The Journal of Supercomputing manuscript No.
(will be inserted by the editor)

Performance Analysis of Available Bandwidth Estimation Tools for Grid Networks

Daniel M. Batista · Luciano J. Chaves · Nelson L. S. da Fonseca* · Artur Ziviani

the date of receipt and acceptance should be inserted later

Abstract Modern large-scale grid computing systems for processing advanced science and engineering applications rely on geographically distributed clusters. In such highly distributed environments, estimating the available bandwidth between clusters is a key issue for efficient task scheduling. We analyze the performance of two well known available bandwidth estimation tools, `pathload` and `abget`, with the aim of using them in grid environments. Our experiments consider the accuracy of the estimation, the convergence time, their level of intrusion in the grid links, and the ability to handle multiple simultaneous estimations. Overall, `pathload` represents a good solution to estimate available bandwidth in grid environments.

Keywords grids · networks · bandwidth estimation · performance evaluation

Mathematics Subject Classification (2000) 68M14 · 90B18 · 62N02 · 68M20

1 Introduction

Modern large-scale grid computing systems for processing advanced science and engineering applications rely on geographically distributed clusters [1], demanding coordinated resource sharing for problem solving in heterogeneous dynamic environments. Grid computing differs from conventional parallel computing since the latter involves confined systems and uses local networks for

* Corresponding Author

D. M. Batista · L. J. Chaves · N. L. S. da Fonseca
Institute of Computing, University of Campinas (UNICAMP), Campinas, Brazil.
Tel.: +55(19)35215878, Fax.: +55(19)35215847,
E-mail: batista@ic.unicamp.br, luciano@lrc.ic.unicamp.br,nfonseca@ic.unicamp.br

A. Ziviani
National Laboratory for Scientific Computing (LNCC), Petrópolis, Brazil,
E-mail: ziviani@lncc.br

data transfer, whereas the former is composed of geographically distributed clusters interconnected via a wide area network with communication links belonging to different administrative domains and shared by a large number of applications.

In parallel/distributed computing, applications are divided in logical executable pieces called tasks. Grid scheduling involves the assignment of tasks to hosts and the start of their execution is influenced by host characteristics such as CPU and memory capacity as well as by network characteristics such as bandwidth.

Since grid scheduling relies on the available bandwidth for data transfer among distributed tasks, accurately estimating this value is a key issue for efficient task scheduling. Moreover, the available bandwidth is highly dynamic and thus needs to be frequently measured to support efficient grid schedules. Given a certain time interval, measurements of the available bandwidth have inherent uncertainties due to both its dynamic nature and the inaccuracy of the adopted estimation tools [2]. These uncertainties justify the estimation of the available bandwidth to be represented by intervals rather than by simple averages. Considering the available bandwidth as a mean value and ignoring existing uncertainties can increase the makespan of grid applications by roughly 20% [3]. Such uncertainties also influence the efficient tuning of data transfer control. For example, recent results indicate that the automatic adjustments of `GridFTP` parameters are directly related to the available bandwidth and the bandwidth-delay product between processing nodes [4].

Moreover, Silvester [5] showed that the execution of highly demanding grid applications can induce a growth of five to eight times in the utilization of network links. This increase can be highly significant for emerging applications, such as the recent CERN's LHC Computing Grid (LCG) which expects to distribute and process around 15 PB per year [6]. Furthermore, such an increase brings significant fluctuations on the available bandwidth.

Several grid systems adopt self-adjustment procedures for the allocation of resources in order to cope with fluctuation of resource availability [7] [8] [9] [10]. The adoption of these procedures are motivated by the fact that resources of grids are not usually owned by their users, whom do not have exclusive right of use of grid resources. In this approach, grid resources are monitored and if a different allocation of distributed resources leads to lower makespan, tasks are migrated to the new allocation scheme. Having accurate estimations of the available bandwidth is, thus, of paramount importance to evaluate the potential re-scheduling of the tasks of an application.

This paper investigates the adequacy of existing available bandwidth estimation tools for their adoption in the scheduling of grid tasks. To the best of our knowledge, this is the first work to analyze available bandwidth tools as a main point for efficient grid scheduling in the context of network-based high performance computing. Criteria for comparing these tools are: accuracy of estimation, convergence time, level of intrusion in grid links, and the ability to handle simultaneously multiple estimations. Convergence time impacts the makespan of applications and it should be relatively short compared to the

expected makespan of the application since the time to produce a task schedule should be short [7]. The level of intrusion in grid links impacts resource availability given that communication links are shared resources. Moreover, analyzing the ability to perform simultaneous estimations is quite relevant since in operational grids users and applications have asynchronous behaviour.

We consider the `pathload` [2] and the `abget` [11] tools for estimating the available bandwidth since among the various estimation tools (e.g. see [11,12]), only these tools provide intervals associated to their estimations. Other studies that compare tools for estimating available bandwidth including `pathload` and `abget` can be seen in [13, 14]. Nevertheless, previous works focus on the accuracy of estimations and disregard all the other metrics analysed in this paper which are of paramount importance for the efficient execution of applications in large-scale grid environments. Overall, results indicate that `pathload` is the preferred tool to be adopted in grid environments.

The remainder of the paper is organized as follows. Section 2 briefly overviews bandwidth estimation procedures and introduces the `pathload` and the `abget` tools. Section 3 describes the experiments performed as well as discusses the results obtained. Section 4 draws some conclusions.

2 Tools for Available Bandwidth Estimation in Grid Networks

Estimating the available bandwidth includes measuring different metrics [15] such as link nominal capacity, bottleneck capacity along a path, end-to-end available bandwidth along a path, and bulk transfer capacity (BTC) between pairs of hosts. Figure 1 illustrates these metrics. Host 1 and Host 2 are interconnected by a path composed of three links, depicted as rectangles, with the gray part representing the used capacity and the white one the available bandwidth. The nominal capacity of each link is $C1$, $C2$, and $C3$ and the bottleneck capacity of the path is $C1$, thus, the end-to-end available bandwidth is $A3$. BTC is the maximum throughput obtained by a TCP connection along the network path. We cannot represent the BTC metric in Figure 1 because it depends on the type of concurrent traffic found in the path links at the moment measures are taken. In this paper, we focus on tools that estimate end-to-end available bandwidth.

In [12], a list of available bandwidth estimators is provided. Among them, `iperf` [16] is very popular among network administrators. `iperf` estimates the available bandwidth along a path by saturating the path with data sent between the two end points and measuring the amount of data sent. However, `iperf` is too intrusive and estimations are given as mean values. The intrusion issue can be ameliorate by employing TCP to send data between the end points and to start collecting measures after TCP slow start phase [17]. Another popular estimator is the `DIChirp` [18] which has access to the network card of a host to exchange packets between the host and another peer host. If the receiver detects a delay increase, the sending rate previous to the delay increase

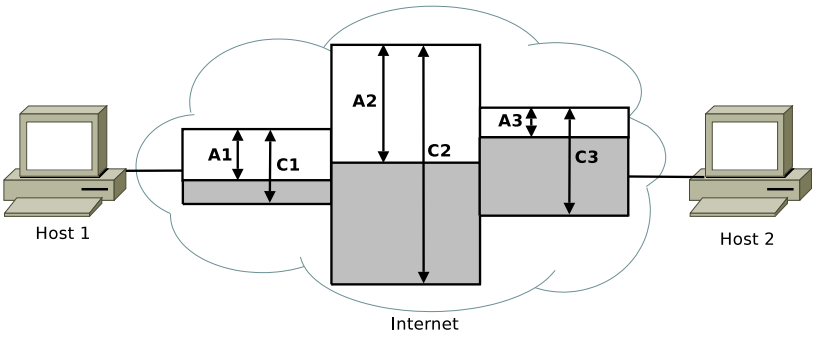


Fig. 1: Illustration of bandwidth metrics (based in a figure from [15]).

is considered to be the estimation of the available bandwidth. As the `iperf`, `DIChirp` provides estimations as mean values.

The `pathload` and `abget` tools perform their estimations using the Self-Loading Periodic Streams (SLoPS) technique. In this technique, several sequences of packets are transmitted between two end nodes using different rates, for measuring the One-Way Delay (OWD)[1] of every packet sent at different rates. If an increase of the OWD value is detected, it means that the corresponding transmission rate is larger than the available bandwidth; otherwise, the transmission rate is smaller than the available bandwidth. Information on variations of the OWD value is exchanged between end nodes to estimate an interval width that represents the end-to-end available bandwidth along the path between the two end hosts. `pathload` and `abget` estimate the end-to-end available bandwidth as follows:

2.1 `pathload`

To estimate the available bandwidth from host A to host B using `pathload`, there is a need to execute a "sender" process at the host A and a "receiver" process at the host B. Information concerning the variations of the OWD value is exchanged between the end hosts via a TCP control connection. The "sender" initiates the estimation process sending one sequence of UDP packets at an initial rate R_{start}. As the packets arrive at the "receiver", the OWD value is computed. In case no increase of OWD value is observed, the "receiver" notifies the "sender" to increase the packet sending rate. Several algorithms for adjustment of the variations of the OWD value that are not caused by sending rate values larger than the available bandwidth value are implemented to avoid incorrect estimations. At the end of the estimation, `pathload` provides

[1] To measure OWD, the end nodes have to be synchronized. Possible solutions are the use of NTP (Network Time Protocol) servers, the adoption of GPS cards at both ends, or a software clock to enhance measurement accuracy without using GPS cards, such as the one proposed in [19].

as output an interval $[R^{min}, R^{max}]$ that corresponds to the range of available bandwidth along the path between the host A and the host B.

2.2 `abget`

To estimate the available bandwidth from host A to host B, an `abget` client at the host A should be directed to a TCP server (e.g. a web server) running either at the host B or at a host located in the same network where the host B resides. `abget` simulates the operation of TCP so that it controls the rate host A delivers packets to the host B. The `abget` client ignores the standard operating system implementation of TCP and manipulates the ACKs sent by the host B. TCP segments with length equal to 1 MSS are sent to the host B. Upon receiving each segment, the host B sends one ACK to host A. If the host A sends segments in intervals of duration T (Rate $R = MSS/T$), the `abget` client induces the host B to send ACKs with a constant rate. Upon the arrival of ACKs at the host A, the OWD value is measured and the rate at which segments are generated is adjusted in order to find the interval $[R^{min}, R^{max}]$ which corresponds to the range of available bandwidth along the path between the two hosts. `abget` can employ two different types of search for the available bandwidth value: a binary search and a linear search. The binary search doubles the sending rate if it is lower than the available bandwidth and reduce it to half of it in case it is greater than the available bandwidth. In the linear search, the sending rate is increased/decreased by one unit of the sending rate.

2.3 Differences between `pathload` and `abget`

The `pathload` and `abget` differ in the way they implement the SLoPS technique. On the one hand, `pathload` requires processes to be executed at both end hosts in order to allow the exchange of information about the OWD value measured from source to destination. On the other hand, `abget` executes at one end host but it requires a web server (HTTP) to be active either at the other end host or in the local network this host is located. The control of the transmission rate of the packets and the monitoring of the OWD in `abget` are performed by the source itself using information related to the congestion control algorithms of TCP.

If, on the one hand, the main advantage of `abget` is to provide estimations by running a process only at one end host; on the other hand, it requires administrative privilege to operate TCP differently than the standard implementation in the local operating system. Another drawback is the need to inform manually several parameters to allow a relatively fast convergence to the interval corresponding to the available bandwidth estimation. `pathload` does not need these parameter values due to the employment of a TCP connection which allows the fine tuning of SLoPS in running time. However, besides involving both end hosts, `pathload` uses UDP to estimate the available bandwidth, which can be ineffective since UDP packets are commonly blocked in

firewalls due to security reasons. In contrast, `abget` does not face this problem because most domains already allow the delivery of packets to HTTP servers.

3 Performance Analysis

Two different scenarios were employed to evaluate the performance of the `pathload` and the `abget` tools. The first scenario involves links of small nominal capacities (10Mbps) and the second scenario links with large nominal capacities (1Gbps). It is important to analyze these tools in both scenarios due to the heterogeneity of the link capacities interconnecting clusters in a typical grid.

3.1 First scenario: small nominal link capacities

The first scenario, illustrated in Figure 2, was emulated using the `NCTUns` emulator [20] which allows the integration of simulated network topologies with real hosts running actual applications without requiring any modification of these applications. Estimations of the available bandwidth in this first scenario are performed from the hosts cronos and eolo to the host mnemosyne. All these hosts are real hosts located in the same local Gigabit Ethernet network. `NCTUns` was executed in the host named urano, located in the same local network. In order to evaluate the tools under different network conditions, two virtual hosts were used in `NCTUns` to generate interfering traffic between the real hosts. Table 1 summarizes the configuration of the hosts involved in the experiments.

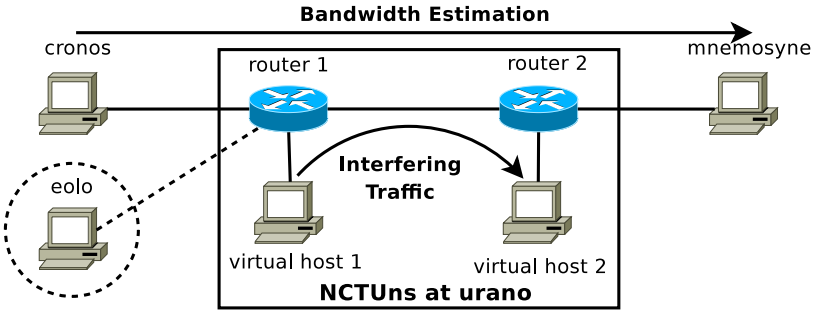


Fig. 2: Experimental setup with NCTUns.

For each tool, three metrics were evaluated in the experiments: accuracy, execution time, and level of intrusion. The first scenario involved CBR UDP traffic sent at 2, 4, 6, 8, and 10Mbps rates, TCP traffic, and no interfering traffic. The nominal capacity of the links were fixed in 10Mbps during measurements. Two sets of measures where collected. Measures in the first set

Table 1: Characteristics of the hosts used in the experiments.

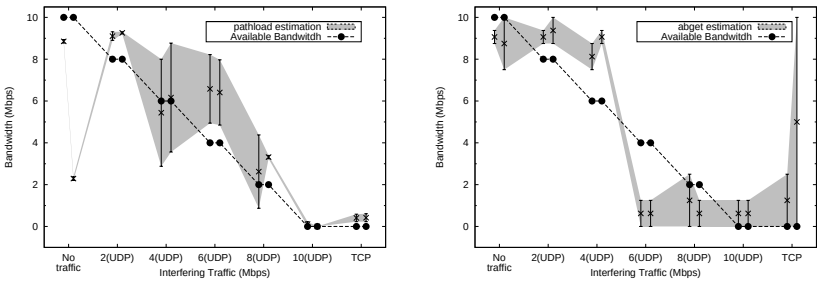
Host	**Processor/Memory**	**Operating System** (Linux)
eolo	Intel Core 2 Quad 2.66GHz / 4GB	Debian kernel 2.6.23.1
cronos	Intel Core 2 Quad 2.40GHz / 4GB	Debian kernel 2.6.23.1
mnemosyne	Dual Xeon 2GHz / 4GB	Debian kernel 2.6.23.1
urano	Intel Core 2 Duo 2.13GHz / 4GB	Fedora kernel 2.6.25.9

were collected between the hosts cronos and mnemosyne. Measures in the second set were collected simultaneously between the host cronos and the host mnemosyne, and between the host eolo and the host mnemosyne. Only the most relevant results are presented and discussed in the paper. In all experiments, binary search was used since the linear one can take three time longer to produce the desired value.

A relevant setting in the experiments was the disabling of Interrupt Coalescence (IC) in the network cards. IC is a feature of some network cards to decrease the amount of interruptions generated by the operating system when packets are either received or sent. When IC is enabled, interrupts are generated only after the existence of a certain amount of packets or at periodic intervals. As shown in [21], the precision of estimations of the available bandwidth are compromised when IC is enabled due to the extra delay the adoption of the IC feature incurs. In this way, we disabled IC in all experiments.

Figure 3 and Figure 4 present the results obtained for the first set of measures. Figures 3(a) and 3(b) present results provided by `pathload` and `abget`, respectively, as a function of the interfering traffic. In these figures, the curve named "Available Bandwidth" shows the actual available bandwidth between the hosts. The gray areas named "pathload estimation" and "abget estimation" show the intervals that have been provided by these tools. For each configuration of interfering traffic, the estimation tools were executed twice. Results evince that `pathload` estimations were closer to the actual values when there was interfering traffic. Besides that, `abget` estimations did not follow the bandwidth availability along the path between the cronos and the mnemosyne hosts. With interfering traffic less or equal to 4Mbps, `abget` estimated that the path was almost 100% available, whereas with interfering traffic greater or equal to 6Mbps estimations indicated that the link was almost unavailable. The estimation intervals from `abget` were on average larger under TCP interfering traffic than when they were under other interfering traffic.

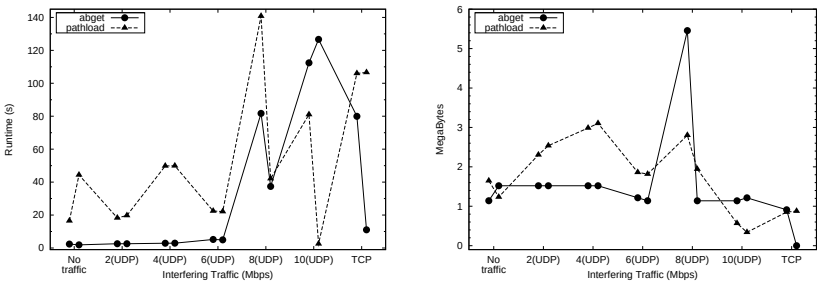
The execution time of the estimation tools is shown in Figure 4(a). Since `abget` does not demand an initial value of the transmission rate close to the nominal capacity, it requires the transfer of the same quantity of data at the beginning of the estimation regardless the availability of the links. As a consequence, as the links become more utilized, the execution time tends to increase, as can be observed in the "abget" curve. Nevertheless, when the



(a) `pathload` (b) `abget`

Fig. 3: Estimations provided by `pathload` and `abget` (first scenario).

interfering traffic is small ($\leq$ 8Mbps), `abget` converges to results faster than does `pathload`. The low execution time of `abget` in one of the executions carried out with TCP interfering traffic was due to the fact that the tool was not able to connect to the web server at the host mnemosyne under heavy load.



(a) Execution time (b) Level of intrusion

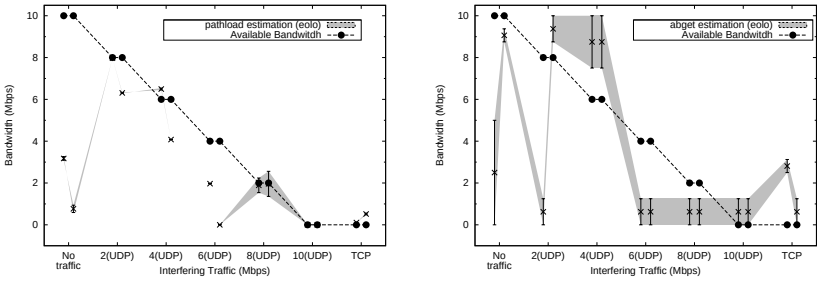
Fig. 4: Performance of the estimation tools in the first scenario.

The levels of intrusion expressed as the volume of injected bytes by each tool during its execution are shown in Figure 4(b). We observe that `abget` is more stable than `pathload`. The `pathload` tool tends to reduce its generated traffic as the links get less available. This happens because the transmission rate of `pathload`, at each iteration of the algorithm, is not guided by the binary search implemented in `abget`. In this way, after some iterations, `pathload` can reduce its transmission rate and inject less traffic than `abget`.

Some of the results obtained in the second set of measures are shown in Figures 5 and 6. Results for the levels of intrusion of the tools were quite similar to those obtained with just one instance of the estimation tools in execution, so they have been omitted here for the sake of clarity. We make a remark that

in order to allow the simultaneous execution of `pathload`, its source code was modified because the ports used by the program are statically defined in the original code.

Figures 5(a) and 5(b) show the results when `pathload` and `abget` estimated the available bandwidth between the host eolo and the host mnemosyne. Results considering the hosts cronos and mnemosyne were quite similar and they will not be shown. Again, `pathload` provided more accurate estimations than `abget`. The main difference between these results and those obtained in the first set of measures (see Figure 3) is the width of intervals given by `pathload` in the second set. UDP traffic sent at fixed rate decreases the bandwidth availability and, as a consequence, intervals given by `pathload` decrease.



(a) `pathload` estimations (b) `abget` estimations

Fig. 5: Simultaneous executions in the first scenario – eolo case.

The execution time values provide the main difference between the experiments with two simultaneous executions and the experiments with a single execution. Figure 6 shows the execution time for samples collected between the host eolo and the host mnemosyne (similar results were observed between the host cronos and the host mnemosyne). Comparing the execution time when a single estimation was involved (Figure 4(a)), an increase in execution time of the `abget` tool can be observed. The simultaneous executions increased the execution time starting at 4Mbps (UDP) while such increase started only at 8Mbps (UDP) when only one execution was involved.

In general, in the first scenario, `pathload` provided better estimations than `abget`, although `abget` executed faster and was less intrusive. Both tools executed relatively fast ($<$ 2m20s) and with a relatively low level of intrusion ($<$ 6MB) considering the expected amount of data transfer done by grid applications.

3.2 Second scenario: large nominal link capacities

In the second scenario, the tools were evaluated (without using `NCTUns`) in a local network with links of 1Gbps interconnecting the hosts. The `iperf` [16]

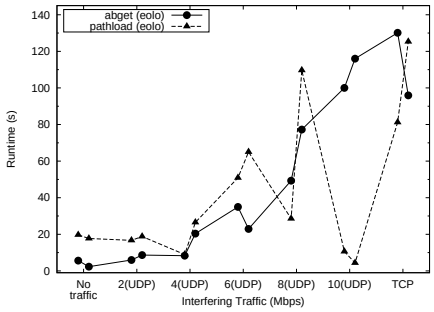


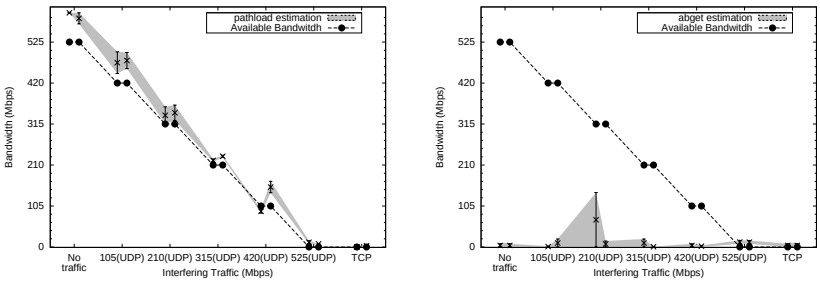
Fig. 6: Execution time of tools (simultaneous execution – eolo case).

tool was employed to generate interfering traffic. Virtual hosts and routers were created in the urano machine (see Figure 2) using virtual network interfaces. Estimations were performed using the same hosts of the first scenario. Although links have nominal capacity of 1Gbps, the observed actual capacity was 525Mbps, so this value was considered as the reference value to the maximum available capacity. The same metrics and steps of the first scenario were considered in this second scenario. The difference is in the rates of UDP interfering traffic: 105, 210, 315, 420, and 525Mbps.

Figure 7 shows the observed accuracy of the tools when estimations were performed only between the hosts cronos and mnemosyne. Estimations provided by `pathload` (Figure 7(a)) were more accurate than those given by `abget` (Figure 7(b)). In contrast with the first scenario, `pathload` estimations clearly track the available bandwidth value. Similarly to the first scenario, `pathload` provided more accurate estimations under higher levels of interfering traffic. Besides that, `abget` presented a different behaviour when compared with that observed in the first scenario. Although the grid links had larger nominal capacity, `abget` estimated that links were almost 100% occupied regardless of the intensity of interfering traffic. Although parameter tuning in `abget` was carried out, no better result was achieved. Actually, the need to set a high number of parameters in `abget` is a major shortcoming, specially when under highly dynamic environments such as grid networks.

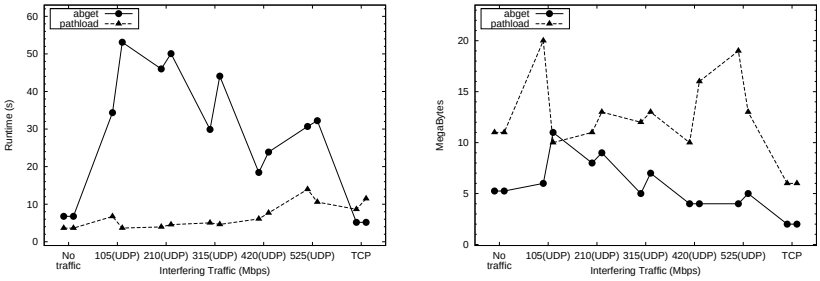
The execution times of `abget` were on average much higher than those of `pathload`, as shown in Figure 8(a). The `pathload` tool took longer periods to execute than `abget` did under the influence of TCP interfering traffic. `pathload` migrates to the next sending rate when it detects instability of the delay values of the first packets sent at a certain rate, while `abget` always sends the same number of packets at all rates. As a consequence, `abget` took a much longer time due to the larger number of rates to cover in the second scenario.

Figure 8(b) shows that `pathload` injected more traffic than `abget`. Moreover, the variability of bytes sent by `abget` was lower than that of `pathload`. Comparing these results with those given in Figure 4(b), it can be observed



(a) `pathload` (b) `abget`

Fig. 7: Estimations provided by `pathload` and `abget` (second scenario).



(a) Execution time (b) Level of intrusion

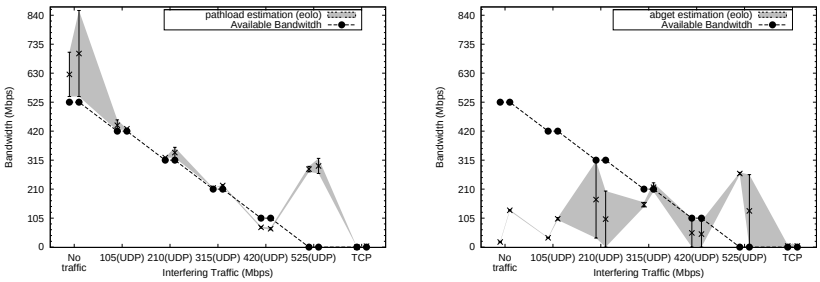
Fig. 8: Performance of the estimation tools in the second scenario.

that the level of intrusion increased with the increase of the link capacity and that the level of intrusion had a larger variability when interfering traffic was composed of UDP traffic. However, when TCP interfering traffic was used, both estimation tools produced more stable results.

Figures 9 to 11 show the results when two simultaneous estimations were pursued. Intervals produced by `abget` were wider (Figure 9) and the `pathload` tool produced estimations that diverged from the real availability when links were congested. The enlargement of the intervals provided by `pathload` also happened when no interfering traffic existed.

Another point to highlight is the significant increase in the execution time of `abget` in congested network. The execution time of `abget` increased by one order of magnitude when UDP interfering traffic grew from 420Mbps to 525Mbps (Figure 10).

It is important to point out the behaviour of `pathload` in the second set of measures, specially when the transmission rate of the UDP interfering traffic was 105Mbps (Figure 11). In one of the executions, this tool generated about 300MB of data, a volume far from being negligible. Such behaviour evinces that available bandwidth estimation tools can be highly intrusive, which moti-



(a) `pathload` estimations (b) `abget` estimations

Fig. 9: Simultaneous executions in the second scenario – eolo case.

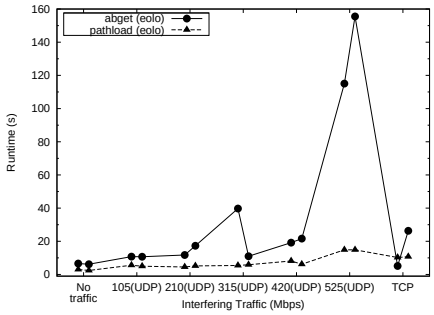


Fig. 10: Execution time of the estimation tools (simultaneous executions – eolo)

vates the implementation of preventive procedures to avoid overhead. A simple solution would be either to request the users a maximum value for the execution time or to quantify the maximum allowed value for the injected bytes. Although external criteria for stop the execution of the estimators may result in less precise estimations, the overall outcome can be preferable if all the involved metrics are jointly considered.

In summary, for the second scenario, `pathload` provided good results even in an environment with high capacity links. The execution time of `pathload` was lower than that of `abget`, although `abget` was less intrusive on the average. Schedulers using `abget` to estimate available bandwidth in this type of scenario would provide schedules that would lead to underutilization of network links due to the underestimation of the available bandwidth. Moreover, the `pathload` tool has the potential to flood the network with traffic that may directly impact other applications in execution.

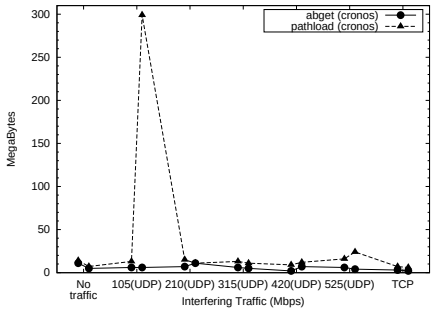


Fig. 11: Level of intrusion of the estimation tools (simultaneous executions – cronos)

4 Conclusion

Resource availability in grids is not only a function of its heterogeneous and decentralized nature but also of the variable level of concurrency found in grid networks. Having a network-based high performance computing environment only renders this scenario more challenging. Estimating the available bandwidth in network links thus plays a key role for scheduling and rescheduling of tasks, specially to highly demanding e-Science applications which are strongly dependent on the transfer of large amount of data.

This paper evaluated the performance of two tools for the estimation of the available bandwidth in network scenarios. Our performance analysis has been driven by the potential integration of these tools into grid systems for helping the decision making process of (re)scheduling the tasks of applications. Several metrics were considered such as estimation accuracy, execution time, level of intrusion, and effectiveness when concurrent estimations were performed.

Overall, `pathload` is the preferable tool for estimating available bandwidth in grid networks. Nevertheless, users and administrators of grid environments should consider the potential impact the estimations can have in other flows during measurements. Furthermore, `pathload` has to be executed in both end hosts, a negative characteristic that might motivate modifications in `abget` to improve its accuracy. The code of `pathload` must also be changed so that the adopted ports can be dynamically defined, allowing the simultaneous use by several pairs of end hosts in large-scale distributed grid environments.

References

1. I. Foster. What is the Grid? A Three Point Checklist. *GRIDToday*, 1(6), Jul 2002. `http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf`. Retrieved January 6, 2009.
2. M. Jain and C. Dovrolis. End-to-end Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. *IEEE/ACM Transactions on Networking (TON)*, 11(4):537–549, Aug 2003.

3. D. M. Batista, A. C. Drummond, and N. L. S. Fonseca. Robust Scheduler for Grid Networks. In *SAC '09: Proceedings of the 2009 ACM Symposium on Applied Computing (to be published)*, pages 354–358, New York, NY, USA, Mar 2009. ACM Press.
4. T. Ito, H. Ohsaki, and M. Imase. On Parameter Tuning of Data Transfer Protocol GridFTP for Wide-Area Grid Computing. In *Proceedings of the BroadNets 2005*, volume 2, pages 1338–1344, Oct 2005.
5. John A. Silvester. CalREN: Advanced Network(s) for Education in California, Oct 2005. `http://isd.usc.edu/~jsilvest/talks-dir/20051021-cudi-merida.pdf`. Retrieved January 6, 2009.
6. WLCG Worldwide LHC Computing Grid, Nov 2008. `http://lcg.web.cern.ch/LCG/public/`. Retrieved January 6, 2009.
7. D. M. Batista, N. L. S. da Fonseca, F. K. Miyazawa, and F. Granelli. Self-Adjustment of Resource Allocation for Grid Applications. *Computer Networks*, 52(9):1762–1781, Jun 2008.
8. Eduardo Huedo, Ruben S. Montero, and Ignacio M. Llrorent. An Experimental Framework for Executing Applications in Dynamic Grid Environments. Technical Report 2002-43, NASA Langley Research Center, 2002.
9. Rubén S. Montero, Eduardo Huedo, and Ignacio M. Llorente. Grid Resource Selection for Opportunistic Job Migration. In *Proceedings of the 9th International Euro-Par Conference*, pages 366–373. Springer Berlin / Heidelberg, 2003.
10. Gabrielle Allen, David Angulo, Ian Foster, Gerd Lanfermann, Chuang Liu, Thomas Radke, Ed Seidel, and John Shalf. The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment. *International Journal of High Performance Computing Applications*, 15(4):345–358, November 2001.
11. D. Antoniades, M. Athanatos, A. Papadogiannakis, E.P. Markatos, and C. Dovrolis. Available Bandwidth Measurement as Simple as Running wget. In *Proceedings of the Passive and Active Measurement Workshop 2006*, 2006.
12. Cooperative Association for Internet Data Analysis (CAIDA). CAIDA : tools : taxonomy, Jun 2008. `http://www.caida.org/tools/taxonomy/performance.xml#bw`. Retrieved January 6, 2009.
13. M. Jain R. S. Prasad and C. Dovrolis. Evaluating Pathrate and Pathload with Realistic Cross-Traffic, 2003. Talk at Bandwidth Estimation Workshop 2003. `http://www.caida.org/workshops/isma/0312/slides/rprasad-best.pdf`. Retrieved January 6, 2009.
14. A. Shriram, M. Murray, Y. Hyun, N. Brownlee, and A. Broido. Comparison of Public End-to-End Bandwidth Estimation Tools on High-Speed Links. In *Passive And Active Network Measurement: 6th International Workshop, PAM 2005*. Springer, Mar 2005.
15. R. Prasad, C. Dovrolis, M. Murray, and K. Claffy. Bandwidth Estimation: Metrics, Measurement Techniques, and Tools. *Network, IEEE*, 17(6):27–35, 2003.
16. NLANR. Iperf, 2008. `http://sourceforge.net/projects/iperf/?abmode=1`. Retrieved January 7, 2009.
17. Ajay Tirumala, Les Cottrell, and Tom Dunigan. Measuring end-to-end bandwidth with Iperf using Web100. Technical Report SLAC-PUB-9733, Stanford Linear Accelerator Center, Apr 2003.
18. Yusuf Ozturk and Manish Kulkarni. DIChirp: Direct Injection Bandwidth Estimation. *Int. J. Netw. Manag.*, 18(5):377–394, 2008.
19. Attila Pásztor and Darryl Veitch. PC-based precision timing without GPS. In *ACM SIGMETRICS*, Los Angeles, CA, USA, June 2002.
20. S. Y. Wang, C. L. Choua, and C. C. Lin. The Design and Implementation of the NCTUns Network Simulation Engine. *Simulation Modelling Practice and Theory*, 15(1):57–81, Jan 2007.
21. Ravi Prasad, Manish Jain, and Constantinos Dovrolis. Effects of Interrupt Coalescence on Network Measurements. In *Proceedings of the Passive and Active Network Measurement Workshop 2004*, pages 247–256, 2004.