation.

These different time sources can support different rate limitation mechanisms. In the absence of any rate limitation mechanisms, packets are sent in bursts due to TCP self-clocking.

2.2 Time control mechanisms

Synchronization mechanisms are often required by scheduling algorithms to enforce transfers start times. Two main solutions are available to control the time: time synchronization (real time) or explicit synchronization through signaling (virtual clocks). They both introduce some synchronization error. This error has to be much smaller (*e.g.* less than 10%) than the difference between minimum and maximum completion time of a transfer. For example, in a 1 Gbps real dedicated network, the difference between minimum and maximum completion time of the 15 GB file transfer with a single flow TCP-based protocol varies from 340 ms at 0.1 ms RTT to 9.3 s at 100 ms RTT.

The first solution relies on the time accuracy several machines can obtain from a time source. As attaching GPS device to each machine isn't practical, NTP-like synchronization of clock will have to be considered. In [16] published in 1994, NTP synchronization over a LAN is said to have an accuracy ranging from 500 μs to 2 ms representing 0.14% (resp. 0.59%) of the difference between maximum and minimum completion time of a 15 GB file transfer time with 0.1 ms RTT on a 1 Gbps link.



(a) Direct Signalization (b) Indirect Signalization

Figure 3: Signalization scenarii.

The second solution to provide synchronization is to use signalization. Both direct and indirect signalization can be used. With the direct signalization method, the first sender signals an event to the next sender (for example when the transfer is done as shown in figure 3(a)). Using indirect signalization, each sender exchange signals through a bandwidth broker (Figure 3(b)).

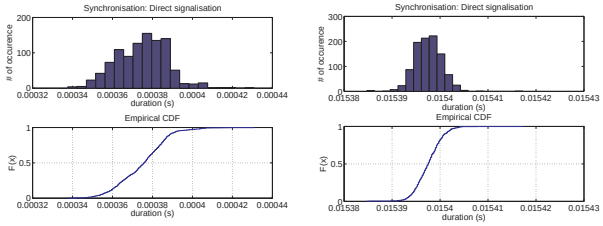
As a robust signalization mechanism is needed a three-way handshake should be used. For example TCP connection establishment ensures packet-loss resilience signalization. This basically means that the cost of the direct signalization will be in the order of $1.5 * RTT$ (due to SYN-SYN/ACK-ACK) where RTT is the path round-trip time between two senders plus the time to cross the local network stacks and the processing time.

In order to evaluate this cost, we measure the time required to perform two signalizations (one in each directions) between two machines and then divide this time by two. Figure 4 shows the distribution of this duration on a 0.1 ms RTT, 10 ms RTT and finally 100 ms RTT network. As expected, measures show that the cost grows linearly with a 3/2 rate from 0.374 ms at 0.1 ms RTT to 150.4 ms at 100 ms RTT. These durations represent respectively 0.11% and 1.6% of the difference of completion for the previously considered 15 GB transfer at 0.1 ms and 100 ms RTT. Thus cost of direct and indirect signalization are still a small part (less than 2%) of the variations of transfers' completion time if the processing time is assumed to be small.

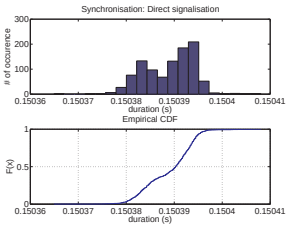
We can conclude from this section that synchronization by NTP and direct or indirect signalization do not introduce too much overhead and unpredictability to the completion time for the file size considered in this paper. Synchronization cost is independent from the size of transferred files. Thus when this size decreases, the relative cost of time control will increase and will not be negligible with respects to the transfer completion time variability and will have to be considered.

3. FLOW SCHEDULING ALGORITHMS

Before evaluating the accuracy and the interaction of transport protocols with these mechanisms, let us present the principle of three scheduling strategies evaluated here with a simple example. In this paper, we compare three flow scheduling strategies, named respectively ViFi for Virtual Finish Time First, IFS for Instantaneous Fair Sharing, MinRate for spaghetti scheduling. We consider two identical requests with the same release date, the same deadline. The



(a) 0.1 ms RTT (b) 10 ms RTT



(c) 100 ms RTT

Figure 4: Distribution of direct-signalization time (1000 repetitions).

Strategy	$t_{c,1}$	$t_{c,2}$
IFS	$\frac{2.v_1}{C}$	$\frac{2.v_1}{C} + \frac{v_2 - v_1}{C} = \frac{v_1 + v_2}{C}$
ViFi	$\frac{v_1}{C}$	$\frac{v_1 + v_2}{C}$
Spaghetti scheduling	$\frac{v_1 + v_2}{C}$	$\frac{v_1 + v_2}{C}$

Table 1: Theoretical completion time of the two transfers.

first request has a volume being twice the volume of the second request.

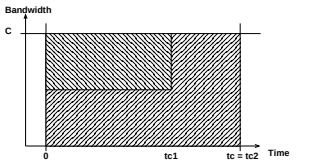
3.1 Simple illustrative example

Let us consider a classical dumbbell topology with two sources nodes (s_1 and s_2) and two sink nodes (S_1 and S_2). The topology is presented in figure 7. The shared link has a capacity of C (1 Gbps Ethernet). Each source node has a file of size v_i to transfer to the corresponding sink node (S_1 for s_1 and S_2 for s_2). We assume that both transfers have the same release date t_r and we define the completion time of the transfers $t_{c,1}$ and $t_{c,2}$. Table 1 summarizes the theoretical completion time of each transfer under the different scheduling strategies.

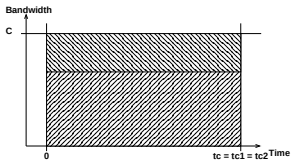
If we consider same size files, IFS and spaghetti scheduling are equivalent. Thus, the two files transferred will have a size of $v_1 = 15000$ MB and $v_2 = 30000$ MB. For the sake of simplicity, we will consider that the release dates are 0 ($t_r = 0$).

Figures 5(a), 5(b) and 5(c) show the different scheduling strategies applied to the two transfers considered.

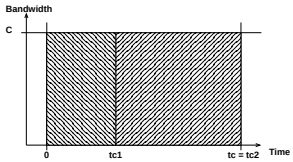
The completion time of each transfer might differ from strategy to strategy but the global completion time of the two transfers – *i.e.* the makespan – is theoretically the same. We will focus on this last objective: minimize the makespan t_c, with $t_c = max\{t_{c,1}, t_{c,2}\} = (v_1 + v_2)/C$. The link capacity C used here is the effective rate of carried data. As the transport protocol provides congestion control mechanisms and reliable transfers, it introduces some overheads. Then the considered link capacity is smaller than the raw capacity of the link. This point will be detailed in next section. The three strategies allocate the full capacity (link utilization ratio of 1 is optimal) provided that a single flow can fill the shared link and we assume an accept rate of 1 (all request accepted). This means that the specified deadline of each transfer is greater than its completion time.



(a) IFS scheduling.



(b) Spaghetti scheduling.



(c) ViFi scheduling.

Figure 5: Three simple different scheduling strategies for two transfers (transfer 1 with volume v_1 and transfer 2 with volume $v_2 = 2.v_1$), having the same time window constraint.

We consider that IFS can be well approximated by TCP's distributed congestion control. ViFi does not require any bandwidth limitation mechanism as there is only one flow at a time. But *spaghetti scheduling* requires bandwidth limitation enforcement mechanisms.

3.2 Transport protocol considerations

Our goal is to determine which configuration (scheduling strategy combined with end hosts mechanisms) is minimizing t_c which is the time the bandwidth broker will have to reserve resources to complete the two transfers.

All proposed algorithms assume that an ideal transport protocol will be used to transfer the files. However, the achievable bandwidth is not known *a priori* because it depends on the efficiency of the real transport protocol and the rate limitations mechanisms. Current transport protocols, used in grid environments, are window-based protocol, like BIC TCP, which are quite efficient and fair when two flows have the same reasonable round-trip time (< 100 ms) [10]. But such protocols still introduce some overhead. This overhead is partially due to the packets' header which size is a linear function of the data carried. The overhead is also due to the retransmissions induced by losses. Under a congested situation (which is the case here as we can have two potentially 1 Gbps sources sending data in 1 Gbps link at the same time with IFS scheduling), the overhead introduced by retransmissions is more difficult to estimate. The arbitration algorithms within the network equipment (*e.g.* Ethernet switch) that mixes the two flows impact the losses distribution among flows [18].

This loss-dependent overhead might suggest that spaghetti

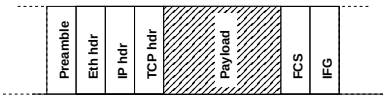


Figure 6: Frame and payload.

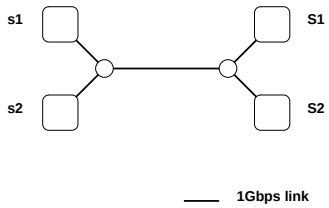


Figure 7: Testbed.

scheduling would provide better performance. It avoids congestions by limiting the sending rate of each flow. ViFi scheduling insures that there is no congestion because only one flow is scheduled at a time. However these two scheduling strategies imply rate limitation or synchronization and signalization mechanisms which isn't costless: Spaghetti scheduling needs rate limitation. ViFi needs to start the second transfer just after the first one finished.

3.3 Goodput *vs* throughput

In this section no attention is paid to the timescale used to define rates. Let r_{EthB} be the Ethernet bit rate. $r_{\mathrm{EthB}} = 10^9\,bps$ for Gigabit Ethernet. We define r_{Eth} as the Ethernet frame rate assuming 1514 bytes frames (without frame checksum (FCS), frame preamble and inter frame gap (IFG)), see figure 6. $r_{\mathrm{Eth}} = (1514/(1514+4+12+8)) * r_{\mathrm{EthB}} = 984.40\,\mathrm{Mbps}$. We define r_{IP} as the rate of IP packets (Ethernet payload). Under the assumption of a 1500 bytes MTU[4]: $r_{\mathrm{IP}} = (1500/1514) * r_{\mathrm{Eth}} = 975.30\,\mathrm{Mbps}$.

In order to compare our measurements we define an ideal transport protocol over an 1 Gbps Ethernet link. We assume this protocol achieve perfectly max-min fairness among contending flows, there is no slow-start and no losses nor retransmissions. We define the data rate r_{TP} of the ideal transport protocol. Under assumptions presented above, $r_{\mathrm{TP}} = ((1500-40)/1500) * r_{\mathrm{IP}} = 949.29\,\mathrm{Mbps}$.

This is especially important to differentiate goodput (r_{TP}) and throughput (r_{EthB}) as flows share links throughput while completion time of transfers depends on goodput. Thus the capacity C used in section 3.2 is actually r_{TP}. We can observe that this rate is a linear function of r_{Eth} as each transport layer payload as its own set of headers, preamble, FCS and IFG. But real transport protocols are not ideal. More specifically, the effective data rates are not a linear function of the portion of the raw link obtained as real transport protocols use congestion window based congestion control driven by loss event or latency changes. These events are not independent between contending flows. In order to obtain deterministic completion time, we have to isolate flows.

4. EXPERIMENTAL EVALUATION

4.1 Objectives

To evaluate some of the rate limitation mechanisms combined with scheduling strategies introduced above, real experiments have been conducted.

The main goal of these experiments is to compare the different scheduling strategies in term of completion time predictability and mean performance. These two points are important because in a scheduling perspective, predictability of completion time is central and secondly scheduling was introduced to improve mean performance. More specifically any scheduling algorithms has to be more efficient than raw TCP sharing.

Next section will present the testbed and scenarii considered, while two following section present mean and scheduling related performance results.

4.2 Testbed, scenarii & practical considerations

As stated before, two transfers are used. These transfers are realized by using *iperf* on Sun Fire V20z workstations of Grid5000 [4] running GNU/Linux 2.6.20.1 kernels. The switch used to implement the topology is a ExtremeNetworks BlackDiamond 8810. Latencies are emulated using a GtrcNet-1 box [13]. Four different experiments were conduced at three latencies (RTT): 0.1 ms, 10 ms and 100 ms. Completion times were measured with a 10 ms precision. Each experiment consists of hundred repetitions of the two transfers.

The first experiment implements the IFS strategy. The two transfers (15GB and 30GB) have been started at the same time without bandwidth limitation. In this experiment the time measured is the maximum of the two individual completion time.

Second experiment implements the ViFi strategy. The 15GB transfer is started first and the second transfer is started as soon as the first completes. The duration measured in this experiment is the sum of the duration of the two transfers.

Third and fourth experiments implement spaghetti scheduling strategy. Third experiment uses PSPacer as limitation mechanism while fourth uses HTB. Here the time measured is also the maximum of the two individual completion time. These two scenarii use `qdisc` based rate limitation mechanisms. First use packet-level rate limitation using PSPacer while second use OS's timer based approach with HTB. As they both use `qdisc` implementation, the rates specified for the two flows are $2/3 * r_{\mathrm{Eth}} = 656\,\mathrm{Mbps}$ and $1/3 * r_{\mathrm{Eth}} = 328\,\mathrm{Mbps}$ for 30 GB and 15 GB transfers as `qdisc` acts on ready-to-be-sent packets (Ethernet packets).

Retransmissions done by the transport protocol during 10 repetitions of these experiments are also measured using Web100 kernel patch.

4.3 Mean performance

Figure 8 shows the distribution and cumulative distribution function (CDF) of the completion times of the two transfers using different strategies at three latencies (RTT). CDF of completion time is important because it shows the dispersion which is a key point for defining the time granularity of the scheduling.

IFS strategy implemented through BIC TCP congestion control mechanism is efficient under low latencies but as the latency grows, the mean completion time grows quickly.

[4]Message Transfer Unit

	0.1 ms RTT	10 ms RTT	100 ms RTT
IFS	3.4%	16.4%	64.3%
ViFi	3.7%	9.2%	9.7%
SS (PSP)	4.0%	4.0%	4.7%
SS (HTB)	5.3%	17.7%	59.6%

Table 2: Relative deficiency of mean real completion time against ideal one.

This is due to the cost of losses which decrease the congestion window, *i.e.* the sending rate. Loss have more and more impact as the latency increases as recovering takes longer. At 0.1 ms RTT, the congestion window is about 8 packets which is smaller than the buffer size of the switch. At 10 ms RTT, the congestion window is about 800 packets and at 100 ms is close to 8000 packets. In these situations, the bursts sent by the two sources are very likely to cause congestion. In addition, as losses do not occur exactly at the same time from experiment to experiment, the dispersion tends to be very important especially under high latencies (figure 10(b)) were losses are more costly.

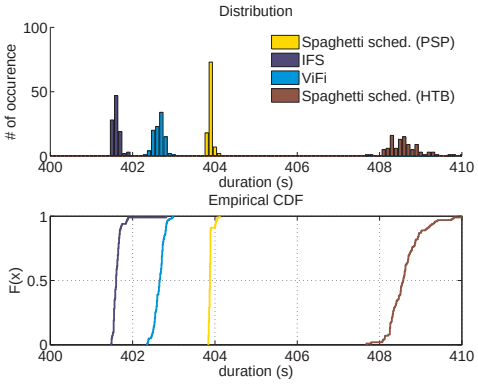
ViFi is a bit less efficient than IFS at 0.1 ms (figure 8(a)) because there is only one flow at a time. Increasing the number of flows sharing a link (*i.e.* the multiplexing level) is known to increase the average utilization rate of this link. Performance decreases a little as latency grows but the distribution remains quite compact.

Spaghetti scheduling's performance depends on the limitation mechanisms used. HTB introduces some burstiness as it use OS timers. Thus the number of losses may be important as observed on figure 9. While IFS strategy allows flows to recover by momentarily stealing bandwidth to each others, HTB does not which leads to poor performance. Using PSPacer, for rate limitation leads to a nearly constant completion time among repetitions. As we can observe on figure 10(a), the mean completion time of spaghetti scheduling using PSPacer is stable with respect to the latency. This combination (spaghetti and packet pacing) offers the near ideal implementation of the flow scheduling approach.

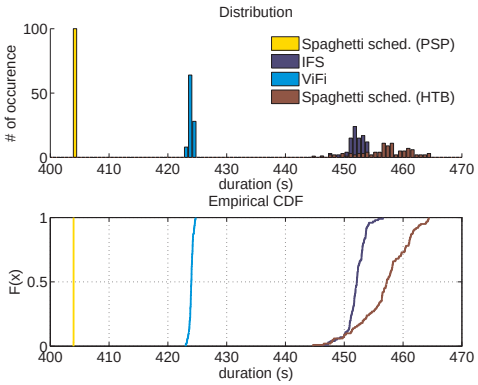
To summarize these results in term of mean, we can compare real completion time to ideal completion time obtained with r_{TP} which is $t_c = (v_1 + v_2)/r_{\mathrm{TP}} = 388.33\,s$ in Table 2. We observe a mean deficiency of spaghetti scheduling using PSPacer below 5% while IFS reaches more than 64.3% under 100 ms RTT. These results represent mean performance. Next section will develop worst case results in terms of minimizing the maximum completion time as stated in section 3.2.

4.4 Scheduling perspective

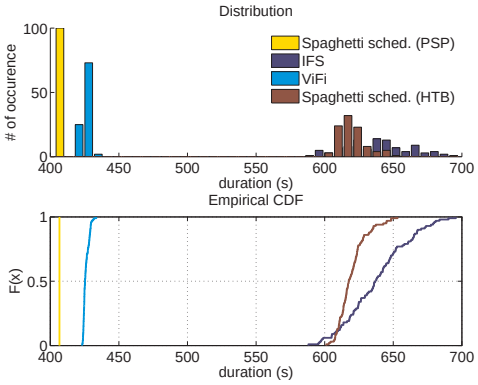
From a scheduling perspective, the main concern is the variability of completion times. This is because when the flow scheduler decides a time slot for a transfer, this one must have, with an high probability, enough time to complete. On the other hand, allocating a larger time window will be wasteful. In this perspective, mean completion time is of little help. As observed on figure 10(b), variability greatly depends on the strategy and end hosts mechanisms used. Variability is actually very low when using PSPacer and can be important (std. dev. greater than 20 s for a mean completion time of about 650 s) for IFS.



(a) 0.1 ms RTT



(b) 10 ms RTT



(c) 100 ms RTT

Figure 8: Completion time distribution and CDF for IFS, ViFi and Spaghetti scheduling using BIC TCP under different RTT (100 repetitions).

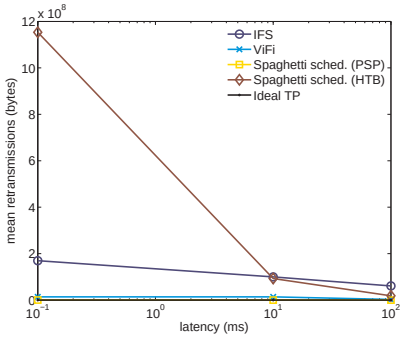
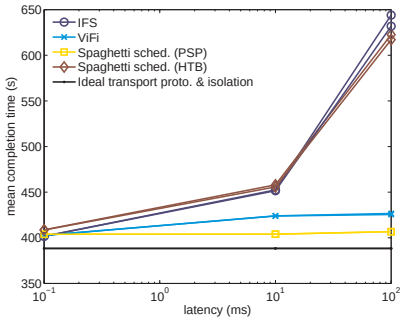
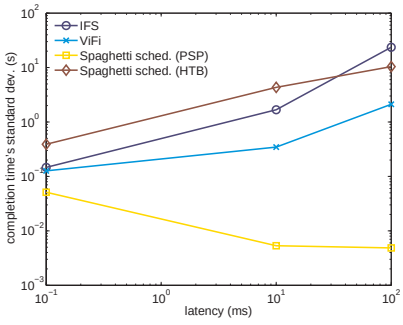


Figure 9: Mean retransmitted bytes as a function of the RTT (10 repetitions).



(a) 99% confidence interval for mean completion time as a function of the RTT.



(b) Standard deviation of completion time as a function of the RTT.

Figure 10: Mean completion times and standard deviations.

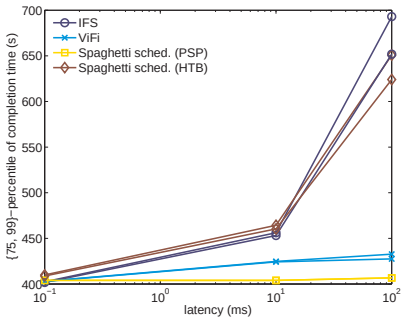


Figure 11: 75 and 99-percentiles of completion times.

	0.1 ms RTT	10 ms RTT	100 ms RTT
IFS	3.6%	17.5%	78.5%
ViFi	3.8%	9.4%	11.4%
SS (PSP)	4.0%	4.0%	4.7%
SS (HTB)	5.6%	19.6%	67.6%

Table 3: Relative deficiency of 99-quantile of real completion time against ideal one.

Figure 11 shows the 75 and 99-percentiles of the completion time for the different strategies as a function of latency. This value is the duration of the slot the scheduling system would have to allocate in order to have a probability of completion of 75% (99%) in a given situation. We can observe that these two values are quite close except under 100 ms RTT with IFS and HTB. Second observation, these last two strategies, and to a lesser extend ViFi, would require a scheduling system aware of the latencies of the links. Indeed, the performance slightly decreases as latency increases.

Performance obtained using real mechanisms and protocols do not behave as an ideal transport protocol as seen on figure 10(a) and 10(b). There are 3 differences. First, the mean completion time of real transfers is obviously larger than the one of ideal transfer (of 3.4% is the best case and 64.3% in the worst one). Second it increases as the latency increases for HTB and IFS as packet losses are introduced. Third, the variance is not null and can be relatively high. To conclude this section, spaghetti scheduling using PSPacer appears to be the strategy which provides the best maximum completion time t_c under real experiments. Its 99-quantile completion time deficiency does not vary much with latency and is the lowest (except for 0.1 ms RTT) as can be seen in table 3.

5. CONCLUSION

In this paper we have quantified and compared, end-host based mechanisms combined with transport protocols to instantiate different scheduling strategies. A simple scenario has been deeply explored in a range of latency conditions. We have shown that, in high speed network, a single-rate scheduling strategy implemented by TCP-variant protocol like BIC with packet pacing mechanism offers predictable performance and is insensitive to latency (deficiency of mean completion time ranging from 4% at 0.1 ms RTT to 4.7% at

100 ms RTT). This paper also highlights the limits of other strategies and rate limitation mechanisms like token bucket which may present unpredictability and other drawbacks. Future work will concentrate on larger experiments on the Grid5000 testbed and will examine the scalability of the flow scheduling approach in real grid context, multirate allocations schemes and transfers preemption issues.

6. ACKNOWLEDGMENTS

This work has been funded by the French ministry of Education and Research via the HIPCAL ANR grant and by the EU IST FP6 EC-GIN Project and GridNet FJ grant. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see https://www.grid5000.fr).

7. REFERENCES

[1] C. B. Bin and P. V.-B. Primet. Scheduling deadline-constrained bulk data transfers to minimize network congestion. In *CCGrid*, Rio de Janeiro, Brazil, May 2007.

[2] R. Braden, L. Zhang, S. Berson, S. Herzog, and S.Jamin. Resource ReSerVation Protocol (RSVP), ietf rfc 2205, September 1997.

[3] L. Burchard, H. Heiss, and C. D. Rose. Performance issues of bandwidth reservations for grid computing. In *Symposium on Computer Architecture and High Performance Computing (CAHPC)*, pages 82–90, Sao Paulo, Brazil, November 2003.

[4] 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 *the 6th IEEE/ACM International Workshop on Grid Computing, Grid'2005*, Seattle, WA, November 2005.

[5] M. Devera. Hierachical token bucket theory. Website, May 2002. `http://luxik.cdi.cz/~devik/qos/htb/manual/theory.htm`.

[6] M. Eriksen. Trickle: A userland bandwidth shaper for unix-like systems. In *USENIX Annual Technical Conference, FREENIX Track*, 2005.

[7] S. Gorinsky and N. S. V. Rao. Dedicated Channels as an Optimal Network Support for Effective Transfer of Massive Data. In *Proceedings of IEEE High-Speed Networking Workshop (HSN) 2006*, 2006.

[8] Y. Gu and R. L. Grossman. Udt: Udp-based data transfer for high-speed wide area networks. *Comput. Networks*, 51(7):1777–1799, 2007.

[9] R. Guérin and A. Orda. Networks with advance reservations: The routing perspective. In *IEEE INFOCOM*, Tel-Aviv, Israel, March 2000.

[10] R. Guillier, L. Hablot, Y. Kodama, T. Kudoh, F. Okazaki, R. Takano, P. Primet, and S. Soudan. A study of large flow interactions in high-speed shared networks with grid5000 and gtrcnet-10 instruments. In *PFLDnet 2007*, February 2007.

[11] H. Jiang and C. Dovrolis. The origin of TCP traffic burstiness in short time scales. Technical report, Georgia Tech., 2004.

[12] K. Kobayashi. Transmission timer approach for rate based pacing TCP with hardware support. In *PFLDnet 2006*, Nara, Japan, 2006.

[13] Y. Kodama, T. Kudoh, T. Takano, H. Sato, O. Tatebe, and S. Sekiguchi. GNET-1: Gigabit ethernet network testbed. In *Proceedings of the IEEE International Conference Cluster 2004*, San Diego, California, USA, Sept. 20-23 2003.

[14] L. Marchal, P. Primet, Y. Robert, and J. Zeng. Optimal bandwidth sharing in grid environment. In *IEEE High Performance Distributed Computing (HPDC)*, Paris, France, June 2006.

[15] A. C. Melo. Dccp on linux. In *Ottawa Linux Symposium*, pages 305–311, 2005.

[16] D. L. Mills. Precision synchronization of computer network clocks. *SIGCOMM Comput. Commun. Rev.*, 24(2):28–43, 1994.

[17] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. In *SIGCOMM '95: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 231–242, New York, NY, USA, 1995. ACM Press.

[18] S. Soudan, R. Guillier, L. Hablot, Y. Kodama, T. Kudoh, F. Okazaki, R. Takano, and P. Primet. Investigation of ethernet switches behavior in presence of contending flows at very high-speed. In *PFLDnet 2007*, February 2007.

[19] J. Stankovic, M. Spuri, M. D. Natale, and G. Buttazzo. Implications of classical scheduling results for real-time systems. *IEEE Computer*, 28(6):16–25, 1995.

[20] R. Takano, T. Kudoh, Y. Kodama, M. Matsuda, H. Tezuka, and Y. Ishikawa. Design and evaluation of precise software pacing mechanisms for fast long-distance networks. In *PFLDnet 2005*, Lyon, France, 2005.

[21] H. Zhang, K. Keahey, and W. Allcock. Providing data transfer with QoS as agreement-based service. In *IEEE International Conference on Services Computing (SCC)*, Shanghai, China, August 2004.