Bridging Local and Wide Area Networks for Overlay Distributed File Systems

Michael Closson and Paul Lu
Dept. of Computing Science
University of Alberta
Edmonton, Alberta, T6G 2E8
Canada
{closson|paullu}@cs.ualberta.ca

Abstract

In metacomputing and grid computing, a computational job may execute on a node that is geographically far away from its data files. In such a situation, some of the issues to be resolved are: First, how can the job access its data? Second, how can the high latency and low bandwidth bottlenecks of typical wide-area networks (WANs) be tolerated? Third, how can the deployment of distributed file systems be made easier?

The Trellis Network File System (Trellis NFS) uses a simple, global namespace to provide basic remote data access. Data from any node accessible by Secure Copy can be opened like a file. Aggressive caching strategies for file data and metadata can greatly improve performance across WANs. And, by using a bridging strategy between the well-known Network File System (NFS) and wide-area protocols, the deployment is greatly simplified.

As part of the Third Canadian Internetworked Scientific Supercomputer (CISS-3) experiment, Trellis NFS was used as a distributed file system between high-performance computing (HPC) sites across Canada. CISS-3 ramped up over several months, ran in production mode for over 48 hours, and at its peak, had over 4,000 jobs running concurrently. Typically, there were about 180 concurrent jobs using Trellis NFS. We discuss the functionality, scalability, and benchmarked performance of Trellis NFS. Our hands-on experience with CISS and Trellis NFS has reinforced our design philosophy of layering, overlaying, and bridging systems to provide new functionality.

1 Introduction

When should a system be significantly redesigned? Or, should a more evolutionary approach be taken? Our experience with the Trellis Network File System (Trellis NFS) contributes a data point in support of an evolutionary and layered approach to distributed file systems.

In metacomputing and grid computing, a computational job may execute on a node that is geographically far away from its data files. For proper virtualization and transparency, some kind of remote data access system or distributed file system is required to provide the jobs with access to their data. Historically, the solutions have ranged from explicit stage-in and stage-out of the data [19, 1], to full-fledged distributed file systems [17].

Although it is quite common with batch schedulers (and other systems) to expect the user to explicitly move the data before the job is started (i.e., stage-in) and after the job is completed (i.e., stage-out), it is a substantial burden on the user. In particular, the user has to know in advance all of the data required by the job, which is error-prone. Also, depending on the specific application, the user has to know the algorithm for how the application maps command-line arguments and configuration files to job-specific names for input files and (even more problematically) for output files. Of course, one of the advantages of a real file system is that the application itself can generate, name, and gain access to files as it needs to. Admittedly, when things are perfect, the *de facto* stage-in/stage-out model does work, but a file system is more transparent and more functional.

Full-fledged distributed file systems, including the Andrew File System (AFS), are powerful systems. But, although AFS, and other systems, have many features to deal with performance issues across wide-area networks (WANs), it is not common to see any distributed file system deployed across WANs. The reasons include: First, not every site necessarily uses the same distributed file system. Second, not every site uses the same security model or system. Third, it can be difficult to arrange and maintain a common administrative policy across independent sites. Even when the technical difficulties can be solved, the social issues can veto a common distributed file system across administrative domains.

In the context of metacomputing (and grid computing), the common case scenario is a virtualized compu-

tational resource (aka metacomputer or grid) that spans different administrative domains as well as different geographical locations. The common case scenario is that all sites will not be running the same distributed file system nor the same security system. One solution to the problem of different administrative domains is to require that all participating sites must adopt a new infrastructure, such as the Grid Security Infrastructure (GSI) [5] for security. GSI has functionality, scalability, and robustness advantages. But, there are significant social (and political) reasons why imposing a common infrastructure might be difficult.

A different approach, taken by the Trellis Project [16], is to layer new functionality over selected, existing infrastructure. Whenever possible, the most common wide area and local area protocols and systems should be bridged and overlayed instead of replaced with new systems. We use the term "overlay" to refer to a layering of new functionality without significantly changing the semantics or implementation of the lower layers. On the one hand, maintaining the existing infrastructure makes it harder to make radical changes and gain (hopefully) commensurate radical improvements in functionality and performance. On the other hand, layering and overlaying are classic strategies that allow for easier deployment (especially across administrative domains) and compatibility with existing systems and applications.

However, the experience of the Trellis Project is that the overlay strategy works in practice. Our design objectives have been reinforced by our experience with the system, especially:

1. As much as possible, new functionality should be implemented at the user-level, instead of requiring superuser privileges and administrative intervention. When deploying systems across administrative domains, the need for superuser intervention is a significant liability. When unavoidable, superuser intervention should be as minimal and familiar as possible. For example, Trellis NFS does require a superuser to create the per-NFS-client machine mount point.

2. Existing systems, such as Secure Shell/Secure Copy and the Network File System (NFS), are flexible and robust enough to form the basis for new WAN-based systems.

3. The challenge lies in how to integrate the different components of an overlay system without having to replace the components. Interfaces and mechanisms for cooperation between systems can solve most problems.

2 The Trellis Project

The Trellis Project is attempting to create a software infrastructure to support *overlay metacomputing*: user-level aggregations of computing resources from different administrative domains. The research problems addressed by Trellis include scheduling policies [18, 14], wide area security [13], file system design [6], and new interfaces between different metacomputing components and applications.

In contrast to grid computing [9], overlay metacomputing requires minimal support from systems administrators. Trellis is implemented at the user-level and adds a thin layer of software over widely-deployed systems such as Secure Shell, Secure Copy, and NFS. It is an open research question as to whether a radical redesign of client-server (and peer-to-peer) computing (e.g., service-oriented architecture, new application programming interfaces (API), new software toolkits), as advocated by grid computing, is required or if an evolutionary and overlay approach will work equally well.

At least in the focused application domain of high-performance computing (HPC), Trellis and the overlay metacomputing approach has had some significant demonstrations of functionality and scalability. A series of experiments, dubbed the Canadian Internetworked Scientific Supercomputer (CISS), have used the Trellis system to solve real scientific problems, and aggregate thousands of processors and many administrative domains. With each subsequent experiment, the Trellis system has evolved with new functionality and redesigns based on the lessons learned. From 2002 to 2004, the experiments were:

1. CISS-1 [19]: November 4, 2002. The Trellis system aggregated 1,376 processors, within 18 administrative domains, at 16 different institutions across Canada, from the West Coast to the East Coast. A computational chemistry experiment (using MOLPRO) involving chiral molecules was computed over a continuous 24-hour production run. Over 7,500 jobs and over 3 CPU-years worth of computation were completed.

   CISS-1 proved that a global HPC job scheduler could be implemented entirely at the user-level, without the need for a job broker or resource discovery, via a pull-based model known as *placeholder scheduling* [18]. The most significant achievement of CISS-1 was the ability to aggregate 18 administrative domains, where each domain was only required to provide a normal, user-level account. When relying on the cooperation of different institutions, the fewer the requirements to participate, the more likely it is for them to agree.

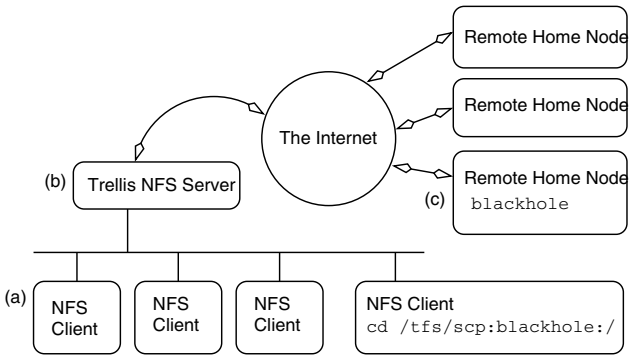

Figure 1: Trellis NFS. These components are (a) the NFS client(s), (b) the NFS server and (c) remote data storage server.

2. CISS-2: December 23, 2002 to January 2, 2003. The Trellis system aggregated hundreds of processors to complete two application workloads. A molecular dynamics problem was computed using GROMACS and a physics problem was computed using custom software.

   CISS-2 proved the usability of the Trellis software over an extended run and for multiple workloads.

3. CISS-3: September 15 to 17, 2004. Thousands of jobs were completed in the ramp-up to CISS-3 from April to September 2004. Then, in a 48-hour production run, over 15 CPU-years of computation were completed, using over 4,100 processors, from 19 universities and institutions. At its peak, over 4,000 jobs were running concurrently, including 180 jobs using the new distributed file system, Trellis NFS. Two different, unmodified binary applications were used: GROMACS and CHARMM.

   CISS-3 demonstrated the functionality and scalability of the Trellis NFS system. CISS-3 was also a proof-of-concept for the new Trellis Security Infrastructure (TSI) [13]. For CISS-3, TSI was developed to provide user-level, secure, scalable, single sign-on functionality for both interactive and background jobs.

3 Trellis NFS

The focus of this paper is the Trellis NFS file system. In providing a file system, the Trellis metacomputing system has specific advantages over the stage-in/stage-out approach. First, the application itself can name and access files on demand (e.g., generate filenames by adding suffixes to command-line arguments). Second, the system (and not the user) is responsible for data movement. Third, aggressive caching strategies can be implemented in a transparent manner.

The basic architecture of Trellis NFS is shown in Figure 1. Note that the NFS client is unmodified. Therefore, unmodified binary applications can use Trellis NFS. A similar strategy was used by the PUNCH Virtual File System [8], although with different WAN transport strategies, different mapping strategies between users and accounts, and different security mechanisms.

Running plain NFS over a WAN is not practical for several reasons. One reason is that the NFS protocol uses short, synchronous messages. Due to this, WAN latency renders NFS unsuitable. We traced the execution of a single invocation of the MAB60 (Modified Andrew Benchmark) [11] benchmark over NFS and counted 106,181 RPC calls which averaged 889 bytes in length for requests and 156 bytes in length for responses. Using the `netperf` [12] tool we measured a request-response rate of 4,537 transactions per second (TPS) for a 100Mbps Ethernet and 17 TPS for the optical network between the University of Alberta and the University of New Brunswick.

The Trellis NFS server is based on Linux's UNFSD server [20]. The server runs at the user-level, but a systems administrator must create the volume mount points on the NFS clients. The mount points can be shared among all processes running on the NFS client. This system administrator involvement is a small but unavoidable violation of Trellis' user-level strategy. By integrating the UNFSD server with the Trellis File System library [21], the modified UNFSD server can access files on any WAN node accessible by Secure Copy and serve the files to unmodified NFS clients with NFS semantics. The global names used by Trellis NFS (see Figure 1, NFS Client), such as `/tfs/scp:blackhole.westgrid.ca:/data/file1`, are recognizable as Secure Copy inspired names, and are known as Secure Copy Locators (SCL).

The Trellis NFS server uses aggressive data caching to overcome the effects of running a file system over a high-latency network. A local disk (or file system) is used as a cache, called the Trellis cache, in a manner similar to Web caches, although the data in Trellis can be read-write files. Files are copied into the Trellis cache on demand. Because an NFS server cannot know when a client has closed a file, a timeout feature is used to schedule the copy-back of the file to the home node, but we are experimenting with the Trellis scheduler telling Trellis NFS when the copy-back should occur. Thus, Secure Copy is the WAN protocol for accessing files and the normal NFS RPC protocol is what Trellis NFS uses to serve NFS clients on the LAN.

For HPC workloads, the common case is whole-file access and caching, and thus Secure Copy is used. For sparse updates of files, it is also possible to transparently use `rsync`-over-`ssh` to move the data in an efficient way, but our HPC workloads do not (in general) benefit from `rsync`'s optimizations.

Through TSI [13], Trellis NFS supports multiple users. As with normal NFS, security on Trellis NFS's LAN-client side is based on the (implicit) security of the local nodes and local user identities. For WAN security, TSI and Trellis NFS uses Secure Shell's agents and public-key authentication (and authorization). Trellis NFS security over a WAN is as secure as using Secure Shell to access remote accounts and, for example, using the Concurrent Version System (CVS) over Secure Shell to share a CVS repository between different users [3]. There are some practical and theoretical security problems with NFS, but NFS is still widely used. Rather than try to replace NFS, Trellis tries to bridge NFS and the WAN.

A newly developed Trellis SAMBA server provides the same functionality as Trellis NFS, but with SAMBA's [2] per-user authentication and the ability to detect when a file is closed. With both Trellis NFS and Trellis SAMBA, the bridging strategy is the key design decision. We focus here on Trellis NFS because of its relative maturity and track record with CISS-3.

3.1 Trellis NFS During CISS-3

As discussed above, we used the Trellis NFS server as part of CISS-3. The CISS-3 experiment included two applications: First, GROMACS [15] is a molecular dynamics simulator. Second, CHARMM [4] is a macromolecular simulator written in Fortran.

The Trellis NFS *server* was used in two administrative domains participating in the CISS-3 experiment. Each server handled multiple NFS clients on a local LAN. The first site was a cluster at the University of Alberta; a 20-node, 40-processor Linux cluster. The second site was a cluster at the University of New Brunswick; an 80-node, 160-processor Linux cluster (Table 1). Between the two domains, 200 jobs were using Trellis NFS at the peak, but the average was 180 jobs. The input and output data for the GROMACS application (i.e., first home node) were stored on a server at the University of Calgary. The input and output data for the CHARMM application were stored on a server at Simon Fraser University (i.e., second home node). Thus, for CISS-3, Trellis NFS was running across nodes in a total of four administrative domains, in three Canadian provinces, and separated by thousands of kilometres of WAN.

Statistic	Average per Hour
Number of NFS Clients (2 jobs per client)	68 to 73
LAN Data Written	200 MB
LAN Data Read	75 MB
LAN RPCs	40,000

Table 1: Trellis NFS Summary for New Brunswick site during CISS-3. All statistics are per-hour.

We are analyzing our trace data from CISS-3. Table 1 is based on an initial analysis of the LAN traffic at the New Brunswick site. The Trellis NFS traffic is determined by the workload and applications themselves. Both applications, like many scientific applications, have a burst of read-only activity at job start-up time and a burst of write-only activity at job exit time. Very few of the LAN RPCs, which are mostly NFS `getattr` RPCs, result in WAN RPCs due to Trellis NFS's cache [6]. Trace data from the New Brunswick site shows one WAN `getattr` RPC for every 360 cache-served LAN `getattr` RPCs. For these workloads, the WAN read and write statistics are likely very similar to the LAN statistics, given the read-only, write-only patterns.

3.2 Micro-benchmark: Bonnie++

The Bonnie++ micro-benchmark [7] is normally used to evaluate local disk subsystem performance. We use Bonnie++ primarily as a standardized workload to do an online measurement of Trellis NFS's performance. There are 3 stages in the Bonnie++ benchmark: write, read, and re-write. First, three 1 gigabyte files are created and written using the `write()` system call. Second, the 3 gigabytes of data is read back using the `read()` system call. Third, the 3 gigabytes of data is split into 16 KB pages; each page is read, modified and re-written (using `lseek()`).

Table 2 shows the throughput of the read, write and re-write tests, including (in line (c)) the additional MD5 computation and data transfer overheads required when

Configuration	Read	Write	Re-write
Local Disk	55.4 (2.8)	23.3 (0.36)	15.5 (0.32)
UNFSD (baseline)	24.1 (0.57)	22.1 (0.62)	7.6 (0.23)
(a) Trellis NFS over LAN	23.9 (0.79)	22.3 (0.54)	7.7 (0.13)
(b) Trellis NFS over WAN	24.4 (0.61)	22.5 (0.54)	7.6 (0.13)
(c) Trellis NFS over LAN; cold cache, flush cache	7.0 (2.87)	5.3 (2.15)	3.1 (1.23)

Table 2: Bonnie++ throughput. All results are in megabytes per second, averaged over 10 runs, standard deviation is in parentheses. Higher numbers are better. (a) and (b) are from NFS client's point of view, with warm caches and no data flush. (c) is for end-to-end performance, including a cache miss, flushing data to home node, and MD5 hash to check data integrity.

moving data in and out of cache. Although the overheads are a key part of Trellis NFS, the main bottleneck is the WAN itself and all distributed file systems would experience similar overheads since they would have the same WAN bottleneck. From the NFS client's point of view, the overheads can be overlapped with computation and amortized over a multi-hour run, so it is useful to also measure the performance from the NFS client's point of view (lines (a) and (b) of Table 2). The marginally better performance of Trellis NFS over WAN (line (b)) as compared to over a LAN (line (a)) is within measurement noise. Not surprisingly, local disk read performance is about 2.3 times faster then the NFS configurations. We include the local disk for perspective. The write test shows all four test configurations have almost equal performance, when working out of cache, which is the common case after the initial start of most HPC applications.

The key conclusion is that, once Trellis NFS has the file data in its cache, the performance of Trellis NFS is comparable to UNFSD. But, when the cache is cold, the performance is bottlenecked by the WAN for data movement. Future work will look at reducing the number of cold misses in the Trellis cache, but the current situation is both functional and reasonable, especially when compared to the UNFSD baseline.

4 Concluding Remarks

Trellis NFS bridges the LAN-based NFS protocol with WAN-based protocols, like Secure Copy, to provide a distributed file system that is relatively easy to deploy because it is (mostly) at the user-level and it is overlayed on top of existing systems. As with the other elements of the Trellis Project, Trellis NFS has been tested in a variety of Canada-wide experiments, known as CISS, that serve to provide empirical evaluations and feedback to the design of the Trellis system.

For future work, we plan to address some of the shortcomings of Trellis NFS and the Trellis system in general. First, the cache consistency model of Trellis NFS is, perhaps, too simple. Currently, there is no locking mechanism for multiple writers of the same file to synchronize their actions. The last job to "close" the file will overwrite all of the updates made by previous jobs. Fortunately, in the HPC application domain, it is rare for applications to actively read-write the same file. Jobs are often well-partitioned into independent, job-specific input (e.g., `config.1-2-1.input`) and output (e.g., `config.1-2-1.output`) files. However, other application domains and file systems (e.g., the append operation of the Google File System [10]) may require some form of file consistency control or multiple-writer semantics.

Second, as previously mentioned, we are implementing a version of the Trellis File System using SAMBA [2] instead of NFS. Among the advantages of using SAMBA include the ability to have per-user authentication to the SAMBA server (instead of NFS's per-client machine security model), avoiding superuser-created mount points (since *some* SAMBA clients allow unprivileged users to create mount points), and the ability of the SAMBA server to see when files are closed, which will make it easier to develop cache consistency strategies. SAMBA's stackable Virtual File System (VFS) mechanism is also a cleaner implementation technique, as compared to modifying a user-level NFS server directly. Of course, the Trellis SAMBA server is consistent with the bridging strategy of the Trellis NFS implementation.

5 Acknowledgments

This research is supported by the Natural Science and Engineering Research Council of Canada (NSERC), SGI, the Alberta Science and Research Authority (ASRA), and Sun Microsystems. Thank you to the Trellis and CISS teams. Thank you to the referees and to Adriana Iamnitchi, our shepherd, for their insightful and helpful suggestions.

References

[1] Portable Batch System. http://www.openpbs.org.

[2] SAMBA. http://www.samba.org.

[3] D. J. Barrett and R. E. Silverman. *SSH, the Secure Shell: The Definitive Guide*. O'Reilly, 2001.

[4] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus. CHARMM: A Program for Macromolecular Energy, Minimization, and Dynamics Calculations. *Journal of Computational Chemistry*, 4:187–217, 1983.

[5] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch. A National-Scale Authentication Infrastructure. *IEEE Computer*, 33(12):60–66, 2000.

[6] M. Closson. The Trellis Network File System. Master's thesis, Department of Computing Science, University of Alberta, 2004.

[7] R. Coker. The Bonnie++ benchmark. http://www.coker.com.au/bonnie++/.

[8] R.J. Figueiredo, N.H. Kapadia, and J.A.B. Fortes. The PUNCH Virtual File System: Seamless Access to Decentralized Storage Services in a Computational Grid. In *10th IEEE Int'l Symposium on High Performance Distributed Computing*, San Francisco, California, 2001.

[9] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, 2002. Open Grid Service Infrastructure WG, Global Grid Forum.

[10] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. In *19th ACM Symposium on Operating Systems Principles (SOSP)*, pages 29–43, Bolton Landing, NY, U.S.A., October 19–22 2003.

[11] J.H. Howard, M.L. Kazar, S.G.Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M.J. West. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems*, 6(1), February 1988.

[12] R. Jones. Netperf. http://www.netperf.org/netperf/NetperfPage.html.

[13] M. Kan, D. Ngo, M. Lee, P. Lu, N. Bard, M. Closson, M. Ding, M. Goldenberg, N. Lamb, R. Senda, E. Sumbar, and Y. Wang. The Trellis Security Infrastructure: A Layered Approach to Overlay Metacomputers. In *The 18th International Symposium on High Performance Computing Systems and Applications*, 2004.

[14] N. Lamb. Data-Conscious Scheduling of Workflows in Metacomputers. Master's thesis, Department of Computing Science, University of Alberta, 2005.

[15] E. Lindahl, B. Hess, and D. van der Spoel. GROMACS 3.0: A package for molecular simulation and trajectory analysis. *J. Mol. Mod.*, 7:306–317, 2001.

[16] P. Lu. The Trellis Project. http://www.cs.ualberta.ca/~paullu/Trellis/.

[17] J.H. Morris, M. Satyanarayanan, M.H. Conner, J.H. Howard, D.S.H. Rosenthal, and F.D. Smith. Andrew: A Distributed Personal Computing Environment. *Communications of the ACM*, 29(4):184–201, 1986.

[18] C. Pinchak, P. Lu, and M. Goldenberg. Practical Heterogeneous Placeholder Scheduling in Overlay Metacomputers: Early Experiences. In *Proceedings of the 8th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, pages 85–105, Edinburgh, Scotland, UK, July 24 2002. Also published in Springer-Verlag LNCS 2537 (2003), pages 205–228.

[19] C. Pinchak, P. Lu, J. Schaeffer, and M. Goldenberg. The Canadian Internetworked Scientific Supercomputer. In *17th International Symposium on High Performance Computing Systems and Applications (HPCS)*, pages 193–199, Sherbrooke, Quebec, Canada, May 11–14 2003.

[20] M. Shand, D. Becker, R. Sladkey, O. Zborowski, F. van Kempen, and O. Kirch. The Linux UNFSD Server. ftp://linux.mathematik.tu-darmstadt.de/pub/linux/people/okir/.

[21] J. Siegel and P. Lu. User-Level Remote Data Access in Overlay Metacomputers. In *Proceedings of the 4th IEEE International Conference on Cluster Computing (Cluster 2002)*, pages 480–483, Chicago, Illinois, USA, September 23–36 2002.