On the Performance and Scalability of a Data Mirroring Approach for I2-DSI

Bert J. Dempsey, Debra Weiss

University of North Carolina at Chapel Hill
dempsey@cs.unc.edu

Abstract

This paper presents work on scaleable design for the automated synchronization of large collections of files replicated across multiple hosts. Unlike conventional mirroring tools, our approach addresses multiple-site file synchronization by capturing file-tree update information in an output file during an initial file synchronization session. Once the update file is available, it can be transmitted over the network using parallel point-to-point file transfers or reliable multicasting and then processed at the remote sites. This paper outlines of how the above concept has been implemented as a modification to the open-source mirroring tool, rsync. It then presents performance experiments designed to characterize the server-side processing costs and network throughput requirements under realistic workloads on large storage servers. The performance experiments use the WAN testbed of the Internet2 Distributed Storage Infrastructure (I2-DSI) project, the context for this work, and the results provide guidance on the scalability of I2-DSI using the proposed data mirroring scheme.

Keywords: mirroring, distributed storage, replication, Internet2, reliable multicast, I2-DSI.

1. Introduction

Client-server applications on the Internet benefit from the use of content replication through improved access for widely distributed clients and balancing load across different servers and network paths. Content replication can be employed either in front of the server (proxy-based caching of server responses) or behind the server (mirroring of source objects). On-demand replication of server responses at proxy-based caches is widely deployed within the current WWW where hierarchical or distributed arrangements of cooperating caches handle HTTP requests and, less commonly, those of other access protocols. While an important component of the current and future Internet, proxy-based caching is tied to specific protocols and has well-studied performance limitations. Problematic aspects of proxy caching include delays and bottlenecks caused by hierarchical processing of client requests, the limits of caching dynamically generated server responses, and the tendency of some content providers to eschew caching in order to retain server-side control [1].

Mirroring source objects across server platforms is another approach to content replication. Among its key advantages are (1) the ability to replicate server-side functionality such as server-managed secure logins, (2) replication benefits for any client-server protocol, not just HTTP services, and (3) a basis for constructing manageable, deterministic control of the replication process for the content provider and service provider. In the current Internet, server-side replication is most commonly employed for (1) the limited (but valuable) case of sharing highly popular static file archives served by FTP or HTTP or for (2) more general replication between servers controlled by a single organization where strong uniformity can be imposed on the server platforms.

The context of our work is the Internet2 Distributed Storage Initiative (I2-DSI) project [2], which is developing replication middleware to extend source-object replication to loosely-coupled, heterogeneous platforms. I2-DSI relies on a set of dedicated, geographically distributed replication hosts (I2-DSI servers) in the network on which user content will be hosted. Application developers publish collections of source objects (I2-DSI content channels [3]) in conjunction with the I2-DSI interface describing supported content types and the standard services available on the replication hosts. Current content channels are linked at [2]. As part of Internet2, the I2-DSI project aims to develop an open architecture for replication services to serve the 140 Internet2 universities and, subsequently, the wider Internet community. Proprietary solutions for application hosting and replication are now appearing in commercial efforts [4] [5].

In this paper a design is presented for scaleable replication of large collections of files in support of file-oriented replication across the I2-DSI servers.[1] Our design is a novel adaptation of an existing data mirroring tool, rsync, that runs in user space and manages data replication through operating system calls to the filesystem. Alternative approaches include block-level management of file replication either at the middleware layer [6] or within a distributed filesystem such as NFS or AFS. A primary advantage of our approach is that it provides a highly portable and easily modified data mirroring mechanism. Application-level solutions, however, are unlikely to match the throughput achievable with kernel-level solutions. This paper provides empirical evidence that the file replication solution proposed here for I2-DSI will provide adequate performance for the current I2-DSI testbed and scale up with the number of I2-DSI replication hosts.

The rest of the paper proceeds as follows. Section 2 presents the motivation and design of a novel file mirroring approach and the scenario for its use within I2-DSI. In Section 3 performance aspects of the solution are explored through controlled experiments that measure server load and the available network bandwidth using the I2-DSI project testbed. Section 4 gives additional discussion of related work, and Section 5 summarizes the findings of the paper.

2 Rsync+

The data mirroring tool developed here, rsync+, is a modification to the open-source file-mirroring tool, rsync. Rsync [7] is a widely used data mirroring tool designed to automate synchronization of file hierarchies residing on the same machine or, alternatively, on machines across the Internet. Rsync has gained popularity due to its rich feature set for controlling the synchronization process, and its open-source C code implementation has enabled it to be ported to many Unix platforms and MS NT. A distinctive feature of rsync (and thus rsync+) is its use of block-level checksumming to perform differential file update when an existing file has

[1] I2-DSI ultimately expects to support multiple replication mechanisms in order to enable the largest possible set of applications to benefit from its replication services.

been updated, i.e., has a new disk image under the same file name.

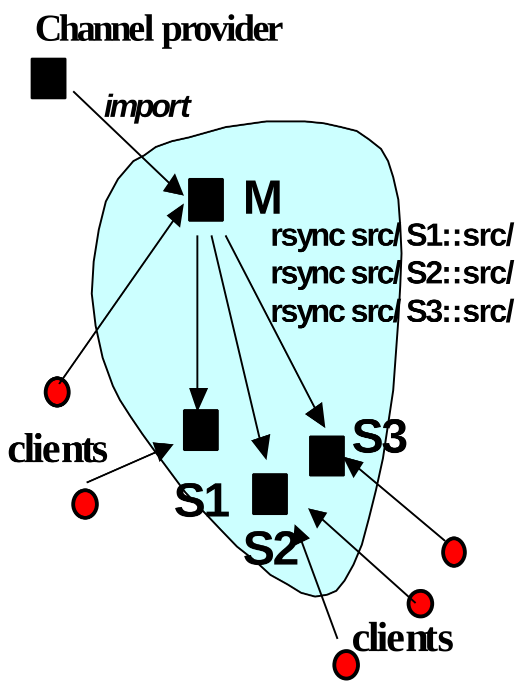


Figure 2.1: An Rsync Scenario for Content Replication in I2-DSI

In its current form, rsync synchronizes remote directory hierarchies during an interactive network session involving two sites, the master site and a slave. Thus, for example, its use in the I2-DSI context would follow the scenario shown in Figure 2.1. In the scenario shown, a channel provider places new or modified files for a file-based channel stored in the directory, **src/,** on the master I2-DSI server, denoted as **M.** For our purposes, the import mechanism here is not specified. In the current I2-DSI architecture, modifications to the source objects in a channel are localized to a master channel copy at one I2-DSI server, here **M**. The set of all other I2-DSI replication servers on which the channel is carried are shown within the shaded area as **S1**, **S2**, and **S3**. On **M** then, a script managing channel replication evokes a set of rsync sessions, one for each remote I2-DSI replication host that carries the channel. As shown, the clients for this I2-DSI channel access the replication hosts nearest to them using resolution mechanisms that are part of the I2-DSI replication architecture.

The modifications developed for r*sync+* are a response to two factors in the scenario in Figure 2.1 that result in inefficiency and limit scalability. First, M performs the same processing on the **src/** directory for each of the three rsync sessions required to update the slave hosts (S1, S2, and S3). This processing includes checking file status information and, if a file has been modified, computing checksums, and thus it can be compute-intensive for large file trees and/or large individual file objects. With multiple point-to-point *rsync* sessions, the processing load on M increases with the number of remote servers being updated. Secondly, M transmits identical data streams over the network in each of the three master-slave *rsync* sessions (S1, S2, S3).

To enable efficient multiple-site synchronization, the rsync source code was modified to create new modes of operation, as shown in Figure 2.2. Options in rsync+ (the **–f** and **–F** flags) enable the end-to-end update process to be decomposed into three independent actions:

Step 1: the generation of replica update information at the master site,
Step 2: the network transport of update information from the master to the slave sites, and
Step 3: the processing of the update information at each slave

site in order to synchronize the slave's files with those at the master.

The key feature of rsync+ is then that it decouples network communication (Step 2) from filesystem update actions (Steps 1 and 3). This decoupling eliminates redundant processing at the master site and enables the use of parallel network (TCP) connections or a reliable multicast transport protocol to deliver data (Step 2) from a single master site to the multiple slave sites, resulting in a scaleable approach to multiple-site file synchronization.

In Figure 2.2, it should be noted that the directory **srcMaster/** that contains the latest updates may be located on **M** (as suggested in the Figure) or at a remote machine. In particular, **srcMaster/** can be on the channel provider machine and rsync+ used as the import mechanism for updating **src/** on **M**. Full details on the rsync+ implementation are in [8].

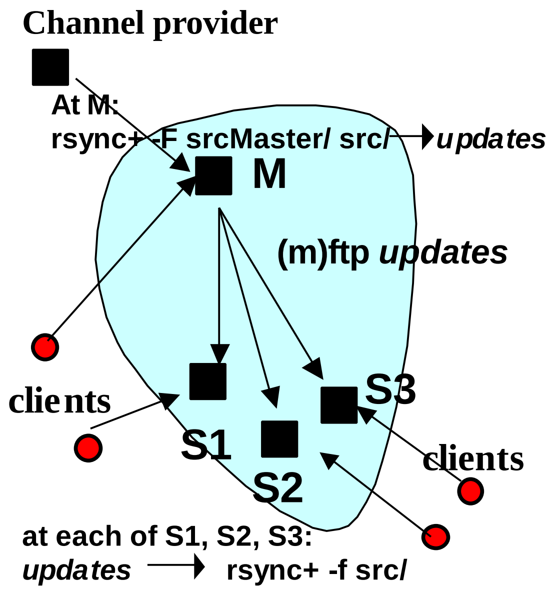


Figure 2.2: Rsync+ Scenario for Content Replication in I2-DSI

3 Performance Analysis

This section focuses on empirical results gathered to characterize the performance of and to provide insight into the scalability of the data mirroring scenario depicted in Figure 2.2. The data represents performance measurements gathered from network and server experiments running on high-performance servers within the I2-DSI WAN testbed. In particular, the results quantify the balance between server processing throughput for file synchronization actions, *server throughput*, (*Step 1* and *Step 3* in discussion above) and the achievable network throughput, *network throughput*, (*Step 2* above) when using parallel TCP connections for data delivery. Understanding this balance is crucial in identifying the current system bottlenecks, in planning future system capacity, in assessing the impact that hardware advances in server and network technologies will have, and in understanding the scale at which deployment of reliable multicast protocols becomes valuable and ultimately critical for the I2-DSI system of replication hosts.

The section has the three subsections: an instrumented mirroring experiment is presented that measures server throughput under rsync+, followed by data from network throughput measurements using the well-known *ttcp* tool [9] and finally, a short discussion on the scalability implications of the server and network measurements.

3.1 Experiment 1: Server Throughput

Methodology: In this section, performance data from a data mirroring experiment utilizing *rsync+* is presented. The experiment created a mirror site of approximately 8 gigabytes (GB) of Linux-related files from a busy WWW archive site (the channel provider in the language of Figure 2.2.), *metalab.unc.edu*. The Linux archives at UNC MetaLab were chosen because the file archive is diverse (e.g., source code, program executables, documentation, html, and graphics files) and active, receiving several significant contributions from developers and users across the open-source Linux community on a typical day. The processing costs associated with managing the mirror's update with rsync+ thus represents changes associated with a real and quite active WWW archive with a modestly large content base.

Seven active subdirectories under the /pub/Linux tree (**X11**, **apps**, **devel**, **docs**, **games**, **system** and **utils)** were chosen for the mirror experiment, and two copies of these directories were placed on an I2-DSI server to initialize the mirror. To match the terminology in Figure 2.2, the mirror machine will be called ***M*** here. It is an IBM enterprise-class storage server in the I2-DSI testbed with 2GB main memory and 72 GB disk. Two copies of the Linux materials referred to as the **srcMaster/** and **src/** copies to indicate their roles. The **srcMaster/** was synchronized with the directories at MetaLab twice daily using an unmodified rsync as the import mechanism. After **srcMaster/** had been updated, *rsync+* **-F** was run on ***M*** to synchronize the **src/** with **srcMaster/.** Each of the seven subdirectories was updated with a separate run of rsync+ **-F**. In addition, for a direct comparison of rsync and rsync+ processing, rsync was run to synchronize separate copies of the **srcMaster/** and **src/** directories on ***M.***

Finally, to measure the processing that would take place for the remote update at each slave site (Step 3 in rsync+ updates), we ran *rsync+* again on ***M***, this time with **-f** option using a pre-updated version of each directory. In this way, each update period in the instrumented mirror generated data on the rsync+ processing times at both the master (the **–F** option) and the slave (**–f** option) side. Mirroring data shown represents approximately one week in duration, i.e., 15 updates separated by 12-hour intervals. Further details on the experimental methodology can be found in [8] in which the mirror was carried out as described here, only using a less powerful mirror host.

Results: Figure 3.1 and Figure 3.2 show the rsync+ processing in the mirror experiment. In Figure 3.1 rsync+ processing times (both **–f** and **–F** modes) for each update of the experimental mirror are plotted as a percentage of the rsync processing time over the same data. The data shows that rsync+ **–F** processing costs are very comparable to those for the original unmodified rsync code. The **–f** mode in rsync+ adds some overhead, on the order of 1.5 to 2 times the unmodified rsync processing.

Figure 3.2 presents data on the measurement of server throughput under the rsync+ **-F** update on **M** to synchronize the **src/** with **srcMaster/**. The dark squares shows the absolute running times of rsync+ **-F** processing in seconds. The updating of Linux files at the original MetaLab archive site was bursty, and several 12-hour periods resulted in no updates, though the rsync-based mirrors must still run and traverse the file trees to determine that no files have been changed.

The two other data sets in Figure 3.2 give two calculations of server throughput (in Mbits/sec), based on the amount of file update information generated. The *unnormalized throughput* is defined as the sum of the size of all files updated in the mirror (added or changed) divided by the running time of rsync+ **-F**. The *normalized throughput* is defined similarly except the fixed overhead for rsync+ to traverse the file tree is subtracted out by subtracting the minimum rsync+ **-F** runtime in the data (13.3 seconds) from each measured rsync+ **-F** runtime value.

Throughput measures under both metrics span a wide range, depending on the size of the mirror update. Over the non-zero entries, the average unnormalized throughput is 2.4 Mbits/s and the average normalized throughput is 11.4 Mbit/s. For data mirroring in I2-DSI then, an important performance element is to aggregate data sets such that non-null updates under rsync+ run over large data sets and thus achieve throughputs in the high end of the measured range.

Rsync+ preserves the differential file update capability in *rsync*. In the experiment reported here, the aggregate amount of data generated by rsync+ **-F** processing represented only a 2% reduction in the total number of bytes for transmission across the network, when compared with summing the bytes in all files added or modified over the lifetime of the mirror. At this level of compression, the rsync feature of differential update (which can be disabled with a flag) seems less than useful. However, the month-long mirror experiment reported in [8] showed that differential file update reduced the total byte count of update information by almost 20%. In the I2-DSI context, the decision to use differential file update will likely be made on a channel-by-channel basis since update behavior will vary between channels.

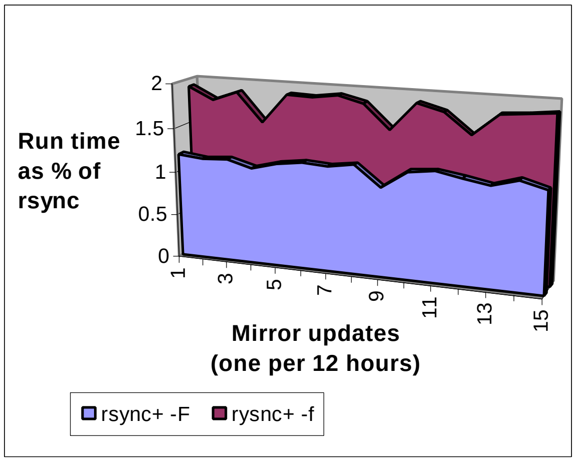
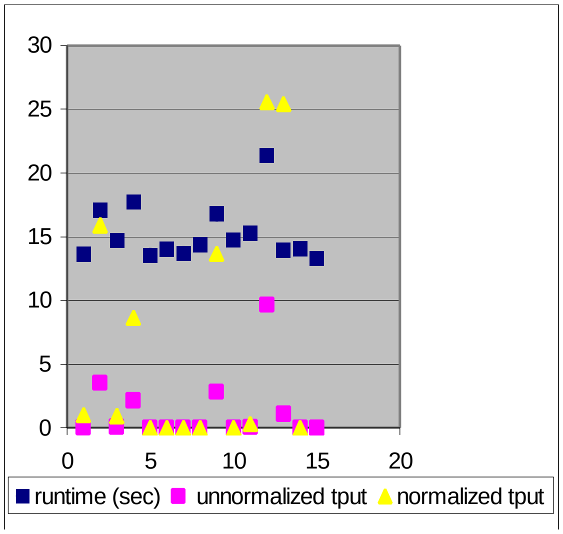


Figure 3.1: Rsync+ processing as a percentage of rsync processing



3.2: Rsync+ -F processing times and data throughput

3.2 Experiment 2: Network Throughput

Data mirroring under the model in Figure 2.2 allows for transmission of the updates file as soon as rsync+ **-F** processing completes. If a reliable multicast transport is available, only a single copy of the file need be transmitted into the network. While this is the most desirable scenario, multicasting is not universally available in WANs today. Instead, in the absence of reliable multicast solutions, the natural strategy is to transmit the file using multiple, concurrent TCP connections, one to each slave site. This section presents experiments measuring the throughputs attainable in current high-speed WANs using concurrent TCP connections. These results benchmark the network throughputs under the most likely scenario for network distribution between the master and slave sites in I2-DSI in the near-term.

Methodology: The experiments presented use the well-known *ttcp* tool for measuring TCP throughput. The critical parameters for the data presented in Figure 3.3 are shown in Table 3.1. The file size of 5.45 MB was selected based on the average size of the updates file generated by rsync+ in the month-long Linux mirror reported in ([8]).

The network path for the measurement shown is between the I2-DSI server on the Internet2 backbone (the same as used in Experiment 1) and a local campus server across a regional network. The minimum speed of all network links between these machines is 100 Mbits/sec, and both are large enterprise-class servers. Measurements using *ttcp* are sensitive to socket buffer settings at the receiver (see [10]). The setting here was determined from *ttcp* runs between the two machines using a range of values. No single value is optimal since network conditions fluctuate, but the 240 KB value is in the range of values offering the highest throughputs achievable.

Parameter	*Values for Experiments in Figure 3.3*
File Size	5.45 MB
Network Path (100 Mbit/s min)	dsi.ncni.net → ils.unc.edu
Concurrent *ttcp* connections	1,2,4,8,16,24,32
Receiver socket buffer size (KB)	240 KB
Buffer Policy	Policy 1: Same-value Policy 2: Shared-by-n

Table 3.1: Parameters for *ttcp* experiment shown in Figure 3.3

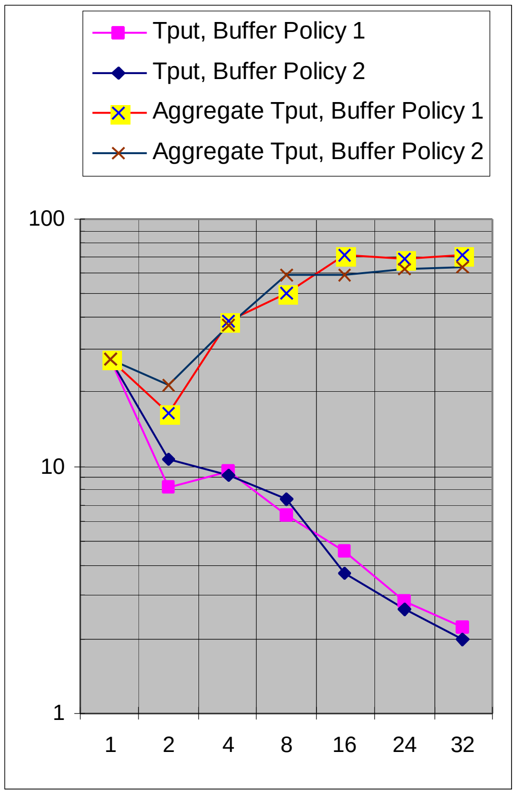


Figure 3.3: Network throughput for concurrent ttcp transmissions

For multiple concurrent *ttcp*s, two different policies for setting the socket buffer size were used: Buffer Policy 1 assigns all concurrent *ttcp*s the same value (e.g., 240 KB), while under Buffer Policy 2 the concurrent *ttcp*s “share” the buffer size (e.g., all use 240 KB divided by the number of concurrent *ttcp*s). Note that, while our experiment uses only two machines and thus simulates a worst-case sharing of the network path, the *ttcp* receivers discarded data upon receiving it (the –s option) so that disk bandwidth at the receiving machine would not be the measured bottleneck. At the transmitting site, data transmissions were taken from a file in order to include the overhead of file I/O, as would be the case under rsync+.

Results: Figure 3.3 shows the average *ttcp* throughput as averaged over a series of runs (6 per data point) taken in off-peak hours. The vertical axis is shown on a logarithmic scale. The curves labeled as *Aggregate Throughput* are calculated by multiplying the average throughput by the number of concurrent sessions. As shown, our data showed single ttcp connections only achieved, on average, 27 Mbit/s, throughput, but when running parallel ttcp transmissions, the I2-DSI server was able to utilize up to 70 Mbit/s of the network path. The use of multiple TCP connections increased the parallelism in the overall network delivery and actually achieved a better utilization of the network, though the reduced throughputs result in greater network transit times for updates.

The data suggests that using TCP to emulate multicasting to at least a few dozen slave sites is a feasible strategy. However, as the number of concurrent sessions grows, the burden of concurrent processing and data movement at the transmitter grows. In our experiments, starting up 48 processes running *ttcp*s resulted in blocked processes, possibly related to the exhaustion of kernel buffers, and hence the ttcp sessions could not operate in parallel. While further tweaking of transmission parameters will likely break down this barrier, such tweaking may be difficult to perform in operational settings. More fundamentally, the limitations of TCP-based delivery are evident as the number of receivers moves beyond the range of a few dozen.

4. Related Work

Rsync is a recent entry in the class of application-level mirroring tools such as rdist, ftp-mirror [11], and mirror [12] that automate the synchronization of file hierarchies on different hosts. Key benefits of application-level mirroring tools such as rsync+ is their high degree of portability, ease and flexibility in modifying and tuning the replication process, and the ability to propagate a solution quickly. Rsync is unique in its support of differential file update using block-level checksumming, and our experimental data suggests that differential file update provides a significant reduction in update bytes using real-world data sets. However, the processing costs of checksumming are high, and further investigation of non-checksumming approaches would be interesting.

Network filesystems represent an alternative approach for file sharing and distribution across distributed platforms. The widely-used commercial solutions, AFS [13] and DFS [14], both implement a limited form of read-only replication that is generally statically configured for load balancing. Other approaches to distributed filesystems such as Coda [15] and Bayou [16] incorporate write-log management in order to maximize file availability for their distributed and/or mobile users. The initial application paradigm in I2-DSI is oriented towards a publisher-subscriber model that makes centralized update a reasonable constraint and therein the lightweight rsync+-based file distribution model of Figure 2.2 feasible for file-oriented channels.

An interesting case of distributing large file sets to widely dispersed hosts is described in [17]. Here a 6-level Network Filesystem (NFS) replication hierarchy for shared read-only data (11 GB, 300,000 files) across a set of very widely distributed hosts is presented as an alternative to hierarchical rdists. Changes at a master site are pulled down the hierarchy by NFS clients polling for changes at their parent. An update file containing commands to describe the new filesystem snapshot at the parent is used to avoid each child performing redundant processing to discover the new snapshot, analogous to our single-shot processing and batch file generation under rsync+ **-F**. This approach is attractive in that it leverages the robustness, error resiliency, and kernel-level performance advantages of the NFS protocol. The key difference between it and rsync+ is that the latter has the ability to exploit and readily incorporate reliable multicasting in the network transport component.

A recent distributed filesystem design, JetFile, [18] does explicitly incorporate reliable multicast transport. JetFile is explicitly designed to be a distributed filesystem for WAN-connected Internet machines, and uses multicast for replica location, synchronization, and management. JetFile exploits multicast for efficient multipoint distribution of immutable file objects. Likewise, other researchers have proposed multicast file distribution in the context of enhancing rdist-like file distribution [19], push caching [20], simple file distribution to WWW servers [21], document delivery applications [22], and other applications. As noted by the JetFile designers, multicasting offers not only advantages in efficient bulk-data distribution, but also flexibility in organizing host communication that avoids the fragility inherent to hierarchical arrangements.

Netlib [23] has been used for some time now in the scientific numerical computing community to share repositories of freely available mathematical software and documentation. Netlib is a set of tools that automates management of the replication procedure across loosely-coupled servers administered by separate organizations. Each server may have a portion of the aggregate repository for which it controls all updates (master) and other portions for which it mirrors remote sites (slave). Netlib sites use background processes to generate an index of each file paired with a checksum over the file's contents. Updating remote slave filesystems uses checksum comparisons to determine which files have been changed, and network transport involves an FTP script generated by a Netlib process.

A key difference between the Netlib scheme and our data mirroring approach is the rsync+ emphasis on timely and synchronous distribution of updates to all participating sites. By contrast, the Netlib project emphasizes automating the replication process in a manageable and robust fashion across loosely cooperating inter-organizational servers. These servers are not simultaneously updated when content changes occur. Moreover, Netlib clients most often access the server within their organization, in contrast with the dynamic binding in I2-DSI between a client and a replication server.

Performance measurements of the vBNS backbone using *ttcp* are reported daily by researchers at [24]. These measurements of individual *ttcp* connections (and others) report generally high (> 100 Mbit/s), though variable, throughput across the vBNS. Factors in our measurements that result in lower throughputs include the use of a production server as the receiving machine, runs over a range of evening times as opposed to the 2 AM tests of the vBNS measurements, and the inclusion of disk I/O at the transmitting node.

5. Conclusions

We have presented a modified open-source data mirroring tool, rsync+, as a basis for constructing scaleable, multiple-site file replication across backbone-speed networks connecting replication hosts. Controlled performance experiments in the I2-DSI testbed with live WWW content were presented to provide insight into the relative match between network and server-side data throughput.

Our performance experiments with rsync+ confirm that server-side processing is a significant cost in the end-to-end update process during file replication. These empirical observations validate the value of the rsync+ design in eliminating redundant server-side processing at a master site for multiple mirrors.

As for scalability of the rsync+ design, our performance numbers show that TCP-only delivery to multiple remote mirrors is feasible for small slave sets (e.g., roughly less than 50). For properly aggregated data sets, the server throughput measurements suggest that rsync+ can update local file hierarchies at 10 Mbit/s or greater. Under these conditions, as the number of slave sites grows, the mirroring mechanism becomes quickly network-bandwidth limited (see Figure 3.3). Thus, for replication host sets beyond a few dozen, reliable multicast solutions will be very valuable in speeding system performance and reducing system load.

References

[1] P. Rodriguez, C. Spanner, and E. Biersack, "Web Caching Architectures: Hierarchical and Distributed Caching," presented at 4th International Web Caching Workshop (WCW 99), San Diego, CA, 1999.

[2] M. Beck, B. J. Dempsey, and T. Moore, "The Internet2 Distributed Storage Infrastructure (I2-DSI) Project,"1999.

[3] M. Beck and T. Moore, "The Internet2 Distributed Storage Infrastructure Project: An Architecture for Internet Content Channels," *Computer Networking and ISDN Systems*, vol. 30, pp. 2141-2148, 1998.

[4] Akamai, Akamai Technologies, Inc., 1999.

[5] Sandpiper, Sandpiper Networks, Inc., 1999.

[6] Y. Chen, J. Edler, A. Goldberg, A. Gottlieb, S. Sobti, and P. Yianilos, "A Prototype Implementation of Archival Intermemory," presented at 4th ACM Conference on Digital Libraries (DL '99), Berkeley, CA, 1999.

[7] Rsync, 1999.

[8] B. J. Dempsey and D. Weiss, "Towards An Efficient, Scalable Replication Mechanism for the I2-DSI Project," School of Information and Library Science, University of North Carolina at Chapel Hill TR-1999-01, April 1999.

[9] Test TCP measurement tool, "TTCP archive site,"1999.

[10] G. Miller, K. Thompson, and R. Wilder, "Performance Measurements on the vBNS," presented at InterOp '98 Engineering Conference, Las Vegas. NV, May 1998.

[11] L. McLoughlin, "ftp-mirror" .

[12] F. Leitner, "Mirror 1.0",1996.

[13] A. Z. Spector and M. L. Kazar, "Wide Area File Service and the AFS Experimental System," *Unix Review*, vol. 7, March 1989.

[14] M. L. Kazar, B. Leverett, O. Anderson, and e. al, "Decorum File System Architectural Overview," presented at 1990 USENIX Conference, Anaheim, CA, 1990.

[15] M. Satyanarayanan, "Scalable, Secure, and Highly Available Distributed File Access," in *IEEE Computer*, vol. 23, 1990.

[16] A. J. Demers, K. Peterson, M. Spreitzer, D. Terry, M. Theimer, and B. Welch, "The Bayou Architecture: Support for Data Sharing among Mobile Users," presented at Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, 1994.

[17] B. Callaghan, "An NFS Replication Hierarchy," presented at W3C Push Technologies Workshop, Peabody, MA, 1997.

[18] B. Gronvall, A. Westerlund, and S. Pink, "The Design of a Multicast-based Distributed File System," presented at Third Symposium on Operating Systems Design and Implementation (OSDI '99), New Orleans, LA,, 1999.

[19] J. Cooperstock and K. Kotsopoulos, "Why use a fishing line when you have a net?: An Adaptive Multicast Data

Distribution Protocol," presented at Usenix '96, 1996.

[20] J. Touch, "The LSAM Proxy Cache: a multicast distributed virtual cache," presented at 3rd International WWW Caching Workshop, Manchester, England, 1998.

[21] J. Donnelley, "WWW Media Distribution via Hopwise Reliable Multicast," presented at 3rd International WWW Conference, Darmstadt, Germany, 1995.

[22] T. Shiroshita, O. Takahashi, and S. Shiokawa, "A large-scale contents publishing architecture based on reliable multicast," presented at 15th Annual Conference on Computer Documentation (SIGDOC '97), 1997.

[23] Netlib, 1999.

[24] vBNS Performance Measurement, vBNS Engineering, 1999.