A Network Performance Tool for Grid Environments

Craig A. Lee
James Stepanek
Computer Science and Technology
The Aerospace Corporation
{lee,stepanek}@aero.org

Rich Wolski
Department of Computer Science
University of Tennessee, Knoxville
rich@cs.utk.edu

Carl Kesselman
Information Sciences Institute
University of Southern California
carl@isi.edu

Ian Foster
Mathematical and Computer Sciences
Argonne National Laboratory
foster@mcs.anl.gov

Abstract

In grid computing environments, network bandwidth discovery and allocation is a serious issue. Before their applications are running, grid users will need to choose hosts based on available bandwidth. Running applications may need to adapt to a changing set of hosts. Hence, a tool is needed for monitoring network performance that is integral to the grid environment. To address this need, *Gloperf* was developed as part of the Globus grid computing toolkit. Gloperf is designed for ease of deployment and makes simple, end-to-end TCP measurements requiring no special host permissions. Scalability is addressed by a hierarchy of measurements based on group membership and by limiting overhead to a small, acceptable, fixed percentage of the available bandwidth. Since this fixed overhead may push host-pair revisit time into the tens-of-hours, we also quantitatively examine the "trajectory" of the cost-error trade-off for measurement frequency.

1 Introduction

In grid computing environments, network bandwidth discovery and allocation is a serious issue. When starting a grid application, how do users know which hosts have suitable bandwidth among them? In the absence of any type of bandwidth reservation mechanisms, how should applications initially configure and subsequently adapt themselves to prevailing network conditions? While running applications can monitor their own performance, they cannot readily measure performance in other parts of the environment. In light of these considerations, a tool for monitoring network performance in grid computing environments is needed that is part of the grid infrastructure itself. However, performance monitoring is not enough if the information is to be used to make resource allocation decisions. Instead, a scheduler (be it a human being or an automatic scheduling program) must use the performance data that has been collected to *predict* what bandwidth will be available when an application will consume it [1]. Gloperf bandwidth information can be used to make these predictions.

To address these needs, *Gloperf* was developed as part of the Globus grid computing toolkit [15] with consideration of all fundamental issues:

- *Accuracy vs. Intrusiveness.*
- *Scalability.*
- *Portability.*
- *Fault Tolerance.*
- *Security.*
- *Measurement Policy.*
- *Data Discovery, Access and Useability.*

Gloperf is designed for ease of deployment to enhance portability. It makes simple, end-to-end TCP measurements such that no special host permissions or specific knowledge of link topology is required. Besides measuring what applications are more likely to experience, this also enhances Gloperf's fault-tolerance. Data discovery and access is provided by storing Gloperf data in a directory service that uses a well-known naming schema.

Scalability of such a tool is extremely important since grid environments have already been built that encompass thousands of nodes. $O(n^2)$ measurements of all pairs can be avoided by using a hierarchy of host-pair measurements [17]. Monitoring overhead can also be limited simply by measuring less frequently.

When designing and deploying Gloperf, however, we realized that even with hierarchical measurement, the host-pair revisit time may in the tens-of-hours if the total overhead is limited to a small, acceptable, fixed percentage of the available bandwidth. Typically performance prediction error is minimized by attempting to track highly volatile network performance with frequent measurements. Rather than doing this, we endeavored to expose the magnitude of the *prediction error* when the measurement period exceeds twelve hours (which is the anticipate measurement frequency for Gloperf when fully deployed). Hence, besides presenting the design of Gloperf, we also quantitatively examine the "trajectory" of the cost-error trade-off such that users and site administrators can intelligently choose the monitoring overhead that best meets their accuracy requirements while not exceeding their pain threshold.

2 Related Work

Network performance measurement is an extremely broad area and we can only briefly mention some of the more relevant work. Carter and Crovella [3, 2] present *bprobe* and *cprobe* to measure the bottleneck link speed and competing traffic, respectively, on a path using ICMP ECHO packets. Since these tools do not use TCP, they do not capture any TCP-related effects that an application might experience. Van Jacobson's *pathchar* [7] estimates bandwidth on all hops of a path and hence can take a very long time. It also requires root access, making it less desirable for grid environments. *TReno* [10] emulates an idealized TCP which makes the measurements independent of host-specific TCP implementations but not representative of what applications would experience. *TReno* also needs root access. *topology-d* [11] uses *ping* and *netperf* to make measurements between all pairs within a *group* and then computes a minimum-cost *logical topology*. The Network Weather Service (NWS) [17] uses TCP to send small, fixed-size probes measuring in the kilobytes with a frequency that is tunable, but typically ranges from tens of seconds to several minutes. Performance measurement systems, such as the National Internet Measurement Infrastructure (NIMI) project [12], however, are being designed for arbitrary Internet hosts. This work is complemented by the Cooperative Association for Internet Data Analysis (CAIDA) [5] whose goal is to develop metrics and tools for analyzing traffic across the Internet.

Finally we note that the data from such tools are intended to be used by "higher-level" systems. Remos [9] also provides an API whereby applications can pose *flow* and *topology-based queries*. AppLeS (Application Level Scheduler) [14] uses information from the NWS to schedule distributed, resource-intensive applications. Ultimately other Globus components will use Gloperf information in similar resource discovery and allocation functions.

3 The Gloperf Design

Any performance monitoring system can be broken into four broad functional areas: (1) sensors, (2) collation of data, (3) production of derived data, and (4) access and use of data. Gloperf is implemented as a set of *gloperfd* daemons that act as the sensors. Gloperf data is collated into the Globus Metacomputing Directory Service (MDS) [4] which is a Lightweight Directory Access Protocol (LDAP) server [6]. The MDS implements a structured information naming schema, thus allowing any MDS client to access Gloperf data. Gloperf also relies on the MDS for information that allows each *gloperfd* to follow the measurement policy. Each daemon does *peer* and *group discovery* based on MDS information thus determining which peers to make measurements with.

When Globus is started on a host, one *gloperfd* will be started that registers itself by writing an object into the MDS. After registration, *gloperfd* goes into its main control loop that has the following psuedo-code behavior:

```
while ( no termination signal )  {
  query MDS for all gloperfds;
  filter list of gloperfds on version number and groups;
  build a randomized list of tests that does a
      bandwidth test and a latency test to each filtered peer;
  for each element of the list  {
    perform the test;
    write the results in the MDS;
    wait n minutes;
  }
}
```

gloperfd loops until it is told to terminate by an external signal. On each iteration, *gloperfd* queries the MDS for all other *gloperfd*s to do peer and group discovery. Group membership is denoted by the group names appearing in the daemon's MDS object. *gloperfd* only does tests to peers that are in the same groups.

Groups are currently administered by hand. Daemons are in a group "local" to their site by default. Site administrators must promote a local host to also be a member of the "global" group. This simple group mechanism allows an arbitrary hierarchy of measurements to be specified. Once a daemon has identified its peers, it constructs a randomized list of bandwidth and latency tests to each peer.

Gloperf does end-to-end bandwidth and latency measurements based a "librarized" version of *netperf* [8]. Gloperf uses a netperf "TCP_STREAM" test such that TCP/IP overhead is included in the measured throughput. Send and receive buffer sizes are the local system defaults. Hence, the performance observed is what an untuned application would observe. Gloperf uses a netperf "TCP_RR" test to measure latency as the inverse of "transactions per second".

One bandwidth or latency test in the randomized list is done every n minutes. Currently $n = 5$ minutes by default. When both bandwidth and latency tests have been done for a peer, the data is stored with the source host object in the MDS. Note that these tests are unidirectional; different tests are done from both ends of a host pair and written under the source host object.

Once the list of tests has been serviced and no termination signal has been caught, *gloperfd* queries the MDS again since other daemons may have been added to or deleted from the testbed.

Design Discussion.

Gloperf does simple, active, end-to-end measurements that require no special host permissions or site-specific knowledge. It is a decentralized design since each daemon does its own "autoconfiguration" of peer/group discovery and scheduling/reporting of test results. While the Gloperf daemons rely on the MDS, there is no interaction between the daemons other than that required for individual tests, i.e., there is no distributed scheduling of measurements. This simplified, decentralized design enhances Gloperf's fault tolerance since a failed host or link only affects the tests that require those resources. We note that all Gloperf daemons and hosts are monitored by the Globus Heartbeat Monitor (HBM) [13] which can be used to build arbitrary fault-recovery mechanisms.

Scalability and *intrusiveness* are major issues for tools such as Gloperf. Scalability is greatly enhanced by using a measurement hierarchy based on group membership. This reduces the total number of tests and the number of "redundant" tests over shared paths such as the backbones between institutional sites. This does, however, mean that some end-to-end performance has to be estimated from a "path" of end-to-end measurements. While this complicates use of the data, we can at least control the length of the paths via the number of groups.

Overall intrusiveness can be bounded simply by limiting each *gloperfd* to do no more than one test every k minutes. Hence, a five-second test every five minutes limits the overhead to 1.6% on a time-spent-testing basis. This policy has the undesirable property, however, of monotonically increasing the revisit time between any given host pair as the total number of hosts increases. While this effect is ameliorated by the group hierarchy, there are still $O(n^2)$ tests between n local host pairs in a local group. A site with ten hosts, for example, could see a 90-minute revisit time for a given local host pair. Given that we do not want to complicate the deployment and configuration of Gloperf and we want to minimize Gloperf's intrusiveness, how infrequently can we measure and still get useful data?

4 Performance Results

Ideally what is desired are highly accurate measurements that are minimally intrusive. Unfortunately more accurate measurements typically means more frequent measurements which increase the intrusiveness. We can ameliorate the effect of more frequent measurements by using shorter tests. While shorter tests will be less intrusive, TCP may never get out of slow start and the measured bandwidth will be lower than what is actually available.

We examine these questions by comparing the error and cost of measurements for different test times and test periods using a modified, experimental version of Gloperf. This version makes one test per minute between a fixed pair of hosts. A cycle of five test times are used: 1, 2, 5, 7, and 10 seconds. Hence, we collect five, interleaved time series of measurements, one for each test time with five minutes between tests. This allows us to examine the effect of test time. To evaluate the effect of test period, we will compare the error in this five-minute data with the error produced when the same data is subsampled to effectively produce a longer test period. For instance, we can subsample every twelfth point to effectively produce a one-hour test period. In the trace data that follows, we will use subsampling to produce 5, 10, 15, 30, 45, and 60 minute test periods.

Figure 1 shows the bandwidth measurements made with this experimental version of Gloperf between sleipnir.aero.org (near Los Angeles) and yukon.mcs.anl.gov (near Chicago) for approximately a 48 hour period over Los Nettos. While these time series are interleaved (and measure the network at slightly different times), the 1-sec. tests are clearly the lowest on average and longer test times produce a higher measured bandwidth.

The histograms of these time series in Figure 2 clearly show this trend, i.e., TCP coming out of slow start. These histograms also show that the 5, 7, and 10-sec. tests all share the same mode (even though the 10-sec. tests have the highest arithmetic mean by a slight margin). This indicates that tests longer than 5-sec. are not necessary on this link in accordance with the bandwidth-delay product governing TCP slow start.

To assess the "error" in these time series requires some notion of what the "true" values are compared to what Gloperf leads us to believe they are. This requires some form of *prediction model* based on Gloperf data. The simplest prediction model assumes that the last measurement predicts the next and the error is the difference between them. Hence, by using the *last measurement predictor*, we can assess the error by computing the Mean Absolute Error (MAE) and assess

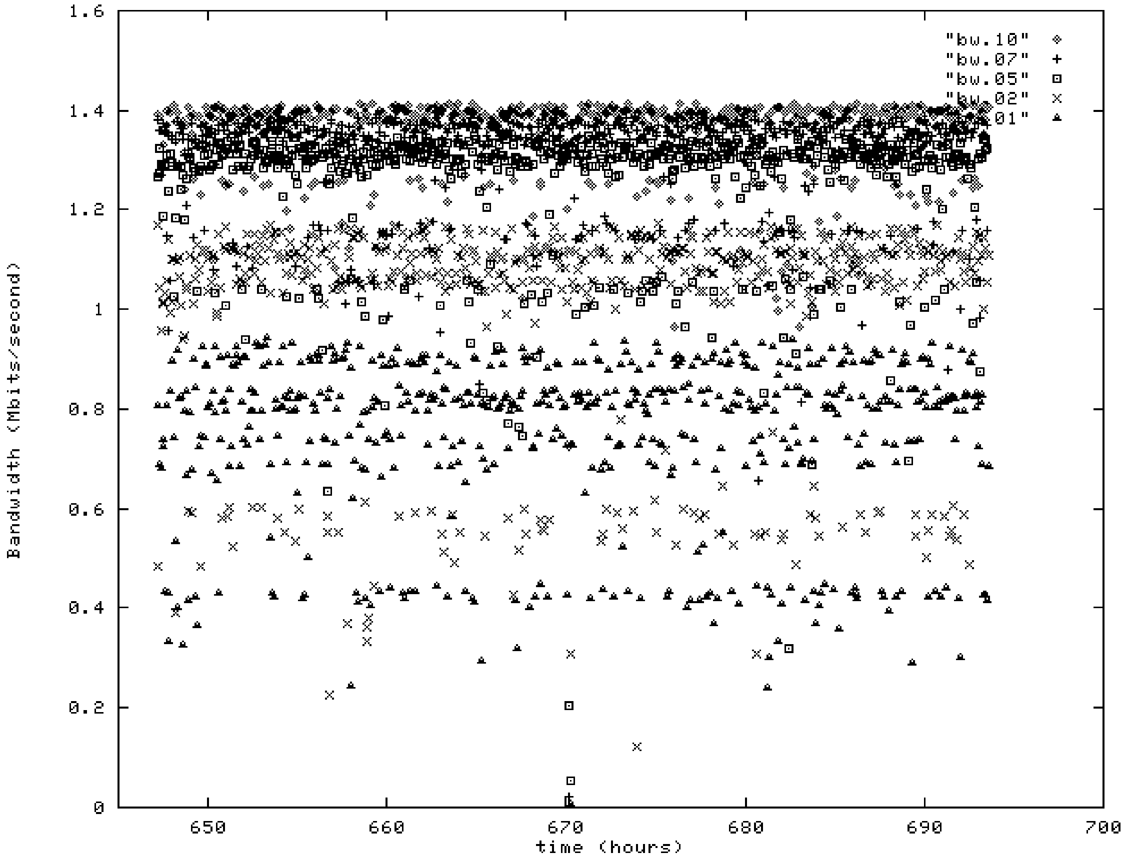
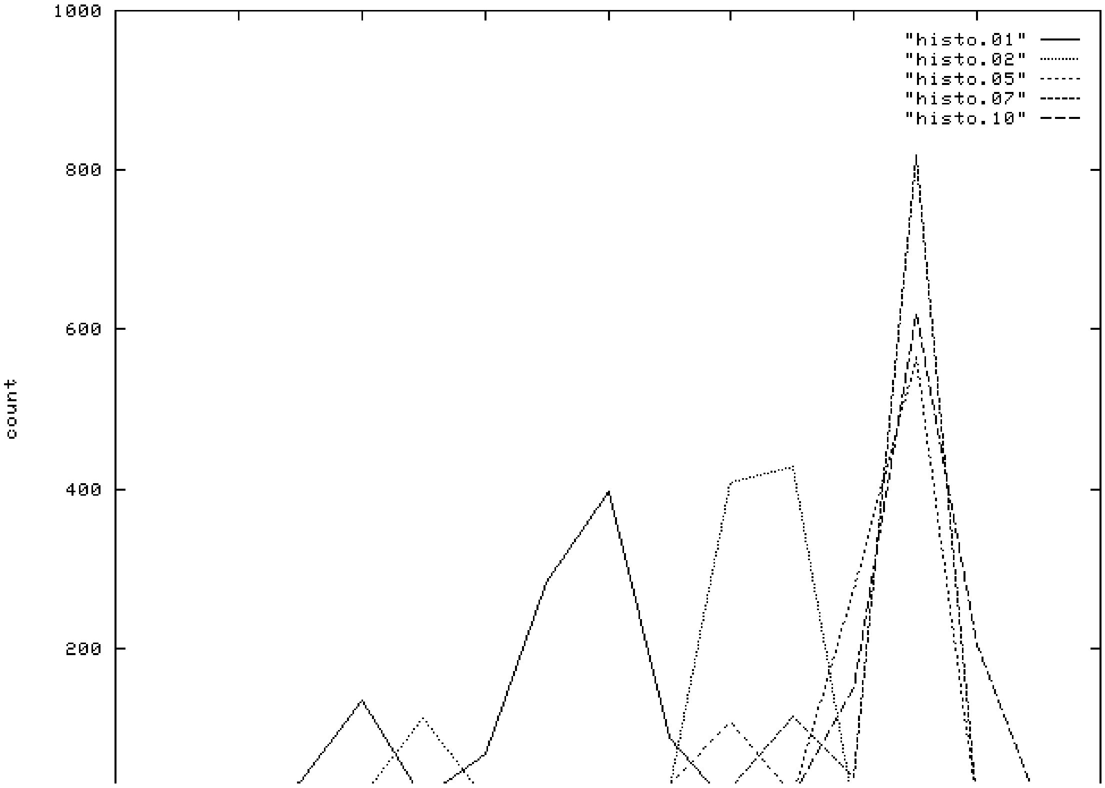


Figure 1: *Representative experimental bandwidth data with stationary means; sleipnir.aero.org to yukon.mcs.anl.gov.*


1000
800
600
400
200
count
"histo.01"
"histo.02"
"histo.05"
"histo.07"
"histo.10"