A Network Topology Mapping Tool For The Grid

Muhammd Murtaza Yousaf
University of the Punjab
Old Campus, New Anarkali, Lahore
murtaza@pucit.edu.pk

Javeria Iqbal
University of the Punjab
Old Campus, New Anarkali, Lahore
javeria@pucit.edu.pk

Michael Wilzl
Department of Informatics
University of Oslo, Norway
michawe@ifi.uio.no

ABSTRAT

In order to avoid that data transfers adversely influence each other, or at least take such knowledge into account, Grid applications should be aware of the network topology. We present a simple tool which measures and provides this information and show that it operates with minimal overhead.

1.INTRODUCTION

A Grid scheduler needs to be aware of the time it takes to transfer a file, and the most common tool that is currently used to obtain this information is the Network Weather Service (NWS) [14], which is shipped with the de facto standard middleware, the Globus toolkit. NWS provides point-to-point information: given a source 'S' and a destination 'D', one can ask the service about the time to transfer a file from 'S' to 'D'. If, however, another source 'S2' would send a file to destination 'D2' at the same time, these two data transfers might share a bottleneck in the network and adversely influence each other. No information of such correlations exists in NWS. By adding information about network links (which can be such bottlenecks) to scheduling decisions, the chance of having data transfers adversely affect each other can be reduced and the overall execution time can improve. To this end, tools for measuring the network topology are needed; this fact has already been accounted for in the literature [13].

We begin this paper with an overview of such related work, and explain its deficiencies that we intend to address with our Network topology Mapping Tool for the Grid (NMTG). We elaborate on the design choices that we made and explain how our mechanism works in section 3, and describe its implementation in section 4. Section 5 provides results from a performance evaluation in the Austrian Grid, and section 6 concludes.

2.RELATED WORK

Many approaches have been proposed to discover internet structure and topology (e.g., [2], [15], [16], [5], [1], [17]) which are mainly based on SNMP, ping, DNS discovery and traceroute [6]. These approaches use some of these tools or a combination of them for topology detection.

FIT'09, December 16–18, 2009, CIIT, Abbottabad, Pakistan.

We found that, despite these tools being readily available, none of these approaches are really suitable for Grids. For instance, any method that uses SNMP requires special privileges from routers, which may be impossible in a Grid environment. For the sake of integration in the regular resource discovery mechanism used in Grids, it must be possible for "normal" end users to invoke a topology discovery mechanism.

The authors of [10] have proposed a Topology Discovery Service (TDS) as a component of Network Information and Monitoring Service (NIMS) [11] for Grid applications. The authors have developed two different implementations for TDS which are designed to work in IP/MPLS router based networks. The first of these implementations is known as Centralized TDS (C-TDS). The main component of the C-TDS architecture is the C-TDS broker.

Discovery mechanism in C-TDS is a four step process. First of all, a topology request is made by the client by an interface provided by the C-TDS broker. As a next step, the C- TDS broker maps and submits the queries to all the routers in the Grid and then gets back XML replies from the routers. Finally, it merges all the information collected from routers into an XML topology file and returns it back to client. In this way it provides information about the physical and MPLS layer topologies, but its main disadvantage is that it needs administrative rights to query the routers. Further, if a Grid environment spans over multiple virtual organizations, which is quite likely, then it is hard to tackle inter-domain issues. The second implementation for TDS is called distributed TDS (D-TDS). The main component of the D-TDS architecture is a Globus Toolkit 3.2 based D-TDS broker which accepts a request from client for topology discovery. In D-TDS sensors are deployed at different hosts in the Grid.

The work done by the authors of [12] is about having a logical view of the network of a Grid in the form of a Direct Acyclic Graph (DAG). This view has network groups and network hosts as the nodes of the DAG. It is not about the physical connections in the network topology.

The edges of the DAG represent the connections between network groups where a network group is a set of hosts with common network characteristics, and a child network group can be assumed to be a sub-network of a major network group.

TopoMon [13] is another tool which provides topology information of the Grid network. It follows the architecture of NWS [14], and its sensors are based on traceroute. Architecturally, TopoMon is quite similar to the tool that we present in this paper, but its overhead is significantly larger.

Rocketfuel [1] is one of the tools which has been deployed and studied on a large scale. Its goal is to infer ISP maps with a reduced number of measurements.

It introduces a technique called direct probing which exploits BGP routing information to select the traceroutes that transit the ISP to be discovered. It also uses another technique, called path reductions, to avoid duplicate traceroutes. This is done by identifying the matching ingress or egress routers in previously completed traceroutes. These techniques helped Rocketfuel to reduce the overhead by a great margin as compared to a Skitter [17] (An earlier tool designed for the similar purpose). Since it relies on BGP, and up-to-date BGP information is typically not available to "normal" network users, Rocketfuel also does not stand our test of applicability in a Grid environment.

3. DESIGN

We have developed this tool as part of a larger infrastructure for Network-Aware Grid scheduling and file transfer delay prediction over the Grid. In this big picture we have an accurate mechanism to detect shared bottlenecks [4] and some methods to extract basic characteristics of network paths (e.g. capacity, delay etc.) in addition to the networking mapping tool for the Grid that is described here. Conjointly, these tools are used to provide a system for reliably predicting the transfer time of files in the Grid. In addition to general engineering principles (such as ensuring scalability, for instance), this framework guided the design decisions that are described in this section.

3.1 Design Considerations

We figured out the following basic requirements for our tool: It should be easy to integrate and adjust well in larger frameworks because the information which is provided by this tool will be used by other applications like the Grid scheduler, resource broker, predictor etc. This tool can be envisioned as a component of the general information infrastructure for Grid resource discovery which is known as Metacomputing Directory Service (MDS). A generic architecture of MDS is depicted in Fig. 1, with our Network topology Mapping Tool for the Grid (NMTG). It is necessary to measure network characteristics for many reasons, including, but not limited to, network planning, service monitoring, cost recovery, and scheduling. All of these functions require up-to-date information about the network, but providing this information is not an easy task because of the highly dynamic nature of the Internet. Thus, it should be a goal to obtain measurements about network characteristics which remain stable for a reasonably long time period. This is the case for the network topology, which tends to be steady until a link upgrade occurs. It is one of the main goals of this tool to provide up-to-date information about this relatively stable characteristic.

It should work with the least possible overhead over the Grid by minimizing its intrusiveness. This ensures that the tool does not affect the other Grid traffic and its client applications. It should be possible to scale with the size of the Grid.

3.1.1 Structure

The structure of our Network Topology Mapping Tool for the Grid is quite the same as the Grid Monitoring Architecture (GMA) [3] and is presented in Fig. 2. To keep our tool simple and to follow one of our design considerations easy to integrate discussed in design considerations, it has been designed with three simple components: Producer, Discovery service and Dummy.

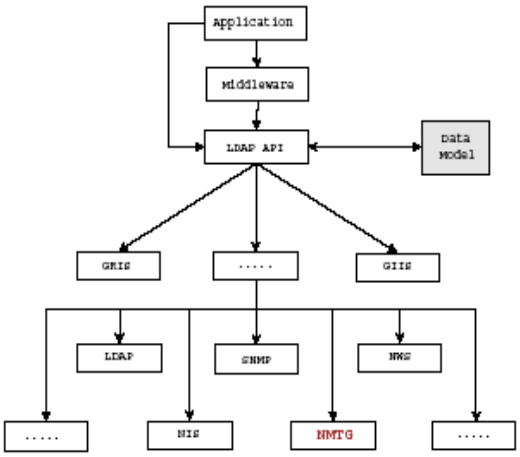


Figure 1. NMTG in generic MDS architecture

3.1.1.1 Producer

Producer gathers data about the Grid network topology and delivers it to the Directory Service in such a way that it just needs to be merged with measurements collected from other 'Producers' without extra processing.

3.1.1.2 Directory Service

It merges the measurements collected from 'Producers' and is responsible for the main computation by initiating the tool. It also has an interface to receive queries from clients and respond to them.

3.1.1.3 Dummy

It is a host which cannot actively take part in the collection of data. In other words, dummies are those hosts that cannot probe the network (but they are used as traceroute destinations).

3.1.2 Approach

Our method is based on traceroute [6] that attempts to trace the path followed by a packet. It works in two major parts which are Full Trace and Smart Trace.

Full Trace is used to initialize the database. First of all it needs a list of all sites of a Grid and look for those sites which have necessary softwares installed to support the generation of network topology of the Grid. Full Trace initiates traceroute from the Directory Service and all 'Producers' to every site in the Grid. This initiation leads to a short term heavy traffic through the system. After getting necessary measurements, all 'Producers' compile their measurements and send it back to the Directory Service in a ready to merge form. The Directory Service then completes computation and generates a map of the network topology for the Grid. The whole approach of Full Trace is summarized in algorithm 1. A Full Trace is initiated once at the start of the initialization of our tool and is followed by frequent Smart Traces to get up-to-date picture of the network topology of the Grid but it can be initiated again anytime because topology may change over time or if the network is found to be available for intrusiveness.

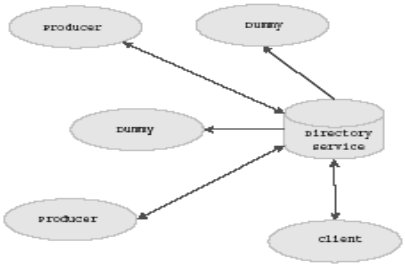


Figure 2. NMTG Structure

Algorithm 1 Full Trace

Input: $G = \{g_1, g_2, g_3, \ldots, g_n\}$ A set of all Grid sites where g_d is a Grid site having Directory Service component of the tool

Output: P: A set of Producers, D: A set of Dummies, and M: A map of Grid topology.

$P = P \cup \{g_d\}$

$D \leftarrow \emptyset$

for $\forall g \in G$ **do**

  if haveTraceroute(g) $\wedge$ havePython(g) **then** $P \leftarrow P \cup \{g\}$

  else $D \leftarrow D \cup \{g\}$

  end if

end for

for $\forall g \; P \wedge {}' g \in G$ **do**

traceroute($g \rightarrow {}' g$)

end for

for $\forall g \in P$ **do**

  sendMeasurements($g \rightarrow g_d$)

end for

generateMap(M)

return M

Smart Trace is actually the core of our tool. It works in such a way that it reduces the overhead of measurements which will be explained in section 5. Before Smart Trace, Full Trace or a Smart Trace should have already run and the Directory Service must have a mapping of the network topology of the Grid.

At the start of Smart Trace, a path is selected with highest number of shared routers from an existing mapping. Such an end-to-end path is selected with the assumption that most of the others path would share it and if we update such a path then we would not have to trace many other paths which actually share the same set of routers. After selecting such a path, we initiate traceroute for this path completely and verify if there have been any routing changes. After that, we initiate traceroute from the other 'Producers' to rest of the nodes in the Grid but we just observe the results of tracerouts for two hops. If no routing change is identified from the result of last trace for the same path then traceroute is terminated immediately which is in fact the smart part of this approach. Typically, networks route according to the destination address, and in this case, any router following a given router X will most probably always be the same – hence, when router X is already in our list for a trace to the same destination, we can terminate. Since router X is usually the first hop along the path, and this router seemed to us to be more likely to have some special configuration than a core router, we decide to never terminate Smart Trace after the first hop, but allow for at least two hops to be detected. There is case when we do not terminate the traceroute for a path even if we do not notice any routing change for two hops from last trace and that is if that path has some network cloud – a portion of the network where routers do not respond to the echo packets of tracerout. We do not terminate traceroute in such situation because it could be possible that for current trace routers may respond and we get a more complete picture of the Grid network topology. The whole method of Smart Trace is summarized in algorithm 2. However, when traffic engineering is used, our assumption of purely destination-based routing is wrong; this is why we do not suggest to configure our tool to strictly rely on only Smart Traces after an initial Full Trace, but occasionally carry out a Full Trace too.

Algorithm 2 Smart Trace

Input: $P = \{g_1, g_2, g_3, \ldots, g_n\}$ A set of all Producers. A source and destination pair (g_s, g_d) of Grid sites with largest number of *shared* routers.

Output:M: An updated map of Grid topology.

traceroute($g_s \rightarrow g_d$)

for $\forall g \in P \wedge {}' g \in G$ **do**

traceroute($g \rightarrow {}' g$)

  if first two routers of path($g_i \rightarrow g_j$) are same AND

    path($g_i \rightarrow g_j$) did not have any could during last trace **then**

    terminate traceroute($g \rightarrow {}' g$)

  end if

end for

for $\forall_i g \in P$ **do**

sendMeasurements($g \rightarrow g_d$)

end for

updateMap(M)

return M

For a better understanding consider a scenario depicted in Fig. 3, having a server host where Directory Service resides, other hosts 'A', 'B', 'C', and 'D' with routers 'a', 'b', 'c', 'd', 'e', 'f', and 'g' and a network cloud representing a path of network where routers do not respond. For a Smart Trace, the path with highest number of routers is selected which is from 'D' to 'B' and traceroute is initiated from 'D'. After this step, a traceroute is initiated from server host to 'B' and we get 'a' and 'c' to be the first two routers as a result which have already been traced in previous trace. But traceroute is not terminated because this path has a possibility that routers in network cloud may respond.

Now consider the situation when traceroute will be initiated from server host to 'A' or from 'D' to 'A'. In these cases, traceroute will be terminated because we will have same first two routers as a result which we had already as a result of Full Trace. If a change is observed in first two routers of traceroute then process is not terminated and later this new information is updated at Directory Service to form a new map of the topology.

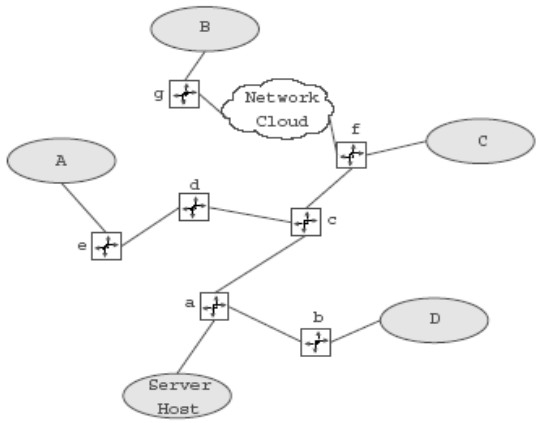


Figure 3. A scenario for topology discovery

4. IMPLEMENTATION

We have built our tool in such a way that we do not need many pre-installation requirements. The main functionality for computational part in Directory Service is implemented in Python, a user interface to configure our tool is designed in java and we have used the standard traceroute to trace the paths in the topology. For the data part, we do not use any special database and work by using simple text files. All these choices make our tool a light weight component which is easy to configure and it adjusts well in larger frameworks which has been a major consideration in our design goals.

4.1 Configuration Files

In this section we discuss some configuration files used in our tool which are simple text files. These are `properties.in`, `hosts.in`, and `hosts_path.in`. `Properties.in` contains paths to the home-directory of Globus installation. This file also contains the frequency of Full Trace and Smart Trace in minutes. Another parameter specified in this file is the server- hostname. A sample `properties.in` is shown in Fig. 4, which shows that a Smart Trace will be initiated after every 10 minutes and a Full Trace will be initiated after every 60 minutes.

```
karwendel.dps.uibk.ac.at#/home/wolf/master#      /host, home
10#                                              /Smart Trace
60#                                              /Full Trace
/home/globus/globus-4.0.2/bin/#                  /globus
```

Figure 4. properties.in

All Grid sites are specified in `hosts.in` which are taken into account for measurement and to create a map of network topology. This includes the hostnames but IP addresses can also be used. The queuing system used at each host is also specified with the hostnames. A `hosts.in` for our experimental set up used in section 5 is presented in Fig. 5.

```
altix1.uibk.ac.at#pbs#
hc-ma.uibk.ac.at#sge#
agrid1.uibk.ac.at#pbs#
schafberg.coma.sbg.ac.at#pbs#
gescher.vcpc.univie.ac.at#pbs#
altix1.jku.austriangrid.at#pbs#
hydra.gup.uni-linz.ac.at#pbs#
```

Figure 5. hosts.in

`hosts_path.in` is responsible to summarize the information about 'Producers' and 'Dummies'. As a first step, all hosts in the `hosts.in` are tested if they have installed Python and traceroute. A request is sent from server-host to all hosts and the collected information is stored in `hosts_path.in`. A sample `hosts_path.in` that was configured for our test bed is depicted in Fig. 6, that shows two hosts can act as 'Producer', the local host as server host and the Dummy hosts. A testing about the installation of Python and traceroute is done at the start of our tool and is not done periodically afterwards, but it can be initiated from graphical user interface any time. It also needs to be initiated if new hosts are added in the setup.

```
altix1.uibk.ac.at#pbs#/home/cb56/cb561080#/usr/bin/python#
hc-ma.uibk.ac.at#sge#
agrid1.uibk.ac.at#pbs#/scratch/cb56/cb561080#
                /bib/traceroute#/usr/bin/python#
schafberg.coma.sbg.ac.at#pbs#
gescher.vcpc.univie.ac.at#pbs#
altix1.jku.austriangrid.at#pbs#
hydra.gup.uni-linz.ac.at#pbs#
```

Figure 6. hosts_path.in

4.2 Full Trace or Smart Trace

For the implementation of Full Trace, first of all, all necessary files for computation and measurement are transferred to all Producers from Directory Service (server host). After that, all 'Producers' and server host initiate their traces to other hosts. When the traces are completed, all 'Producers' merge their respective information and at the end this compiled result is sent to server host. Server host merges the information received by all 'Producers' and performs necessary computation using Python scripts to form `hosts_path.in` as the final mapping of complete network topology of the Grid.

After completing this task a clean up process cleans all the 'Producers'. In the case of Smart Trace, Directory Service computes the path with largest number of shared routers from the up-to-date mapping and trace is initiated for that particular path. After that, tracing is initiated from server host to all hosts and then from all 'Producers' to all hosts. While tracing, 'Producers' send the gathered data set one by one to Directory Service at server host and this data set is compared with the up-to-date mapping to decide if the tracing should continue or it could be stopped as discussed in section 3. If a trace must be stopped then 'Producer' terminates its traceroute by using its PID with "`signal.SIGTERM`".

5. PERFORMANCE EVALUATION

5.1 Testbed

In order to evaluate the performance of our tool in a realistic setting, we have deployed our tool on the Austrian Grid [7] and measured its performance. We used the Grid sites presented in table 1 of the Austrian Grid from which `karwendel.dps.uibk.ac.at` was used as the server host because it has all the necessary components installed. All hosts are using "PBS" [9] except for `hc-ma.uibk.ac.at` which uses "SGE" [8]. Out of all the hosts, only two have Python and *traceroute* installed; these hosts are `altix1.uibk.ac.at` and `agrid1.uibk.ac.at`.

5.2 Results

In this section we discuss the results of our tool. The main focus of these results is about the analysis of overhead caused by the different types of measurements and verification of our design goals. We also wanted to validate the smartness of our Smart Trace and to confirm that it really reduces the intrusiveness of our tool to get an up-to-date mapping of the Grid network topology. In what follows, the term "cloud" refers to one or more unknown routers.

Table 1. Hosts of Austrian Grid

Queuing System	Grid Sites
pbs	karwendel.dps.uibk.ac.at
pbs	altix1.uibk.ac.at
sge	hc-ma.uibk.ac.at
pbs	agrid1.uibk.ac.at
pbs	schafberg.coma.sbg.ac.at
pbs	gescher.vcpc.univie.ac.at
pbs	altix1.jku.austriangrid.at
pbs	hydra.gup.uni-linz.ac.at

In our first experiment we investigated the completeness of our Full Trace.

The results in Fig. 6 show that the percentage of "cloudy" paths (paths containing unknown routers) is reduced with an increasing number of tFull Traces. We initiated 30 Full Traces within a times pan of 3 hours daily for one week. Each measurement in the Figure is the average of all the measurements taken during the whole week.

We can observe that the number of paths with a cloud is decreased to 80% just after the third Trace and it reaches a level of 50% around the seventh Full Trace. This means that successive Full Traces help in resolving the network clouds by regular attempts, thereby contributing to the completeness of the whole topology.

While only Full Traces could be used for thorough topology discovery, such usage causes a lot of overhead which could affect the traffic of other Grid applications. We initiated a Full Trace after every hour from 6:00AM till 7:00PM for one week and observed that the measurement time increases by a factor of almost 2 during mid-day as shown in Fig. 7.

This is clearly due to the load of traffic during daytime that causes queues at routers to be congested, resulting in frequent packet drops.

As we can adjust the frequency of Full Traces, we can conclude that it may not be a good idea to initiate a Full Trace during daytime to avoid overloading the network links.

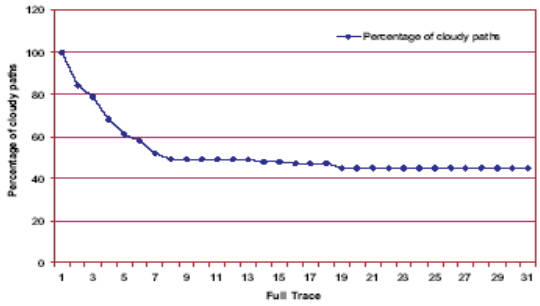


Figure 7. Percentage of cloudy paths in a sequence of Full Traces

The behavior shown in Fig. 7 can be improved by using Smart Trace because this mechanism does not need to trace the full path. One aspect that we wanted to study is how many Full Traces combined with Smart Traces are required to gain an optimal topology map with a minimum number of cloudy paths. To this end, we performed a series of experiments with a varying number of Full Traces among Smart Traces during two different times of the day.

The results are shown in Fig. 8. The graphs show the number of completed traces (i.e. traces that were not interrupted, as it is done with Smart Traces when known routers are encountered again) for consecutive Full and Smart Traces that were carried out. Each measurement in all graphs is the average value of all the measurements collected during two weeks.

At first we initiated a Full Trace followed by a series of Smart Traces. As shown in Fig. 9(a), the number of finished traces can be observed to decrease immediately after the Full Trace. Afterwards, during a series of Smart Traces the number of finished traces decreases relatively slowly. It is again evident that it is better to run measurements early in the morning as compared to choosing a day time for topology discovery for the sake of other Grid applications.

In our next experiment, we initiated a Full Trace followed by three Smart Traces and then again a Full Trace followed by a series of Smart Traces. It can be seen in Fig. 9(b) that the number of finished traces rapidly decreases to 12 after the second Full Trace, which is an improvement over the decrement to 17 from the first Full Trace.

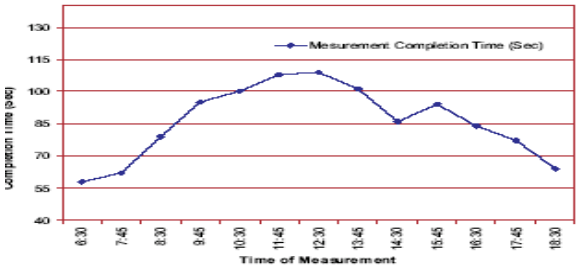


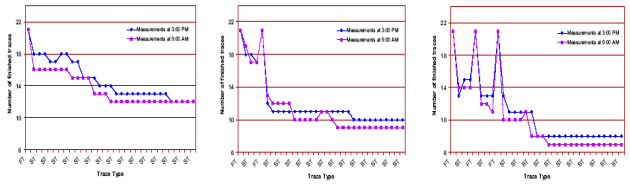
Figure 8. Behavior of Full Trace during the day

This improvement is even more pronounced in Fig. 9(c), where we used a Full Trace three times at the start with a gap of three Smart Traces in between. From this we can conclude that, if the

frequency of Full Trace and Smart Trace is set with some care then we can get efficient results with minimized intrusiveness.

6. CONCLUSION AND FUTURE WORK

In this paper, we have presented NMTG, a Network topology Mapping Tool for the Grid. NMTG measures the network topology in a way that is ideal according to our design goals, of making the system simple (and hence easy to integrate in existing frameworks), let it obtain up-to-date information and make it scalable by minimizing the overhead that it causes. This is achieved by running traceroute in a distributed fashion and terminating it earlier when the full information of traceroute does not seem to be necessary, as the results are likely to be similar to results that were previously obtained. As already mentioned in section 3, we have developed this tool as part of a larger infrastructure for Network-Aware Grid scheduling and file transfer delay prediction over the Grid. Among other things, our complete system include a method for shared bottleneck detection (we already laid the theoretical foundation in [4]). The purpose of this feature is to find out whether a shared link really is a shared bottleneck, i.e. whether correlations exist between flows that share a link. In this context, we intend to use the tool presented in this paper to reduce the number of measurements that need to be carried out: if two source-destination pairs do not even share a link, they surely do not share a bottleneck, and hence there is no need to carry out such a measurement for them.



(a) One Full Trace (b) Two Full Traces (c) Three Full Traces

Figure 9. Reduction of overhead with different combinations of Full Trace and Smart Trace

Our tool could be enhanced in various ways; for instance, we only partially satisfied our own design goal of making the system scalable when we decided to eventually store all information in a central Directory Service. Ideally, one would store our measurement results in an entirely and automatically distributed fashion, as in a P2P system. However, this information is dynamic, and is therefore best stored close to where it was measured. This data dependency seems to make our system a bad fit for standard P2P methods such as a Distributed Hash Table (DHT), and therefore properly disseminating our Directory Service is a nontrivial effort that we leave for future research.

7. REFERENCES

[1] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP Topologies with Rocketfuel. In Proceedings of ACM SIGCOMM '02, Pittsburgh, PA, August 2002.

[2] D. Alderson, L. Li, W. Willinger, and J. Doyle. Understanding Internet Topology: Principles, Models and Validation. IEEE/ACM Transactions on Networking, 13(6), December 2005.

[3] R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, B. Tierney, and R. Wolski. A Grid Monitoring Architecture. Global Grid Forum, PerformanceWorking Group, GridWorking Document GWD-Perf-16-1, 2001.

[4] Muhammad Murtaza Yousaf, Michael Welzl, B¨ulent Yener: "On the Accurate Identification of Network Paths having a Common Bottleneck", poster, accepted for presentation at ACM SIGCOMM 2008, 17-22 August, Seattle, USA.

[5] Brian Eriksson, Paul Barford, Robert Nowak. "Network Discovery from Passive Measurements", accepted for publication at ACM SIGCOMM 2008, 17-22 August, Seattle, USA.

[6] Traceroute,Van Jacobson,ftp://ftp.ee.lbl.gov/traceroute.tar.gz

[7] Austrian Grid Consortium, http://www.austriangrid.at

[8] http://unix-docu.uibk.ac.at/zid/fremddoc/sge/

[9] Portable Batch System (PBS), http://www.doesciencegrid.org/public/pbs/homepage.html

[10] L. Valcarenghi, L. Foschini, F. Paolucci, F. Cugini, and P. Castoldi, Topology discovery services for monitoring the global grid, IEEE Communications Magazine, vol. 44, no. 3, pp. 110117, March 2006.

[11] G. Clapp et al., Grid Network Services, Informational, Grid High-Performance Networking Research Group (GHPN-RG), May 2005.

[12] S. Lacour, C. Prez, and T. Priol. A Network Topology Description Model for Grid Application Deployment, Proc. 5th IEEE/ACM Intl. Wksp. Grid Computing (GRID 2004), Pittsburgh, PA, USA, pp. 6168, Nov. 2004.

[13] M. D. Burger, T. Kielmann, and H. E. Bal, TopoMon: A Monitoring Tool for Grid Network Topology, Proc. Intl. Conf. Computational Science (ICCS 2002), Amsterdam, Apr. 2124, 2002.

[14] R. Wolski, Experiences with Predicting Resource Performance Online in Computational Grid Settings, ACM SIGMETRICS Perf. Evaluation Rev., vol. 30, no. 4, Mar. 2003, pp 4149.

[15] P. Barford, A. Bestavros, J. Byers, and M. Crovella. On the marginal utility of network topology measurements. In Proceedings of ACM Internet Measurement Workshop (IMW '01), San Francisco, CA, October 2001.

[16] R. Govindan and H. Tangmunarunkit. Heuristics for Internet Map Discovery. In Proceedings of IEEE INFOCOM '00, Tel Aviv, Israel, March 2000.

[17] K. Claffy, T. E. Monk, and D. McRobb. Internet tomography. In Nature, January 1999.

[18] A. B. Downey. Using pathchar to estimate Internet link characteristics. In Proceedings of ACM SIGCOMM 1999, Cambridge, MA., September 1999.