Towards an Efficient, Scalable Replication Mechanism for the I2-DSI Project

Bert J. Dempsey and Debra Weiss

April 30, 1999
Technical Report TR-1999-01
School of Information and Library Science
University of North Carolina at Chapel Hill
{dempsey, weisd}@ils.unc.edu

Abstract

This paper presents the development of new functionality for the open-source *rsync* utility aimed at producing an efficient, scalable solution for multiple-site file synchronization. The context of our work is the Internet2 Distributed Storage Infrastructure (I2-DSI) project, which is developing a reliable, scalable, high performance storage service infrastructure for advanced applications in research and education. Specifically, the I2-DSI project is working on middleware software to enable the replication of applications across a set of geographically distributed hosts. This paper presents a new mechanism for replicating filesystems, *rsync+*, which is a modification of an open-source *rsync* file synchronization utility. Using *rsync+* for file updates, a flexible, powerful replication mechanism can be developed for publishing source objects into the I2-DSI replication service, and the approach enables scalable network distribution through multicast-based solutions. The paper presents the technical details behind the *rsync+* tool, its use as a replication solution within I2-DSI, and performance results from a large-scale (multi-gigabyte) WWW mirroring experiment using *rsync+* that demonstrate correct operation and efficiency gains with actual data from an active WWW document archive.

Keywords: Caching and Replication, Multicast Protocols, Distributed Storage

1 Introduction

The Internet2 Distributed Storage Infrastructure (I2-DSI) project seeks to develop a reliable, scalable, high performance storage service infrastructure for advanced applications in research and education, specifically within the community of 140 U.S. research universities supporting the Internet2 project. I2-DSI represents a new framework for integrating storage into the network, but the project focuses on advances that are readily achievable through innovative use of the storage and networking technology available today.

The work in this paper addresses the need for efficient, scalable replication mechanisms within the context of replicating content and services across hosts in a wide-area network, e.g., to I2-DSI servers located at Internet2 sites and cooperating institutions. Here we motivate and overview the I2-DSI project and its goals before describing our replication approach in the rest of the paper.

1.1 I2-DSI Project

As Internet connectivity and network bandwidth continue to increase, the potential grows for large-scale applications involving widely dispersed user communities. In practice, performance challenges in delivering rich digital media and large data sets are limiting the feasibility and utility of Internet delivery for many research and education projects. Simply overprovisioning the network is not economically feasible as wide-area networks (WANs) must be aggressively shared, leading to transient periods of congestion that result in delay and packet loss. Traffic statistics from major exchange points, for example, show high levels of traffic, even on holidays and "off-hours" [1].

A long-term solution is the development of quality-of-service networking on an end-to-end basis. QOS networking refers to network-level technology that will enable service guarantees, e.g., bounds on packet latency and minimum throughputs, from the network to be made to individual or aggregate application data flows. Services with statistical service guarantees, as opposed to deterministic guarantees, are being developed in the Internet community [2], and an increasing number of network technologies, e.g., ATM networks, offer some inherent support for QOS service guarantees. Yet, true end-to-end service guarantees require unification of all intermediate guarantees under one scheme, which is especially challenging, and even limited forms of QOS networking are not widely available at this time. Reliance on QOS networking to ensure high-performance access to Internet-hosted services is not possible for now, and uncertain for the foreseeable future.

The Internet2 Distributed Storage Infrastructure (I2-DSI) project takes an alternative approach to improving client-server interactions in networked applications. I2-DSI focuses on creating a scalable, heterogeneous middleware architecture that is a platform for replicating services. First, I2-DSI enables the replication of application services by replicating software servers (e.g., a WWW server) and source objects (e.g., source files) across a set of dedicated, geographically distributed hosts in the wide-area network. Then, on the client side, mechanisms are deployed that allow clients to access content hosted by I2-DSI hosts transparently. That is, without knowledge of the replication scheme, clients will use a global name (e.g., a URL or URN) to locate application content and be directed by network mechanisms to a "good" replica server where "good" means a server that has the requested content and is as "local" as possible for performance reasons.

Content in the I2-DSI framework is grouped into *content channels.* A content channel is defined to be [3] "a collection of content, which can be transparently delivered to end user communities at a chosen cost/performance point through a flexible policy-based application of resources." The replication framework in I2-DSI provides the flexibility of determining the resources that will be devoted to improving access to a content channel. Trivially, the number of replica hosts on which the channel will be replicated is one such control point. For transparent resolution, content channels will be associated with Internet domain names so that the ubiquitous distributed database of the Domain Name Service (DNS) can hold channel information. DNS allows multiple IP addresses to be associated with a single domain name, e.g., the IP addresses of the replica hosts on which a channel is replicated. Special DNS resolution software can then dynamically determine, using network metrics, the IP address of the closest replica host during DNS name resolution and return that IP address to the client software in the name resolution interface [4]. Because the DNS resolution interface is unchanged in this scenario, we call this transparent resolution for the client.

I2-DSI emphasizes localized access because the local network, either an extended campus local area network (LAN) or even a regional network, will be inherently fast and reliable. Most LANs today are utilized well below their capacity and generally provide high-bandwidth, low-latency communication. Moreover, for applications where true QOS guarantees are necessary (e.g., applications with real-time interaction requirements), providing QOS networking over a small part of the network is inherently more manageable and easier to deploy than providing end-to-end QOS across WANs spanning multiple administrative domains.

I2-DSI also seeks to leverage powerful technology trends in mass storage and high-speed networks. Storage costs have been dropping by half each year in recent years, and single systems with terabytes of storage are now in place. At the same time, the raw transmission speeds of the network have reached gigabits per second in operational networks with dramatic possibilities for further gains as wave-division multiplexing becomes a commercial reality. These fundamental trends bode well for content replication schemes such as that envisioned by I2-DSI.

1.2 I2-DSI Application Scenarios

The target application set for I2-DSI is initially research and education applications since the project has grown out of the Internet2 project. Directed by the non-profit University Corporation for Advanced Internet Development (UCAID), the Internet2 project (http://www.internet2.edu) seeks to accelerate the development of next-generation Internet technology through a powerful partnership of the academic community with industry and government. I2-DSI represents one of a higher-layer systems effort within Internet2 aimed at developing next-generation network services. The envisioned DSI project will provide the software infrastructure and seminal research to develop the network-based middleware necessary to support replicated services.

While the replication middleware of I2-DSI does not require the high-bandwidth and advanced network features of the Internet2 network in any fundamental way, the Internet2 context provides both a powerful, state-of-the-art networking environment in which to develop technical solutions (e.g., we intend to exploit IP multicasting, where available). The Internet2 community also provides access to innovative researchers and systems administrators who are willing to work with I2-DSI and who can enable large-scale experimentation with new ideas. In the application context, Internet2 is focused on the researchers with next-generation applications, and these application groups are the ones that I2-DSI is reaching out to as early adopters and collaborators.

In March 1999 at the University of North Carolina, application groups were invited to explore requirements and possibilities within the I2-DSI framework in a one-day workshop [5]. Groups participating covered a range of application types, including:

1. Digital libraries of streaming media and large images such as
    - University of Indiana's *Variations* that provides access to over 5000 titles of near CD-quality digital audio,
    - the California State University Image Consortium that has digitized and cataloged over 12,000 images to-date for art history education, and
    - a medical image database at Vanderbilt University that uses multiple-resolution imagery and software-based zooming to provide very high-quality access to a research project using images produced by CT, MR, and other modalities.
2. Document repositories, like the Internet standards collection at *Normos.org* and the Linux Archives at http://metalab.unc.edu/
3. On-line publishing services such as the Columbia EARTHSCAPE project.
4. New applications in scientific collaboration models such as
    - the GIOD project to address the data storage and accesses problems posed by the next generation of particle collider experiments which will start at CERN in 2005, and
    - the University of North Carolina distributed, virtual laboratory project that is advancing nanotechnology through development of virtual reality interfaces to scientific instruments.

As seen in this list of potential I2-DSI applications, the range of application-driven requirements for effective replication will be quite large. No one replication or resolution mechanism can serve the needs of all possible content channels, and the I2-DSI development plan explicitly embraces multiple replication and resolution solutions, as needed, in coordination with an evolving understanding of the taxonomy of applications and their common needs.

The work described in this paper then is one approach to replication. It will serve the needs of any application with file-based source objects that must be updated from time to time. Our replication solution also provides an open-source interface that synchronizes source objects at a master site controlled by the content channel provider with a master I2-DSI site for the channel. Thus, the channel provider maintains full control over its source objects and has a completely automated solution for object update. Furthermore, for synchronization between I2-DSI sites, our solution offers an efficient, scalable solution, and in particular the ability to exploit network-level multicast.

The rest of the paper is organized as follows. Section 2 lays out the *rsync+* replication tool we have developed and its envisioned use in I2-DSI. Section 3 gives the technical details of the *rsync+* implementation. This section is intended to serve as a basis for users who want to know how the tool works and for application developers or implementers who want the gory details. Section 4 presents a mirroring experiment done using *rsync+*. This experiment was designed to validate that our code modifications to *rsync* work correctly in operational use and to provide data on the performance of *rsync+*. Section 5 summarizes our conclusions and outlines future work plans.

2 Rsync+ Motivation

This section describes the *rsync+* tool we have developed, a scenario for its use as a replication mechanism in I2-DSI, and the advantages and limitations of our approach. *Rsync+* is the name we use to denote our modifications to an excellent open-source tool for file mirroring, *rsync* [6]. In this section we motivate these modifications in terms of developing a scalable, efficient tool for publishing source objects in an I2-DSI content channel. Publishing here means moving new or modified source objects from a channel provider's site to the set of I2-DSI replication hosts on which the channel resides.

Our focus in this discussion is on the *rsync+* mechanism as a replication transport, that is, an efficient file update and file transfer protocol for synchronizing a master filesystem with a set of remote filesystems. The *rsync+* mechanism must have a replication framework built over it to create a complete file replication solution that addresses replica consistency, atomicity of updates, and other higher-layer issues. We believe *rsync+* provides an efficient, flexible data transport solution to which higher-layer protocols can be added, as needed, to create replication solutions appropriate for different classes of applications. As a concrete starting point, our canonical application for *rsync+* is I2-DSI replicable WWW service, that is, replication of WWW-hosted document archives held in hierarchical filesystems.

For replicating file-oriented channels, one possible approach is to rely on the replication and concurrency mechanisms embedded in a distributed filesystem (DFS), e.g., DFS solutions based on the standards of Distributed Computing Environment (DCE) or the Andrew filesystem (AFS). The distributed file services provided by these solutions are quite powerful and of interest to the I2-DSI project[1], but these solutions also have significant costs with regard to deployment effort, portability, administration, and configuration overhead. Because these costs limit flexibility and serve as barriers to rapid

[1] An interesting performance question, for example, is how well DFS/AFS solutions, which have generally been deployed inside enterprise-wide networks, will perform when using wide-area networks where packet loss rates are potentially greater and latency higher than traditional deployment environments.

adoption, simpler solutions such as weak-consistency replication based on *rsync+* have value in the spectrum of possible replication solutions.

2.1 File Mirroring Tools

Tools for site-to-site file synchronization are widely used among current Internet sites that replicate files for FTP or HTTP, so-called *mirror sites*, and our initial approach was to survey and leverage state-of-the-art mechanisms from these tools. Mirroring tools go back to early Unix utilities such as *rcp* and *rdist* for remote copying and automatic file updating between distributed Unix file systems. Solutions in use at FTP and HTTP archives today are generally front-end scripts to FTP such as mirror [7] and ftp-mirror [8]. They use filesystem commands (e.g., ls -lR) to compare the state of the source and target filesystems and then build a list of files that must be moved by FTP from the source to the target. A range of options allow the update process to be tuned to local site needs, e.g., exclusion of files with certain extensions, use of compression, logging, and so forth. The wide use of such tools attested to their effectiveness in automating the process of synchronizing file archives at replicated servers.

A recent entry among file synchronization tools is an open-source called *rsync* [9]. *Rsync* has similar goals and functionality to the tools above. It has many options for flexible configuration, has been ported to a number of platforms, and is widely used. *Rsync* has, however, a distinctive feature that separates it from FTP-based mirroring tools: it implements a novel file update algorithm that computes checksums over blocks of information in changed files and transmits over the network only those data blocks necessary to update the target file. This approach trades off higher processing costs (checksums over changed files) for reduced network bandwidth. Section 3 provides details on the exact processing performed by *rsync* during file update and transmission.

A different approach to the problem of synchronizing files is taken in the NetLib project. Netlib is a very successful, long-running program used in the scientific computing community to share repositories of freely available mathematical software [10]. In this system, a set of distributed servers cooperate to share software repositories. 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. Network transport is then accomplished with an FTP script generated by a Netlib process.

Netlib relies on checksum comparisons, not file modification information, to determine the list of changed files. By computing checksums in background processes, file checksums are readily available for this purpose. Besides ensuring detection of subtle changes in a file's content, the Netlib authors report a number of advantages to this approach, including, for example, (1) flexibility in dealing with access control and firewalls, (2) avoidance of idiosyncrasies and limitations in filesystem utilities related to older systems, and (3) flexibility in performing compute-intensive comparisons of compressed files. As emphasized in this last point, the Netlib approach of relying on information gathered during background or batch computation (e.g., file checksums) is in contrast to the general-purpose mirroring tools (e.g., ftp-mirror and rsync) where file comparisons take place while connected to an network session, potentially tying up network resources for a long time [11].

Our replication solution, *rsync+*, follows from a recognition that the flexibility of the asynchronous update model used in Netlib could be achieved in combination with the fine-grained use of checksums in rsync by adding a "batch-mode" operation to the *rsync* tool. The next sections illustrate the advantages of *rsync+*.

2.2 Rsync as a Replication Transport

To motivate our *rsync* modification, we first consider the use of an unmodified *rsync* (as a state-of-the-art Internet file-mirroring tool) for replication of file-oriented channels in the I2-DSI context. Figure 2.1 shows how *rsync* in its current form could provide a channel publishing interface into the I2-DSI core. A channel provider need only set up *rsync* and configure it to periodically update the channel. In the figure, new or modified source objects from the directory **src/** are synchronized with the corresponding directory on a master site (M), one of the set of I2-DSI replication hosts (shown as shaded area).

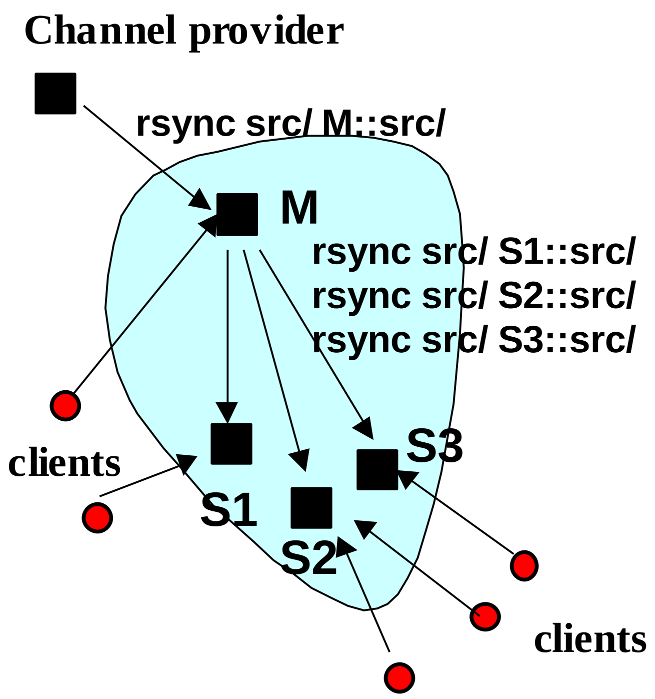

Figure 2.1: Using Rsync as a Replication Mechanism in I2-DSI

The master site then performs an *rsync* session with each of the remote I2-DSI hosts on which this channel is replicated (S1, S2, and S3 in the figure). As shown, clients for this I2-DSI channel access the replication hosts closest to them using transparent resolution mechanisms.

Note that, as with the Netlib model, a channel could be an aggregation of separate file collections, each controlled by a different site. In that scenario, Figure 2.1 would have multiple "channel provider" sites using *rsync* to synchronize portions of the channel with one or more master I2-DSI site(s).

Rsync+ is a response to two factors in the scenario in Figure 2.1 that result in inefficiency and limit scalability:

- 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 linearly with the number of remote servers being updated.

- M transmits identical data streams over the network in each of the three master-slave *rsync* sessions (S1, S2, S3).

2.3 Rsync+ as a Replication Transport

By capturing into a local file the information generated during an *rsync* file synchronization session (i.e., file status, checksums, and data blocks*), rsync+* enables an efficient, scalable scenario for multiple-site file synchronization, as shown in Figure 2.2.

During the initial synchronization with the channel provider, the file of update information (labeled **updates** in the figure) is captured using the *rsync+* option (**-F**). This file can then be transmitted using three separate FTP connections or, more powerfully, a reliable multicast program. The remote servers (S1, S2, and S3) use an option in *rsync+* (**-f**) to update their **src/** directories through local processing on the **updates** file.

The solution eliminates redundant processing at M while also allowing, where operationally feasible, the network transmission to be done with a reliable multicast protocol. Multicast refers to network transmission in which the data source transmits a single copy of the data and network elements create data copies as needed to deliver the transmitted information to a set of receivers. As compared with multiple one-to-one transmissions, multicasting offers the efficiency gains of using less network bandwidth, CPU processing, and other computing resources along with the delivery speedups of transmitting a single copy of the data. These efficiency gains within the DSI model will grow as the product of the size of the source objects, the frequency with which they are updated, and the number of DSI servers to which the objects must be delivered. Where delivery latency is crucial, multicasting may enable scenarios that can not be achieved with multiple one-to-one connections.

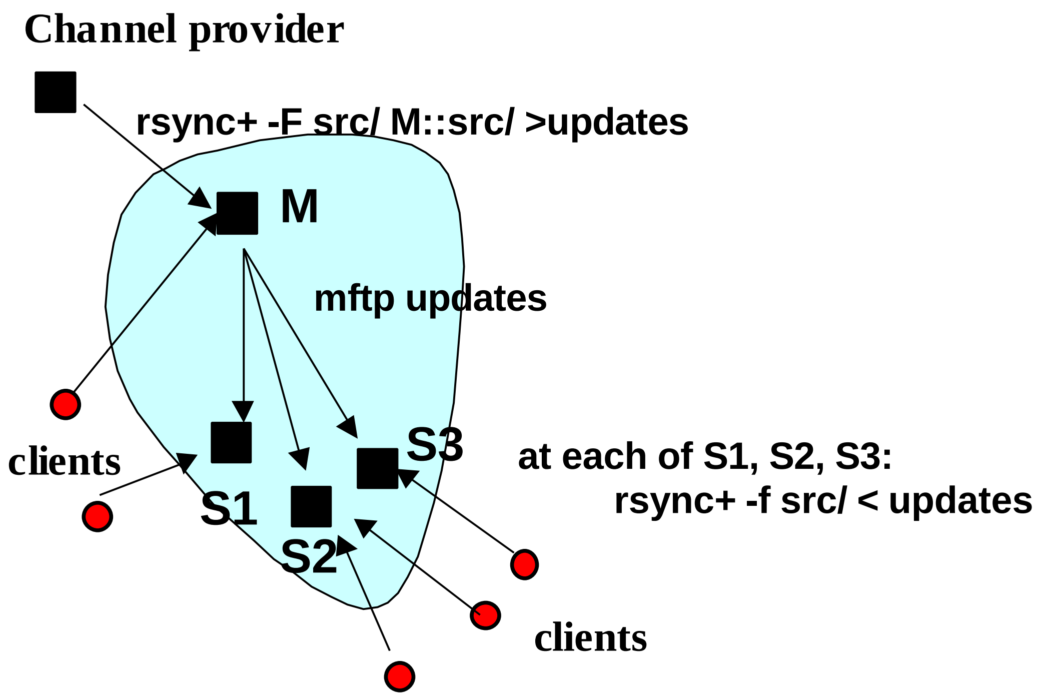

Figure 2.2: Using Rsync+ as a Replication Mechanism in I2-DSI

While IP-level multicasting is not available in all portions of the network, many service providers are committed to it, including the Internet2 community [12], and application-layer tunneling strategies offer similar performance gains in the short term [13]. Higher-layer reliable multicast protocols are now available from both commercial and research sources, e.g., [18]. Within the I2-DSI project, we will experiment with Starburst Communications Multicast File Transfer Protocol (MFTP^{TM}). MFTP^{TM} is an open protocol [14] and one of the protocols under consideration for standardization in the active research now taking place in an IRTF Working Group on Reliable Multicast.

3 Rsync+ Implementation

For I2-DSI, we are leveraging the power and flexibility of *rsync*, a public-domain mirror tool, to create a lightweight replication mechanism for content channels. *Rsync* is widely used on a variety of platforms and provides an efficient method for remote filesystem synchronization. In its current form, this synchronization occurs in a single network session between two sites. Provisioning for the network performance gains made possible by multicast protocols, we created batch features for *rsync* and call our new derivative *rsync+*. The new batch-mode operation of *rsync+* trades off local processing for efficient use of network bandwidth regardless of the network transport method used, and offers a scalable solution for multiple-site replication. This section provides an overview of *rsync*, discusses the batch options we added to create *rsync+*, and shows how the new features can be used. Section 4 addresses performance aspects of *rsync+* in relation to results obtained in a local mirror experiment with a multi-gigabyte Linux archive.

3.1 Rsync

rsync is similar to the Unix remote file copy program, *rcp*, but has a rich set of additional features which provide considerable execution flexibility. The primary feature is an internal checksum-search algorithm with can greatly speedup remote file synchronization by computing the differences between two files and sending just the differences across the network link. While it works best when the two files are similar, the algorithm has been proven to be reasonably efficient even when the files are quite different [15]. Complete documentation describing *rsync* and its use can be found at http://samba.anu.edu.au/rsync/.

rsync operates between a *master* site that maintains the master copy of a shared archive, and a *slave* site that stores a replica copy. Like *rcp*, it requires that source and destination files be specified as command line arguments to identify the part of the archive that is to be synchronized, and accepts both local and remote filenames. Additionally, other command line arguments make it possible to choose various mirroring options. For example, some of the available options include:

- preserving file permissions and modtimes
- changing checksum blocking size
- recursing into subdirectories
- designating file exclusions

Figure 3.1 presents a high-level view of how *rsync* works to synchronize files in directory **mirrorfoo/** at the slave site with files in **foo/** at the master. All communication and data exchange between master and slave occur via open network pipes connecting the two sites. From the command line arguments supplied, a process at the master site determines which master files are to be copied and builds a list of file information (e.g. file name, directory name, size, modtime) to send to the slave (Fig. 3.1, circles 1 and 2). The slave receives this information and by comparing file sizes and modtimes, determines local candidates for update. When a potential file change is detected, the slave splits its local copy into a series of non-overlapping data blocks, calculates a weak “rolling” 32-bit checksum and a strong 128-bit checksum for each block, and sends the checksums back to the master (Fig. 3.1, circles 3 and 4).

The master then initiates its own checksum process as it searches for data blocks in **foo/** that have weak and strong checksums that match those received from the slave. A special property of the rolling checksum enables the process to proceed very quickly in a single pass of a file. When the checksum algorithm shows file discrepancies between the two sites, the master will send instructions and data block differences so the slave can reconstruct the master copy locally (Fig. 3.1, circles 5, 6 and 7). To minimize latency, the slave process that generates and sends checksums to the master runs independently of the process that receives file delta information from the master and reconstructs replica copies. After files are reconstructed at the slave site, **mirrorfoo/** and **foo/** will be in sync.

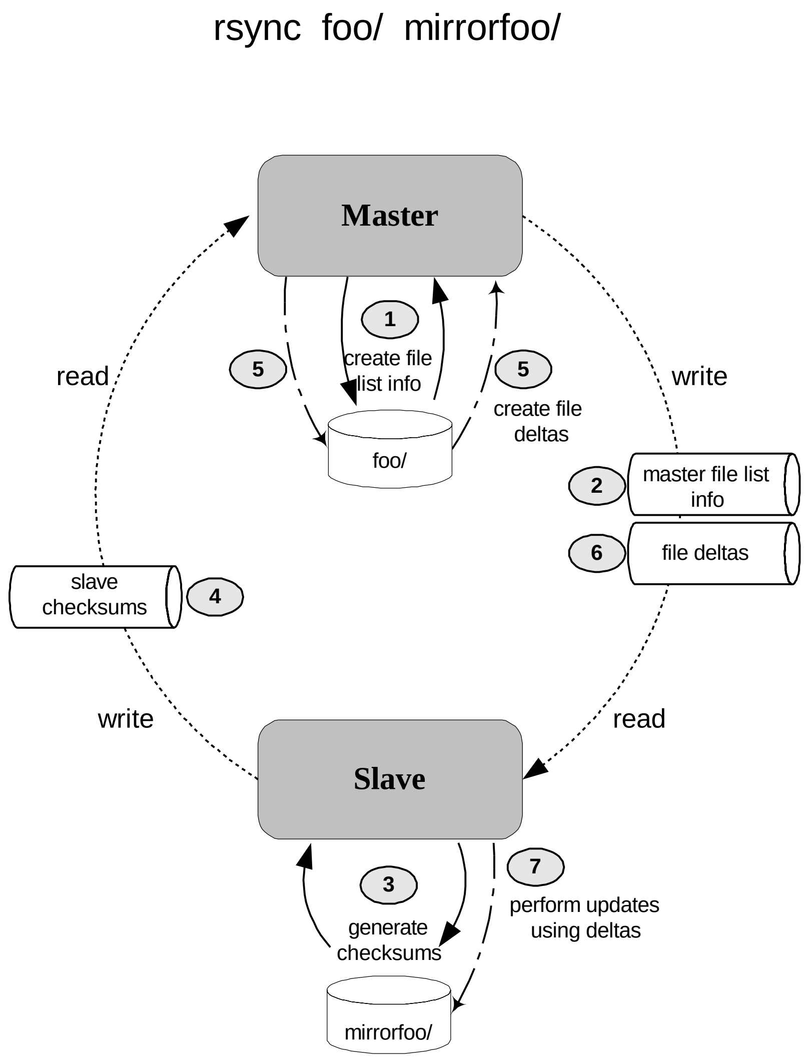

Figure 3.1: rsync

3.2 Rsync+

As described in Section 2, *rsync+* adds a new batch feature to rsync which makes it possible to decouple the file update process at the slave (Fig. 3.1, circle 7) and run it locally at remote sites without consuming network bandwidth. It takes advantage of *rsync*'s speedy checksum-search algorithm and the many mirroring options provided, and enables a mode of synchronization where highly efficient multicast protocols can be used for data delivery. It is an attractive solution for the content channel model of I2-DSI where potentially very large data sets will be replicated at multiple servers throughout the Internet.

Two new command-line options work in tandem to provide the new batch capability of *rsync+*. A "write-batch" option, **-F**, runs at the master site and behaves just like *rsync*, with functionality added to capture batch information that remote slaves will subsequently need to perform local batch updates. This information includes command-line arguments that designate mirroring options requested, master copy file information, slave checksums, and the actual file deltas which will be used to reconstruct files at the slave. It is important to note, as with normal *rsync* operation, that running *rsync+* with the write-batch option will establish a point-to-point session and synchronize one master-slave pair over the open network link. However, by running *rsync+* in write-batch mode between the master and closest slave in the mirror group, network overhead can be minimized or even eliminated altogether if one slave replica is setup as a local product backup of the master archive. Once the write-batch option is used to capture batch information at the master site, the new "read-batch" option of *rsync+*, **-f**, can be used at multiple remote sites to perform batch synchronization.

Figure 3.2 shows *rsync+* behavior using the write-batch, **–F**, option. Darkened circles on the diagram (Fig. 3.2, circles1a, 2a, 4a and 6a) identify the new functionality added to *rsync*. At opportune stages in the synchronization process, information described earlier is written to four batch files:

- a batch argv file with command-line arguments (Fig. 3.2, circle 1a)
- a file list containing information about master files being synchronized (Fig. 3.2, circle 2a)
- a file of checksums computed over slave files being synchronized (Fig. 3.2, circle 4a),
- a file containing the actual data blocks that remote slaves will use to bring their mirrors in sync (Fig. 3.2, circle 6a).

After *rsync+* completes, these four files can then be bundled into one compressed *rsync+* batch file and shipped off to others in the mirror group (Fig. 3.2, circle 8).

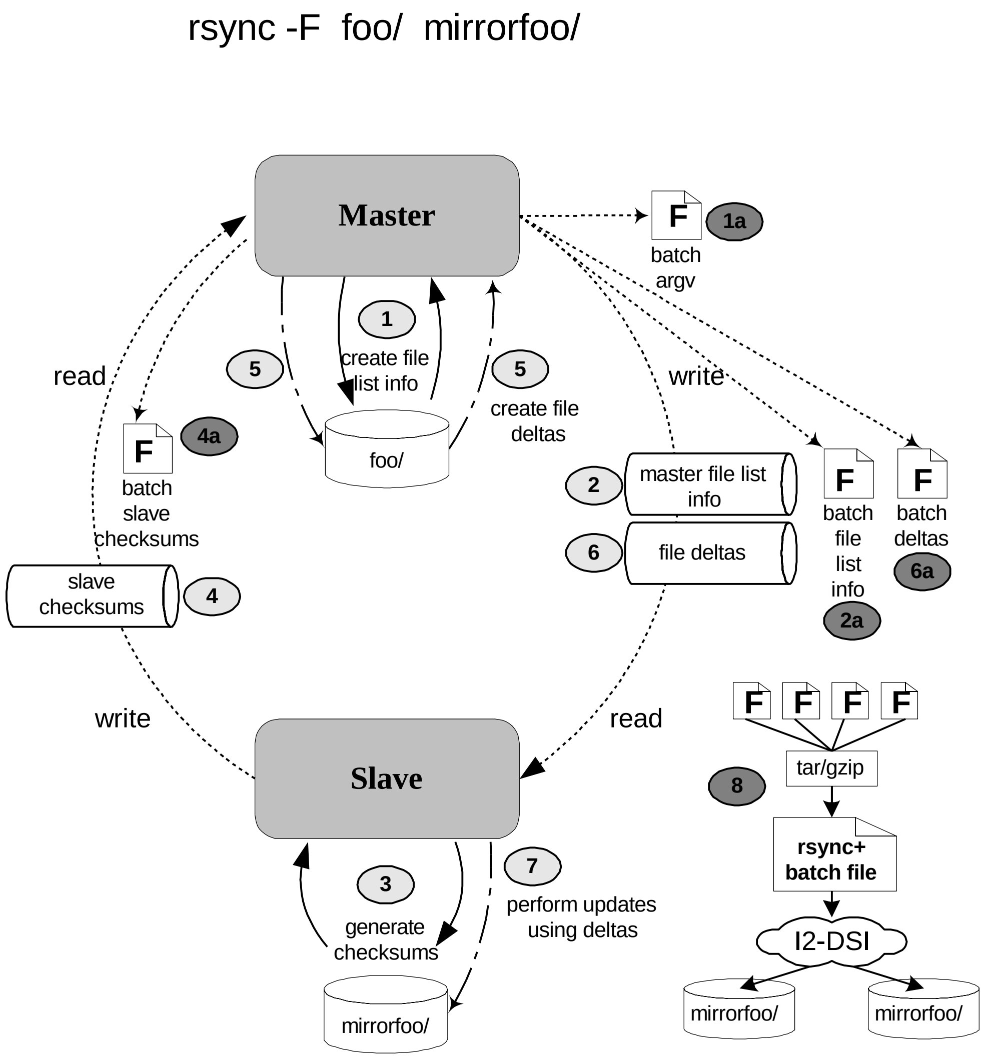

Figure 3.2: rsync+ using -F option

Figure 3.3 shows *rsync+* behavior using the read-batch, **–f**, option at a remote site. Requisite to the run, the *rsync+* batch file received from the master is uncompressed and unbundled into the batch files created from the write-batch, **-F**, run (Fig. 3.3, circle 1). Running locally in read-batch mode *rsync+* will read the batch information (Fig. 3, circles 2, 3, 5 and 6) and use it with locally computed checksums (Fig. 3.3, circle 4) to synchronize slave files. By receiving the slave checksums calculated when *rsync+* ran in write-batch mode at the master site, a remote site can perform a safety check to ensure that local files have not become out of sync with replica copies at other slave sites. Assuming no such discrepancies, *rsync+* will proceed to reconstruct the file using the batch difference information it received from the master (Fig. 3.3, circle 7).

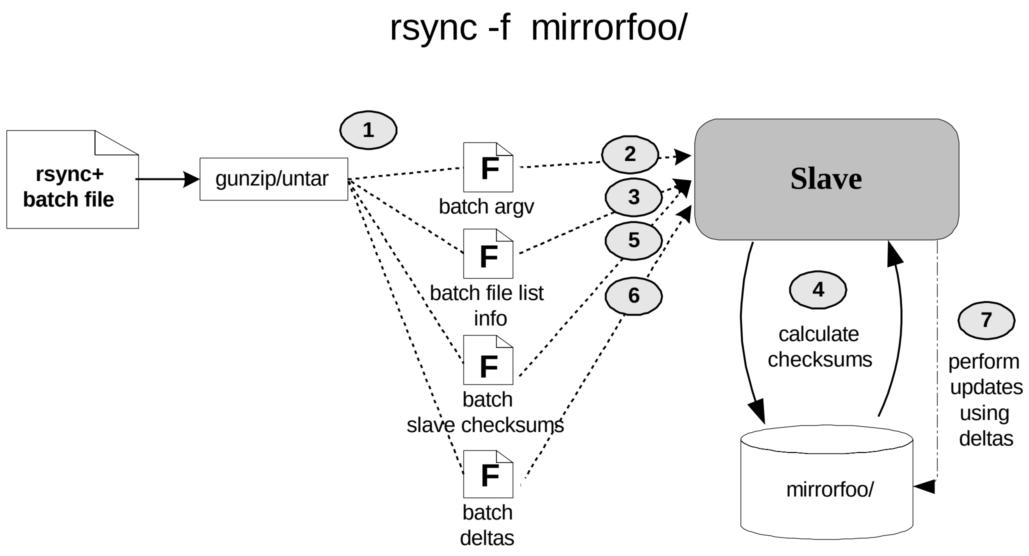

Figure 3.3: rsync+ using -f option

3.2.1 Rsync+ Implementation Details

System Information

Rsync+ was created from *rsync* version 2.1.1 for Solaris and run on high-performance Sun UltraSparc machines using rsh.

Program Code

The main design goal was to disrupt as little of the existing *rsync* code as possible, avoiding any changes that would break *rsync*'s internal data structures. In the final version, a new program module, **batch.c**, was created to include all functions related to the new batch processing features, and the following programs were modified: **options.c**, **main.c**, **flist.c**, **compat.c**, **sender.c**, **token.c**, **util.c**, and **match.c**. Approximately 550 lines of code were written in **batch.c**, and about 200 total new lines of code were added to the other programs. Most of the new program statements added to the other modules were short **if** blocks that check for the read-batch and write-batch options, and call appropriate functions or alter existing program flow. For example, during read-batch processing (**-f** option), **if read-batch** statements recircuit existing *rsync* code to "snip" network pipes.

In our implementation of *rsync+*, we opted to save batch information in four files. This made it possible to keep much of the current *rsync* code in place, and to integrate batch file read/writes at places in the code where pipe I/O occurred. To make it easier to associate the four files created in any one run, and to enable multiple runs to process sets of files in the correct chronological order, *rsync+* creates the four files with a common timestamp extension. Along with command-line arguments, the timestamp is also saved in the argv batch file which *rsync+* makes executable (with **–F** changed to **–f**) to enable automated read-batch processing at remote sites..

Compiling and Running rsync+

rsync+ is compiled like *rsync* using the same makefile with the addition of **batch.c**(**o**). It is executed like *rsync* with the addition of the **–F** or **–f** option. Since it is mainly comprised of *rsync* code, it should port to every platform that supports *rsync*.

Sample Implementation Scripts

Write-Batch (-F option)

```
#!/bin/ksh
# Very basic rsync+ -F script with no exception handling

# Run rsync+
rsync -F -r -t -l -H -S --delete /export/home/foo slave:/export/home

# Get the file extension of the argv batch file.  Use it to get the
# other batch files which go with it. Revise if more than one
# set of batch files in current directory
argv_filename=`ls rsync_argvs.*`
file_ext=`basename "${argv_filename}"|awk -F\. '{print $2}'`

# Tar up the batch files with the same file extension, then compress
tar -cf rsync_tar.$file_ext *.$file_ext
gzip rsync_tar.$file_ext
```

Read-Batch Script (-f option)

```
#!/bin/ksh
# Very basic rsync+ -f script with no exception handling

# Unzip the rsync_tar.gz file
gzip -d rsync_tar.*.gz

# Get the file extension of the unzipped rsync_tar file.
# Revise if more than one tar file in current directory
rsync_tar_file=`ls rsync_tar.*`
file_ext=`basename "${rsync_tar_file}"|awk -F\. '{print $2}'`

# Untar the rsync_tar file
tar -xf $rsync_tar_file

# Run rsync+ using rsync_argvs file
rsync_argvs.$file_ext
```

3.2.2 Future Work

For our purposes, we tested *rsync+* with the following set of *rsync* options:

-r	recurse into subdirectories
-t	reserve times
-l	preserve soft links
-H	preserve hard links
-p	preserve permissions
--delete	delete files that don't exist on the sending side
--exclude=PATTERN	exclude files matching PATTERN

Further testing needs to be done using other *rsync* options. Additionally, when making code changes, we focused on *rsync* "push" behavior where data is mirrored from a local master to a remote or local slave. That is, the files created as shown in Figure 3.2 are created on the source machine (master) doing a "push" to a slave machine. We are investigating for flexibility in use enabling creation of *rsync+* output files at the destination (slave) machines.

rsync+'s full dependence on *rsync* makes it unfriendly to new *rsync* versions. If *rsync* changes, we will have to add our code to each new version until it can be integrated into the public *rsync* source code domain. The latter is something we hope to initiate in the near future.

Finally, stronger error-handling and reporting mechanisms need to be developed to make *rsync+* fully robust.

4 Local Mirror Experiment using Rsync+

In this section, we present performance data from a long-duration data mirroring experiment utilizing *rsync+*. The experiment was to create a local (i.e., on the same local area network) mirror site of 8GB of Linux-related files from a busy WWW archive site, *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 30-50 contributions per day from developers and users across the open-source Linux community.

The goals of the experiment then were to confirm correct operation of the *rsync+* code with a large set of WWW content and also to gather performance data for preliminary evaluation of rsync+. Here, we describe our methodology, present our results, and summarize our findings.

4.1 Methodology

The Linux repository at UNC MetaLab is served with other collections from a single high-performance (4 processors, 1 GB memory) Sun UltraSparc Enterprise 400 machine, *metalab.unc.edu,* and is available for download at http://metalab.unc.edu/pub/Linux. We mirrored an 8-gigabyte subset of the archive (approximately one-half of total collection) to a dedicated Sun Sparc workstation co-located on the same campus LAN. The network path between the two machines includes both 100-Mbit/s and 10-Mbit/s links through a switched LAN and passes through one large Cisco router.

Seven active subdirectories under the /pub/Linux tree (**X11**, **apps**, **devel**, **docs**, **games**, **system** and **utils)** were chosen for the mirror experiment. A script on *iris.unc.edu* was set to run twice a day, at 8:30 a.m. and 8:30 p.m., to synchronize each of the local directories with the source directories on *metalab.unc.edu***.** Each script run consisted of three phases:

(1) Each directory on *iris.unc.edu* was synchronized with the master on *metalab.unc.edu* using normal (unmodified) *rsync*, one *rsync* run per directory**.**

(2) Then, locally on *iris.unc.edu*, we ran *rsync+* in write-batch mode (-F option).

(3) Finally, to simulate remote updates, we ran *rsync+* again on *iris.unc.edu*, this time in read-batch mode (-f option), using a pre-updated version of each directory, that is, a directory image identical to that on *iris.unc.edu* before step (1).

4.2 Results

Rsync+ preserves the differential file update capability in *rsync*. In Figure 4.1, we show the total number of bytes for all files added or modified in each 12-hour interval for the seven directories on the *metalab.unc.edu* site (labeled *source files*, light bars) and the total data bytes that the *rsync+* algorithm will move over the network to perform these updates (labeled *rsync+ files*, dark bars). The data shown covers a three-week period from March 17th to April 6th, 1999. Note that the top graph is plotted with a log-scale y-axis whereas the two lower graphs have a linear y-axis.

As shown in the figure, the differential file update algorithm has a modest effect for some update periods, those in which updates are largely new files added to the archive. In other periods, however, where changes to the archive include file modifications as well as additions, the charts show that differential batch updating is quite significant. Over all updates, the sum of *rsync+ files* represents 81% of the sum of *source files*. Also, in general, the figure indicates great variation in the aggregate size of the source files changed over any 12-hour period, including periods of no change at all

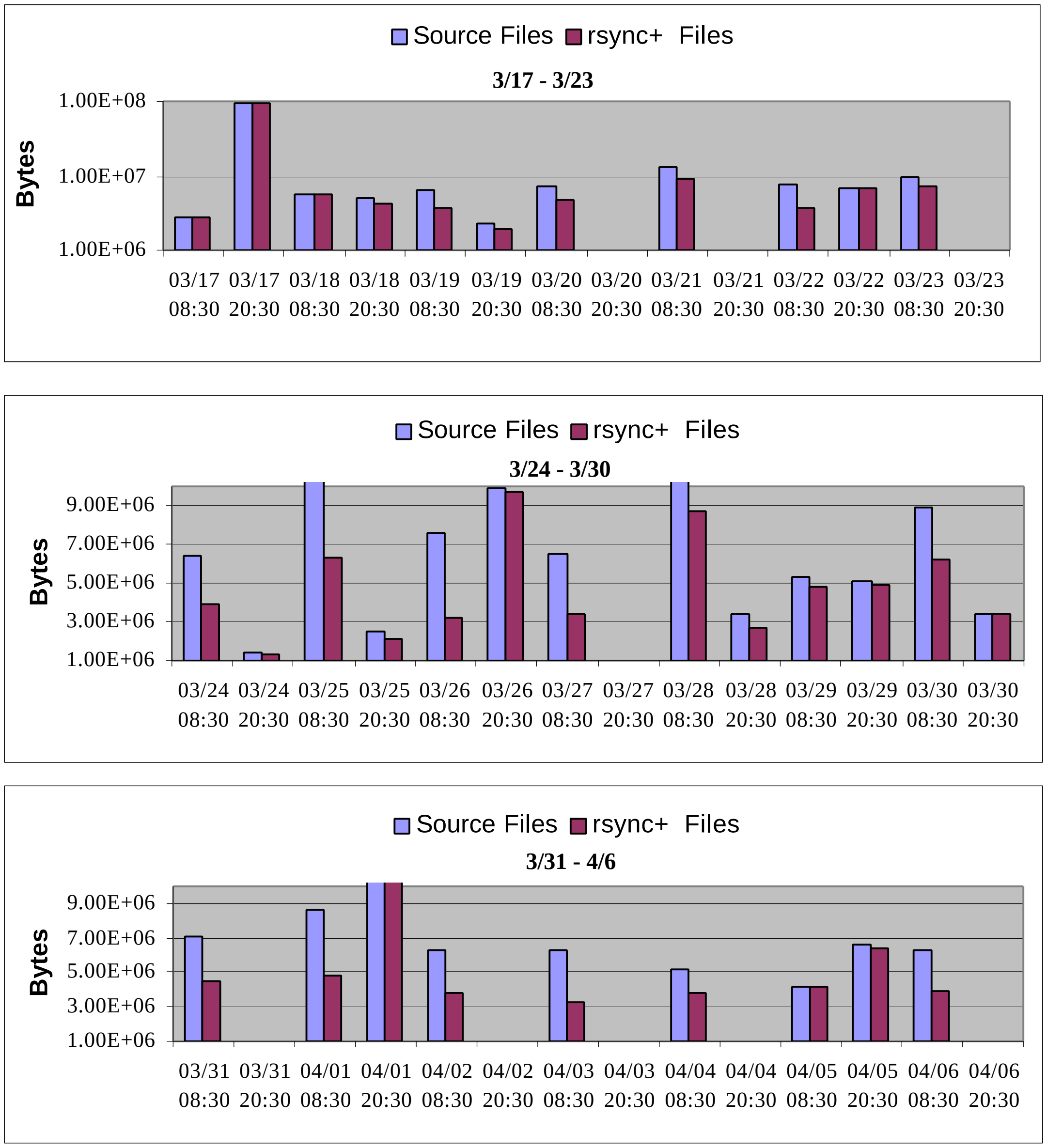

Figure 4.1: Size of Source Files and Rsync+ Output Files

Figure 4.2 shows the real-time duration in seconds for Step 1 (light bars) and Step 3 (dark bars) in each run over the three-week period. The data shows that the network *rsync* between *metalab.unc.edu* and *iris.unc.edu* varied greatly in duration, with some sessions taking 10
minutes or more. One factor here is the high activity rate on the metalab.unc.edu server, which serves over 2 million HTTP and one-third of a million FTP hits per day.
By contrast, the same synchronization activity performed with a local batch file

on the dedicated machine *iris.unc.edu* using rsync+ -f was much faster (dark bars). While further experimental analysis is required to be conclusive, we speculate that the *rsync+* approach of decoupling network communication and file update operations will result in significant

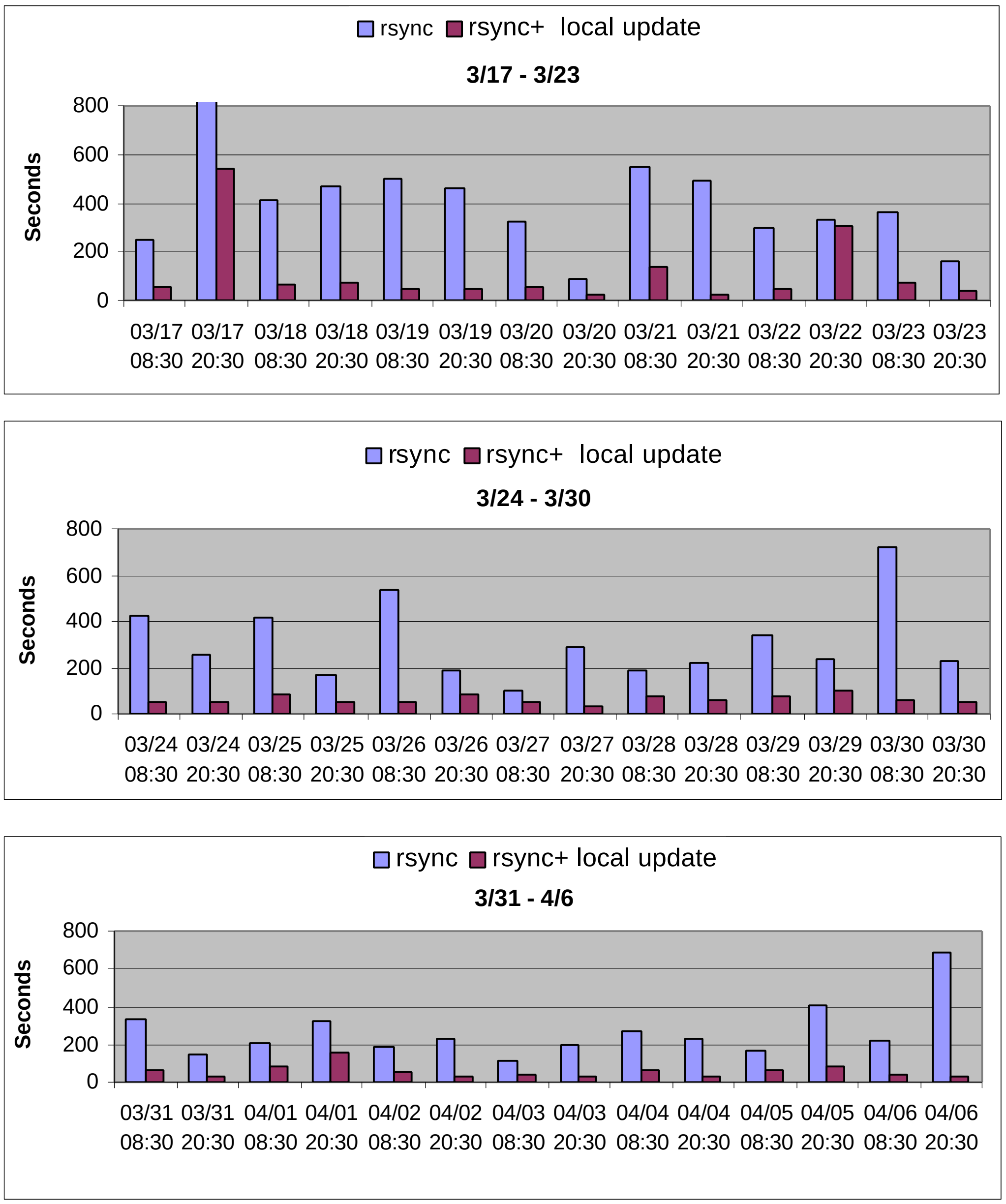

Figure 4.2: Duration of Rsync Update and Rsync+ -f Update

speed-ups in replication update latency when compared with interactive update during a network session. The data in Figure 4.2 supports this conjecture.

5 Conclusions and Future Work

To achieve its vision, the I2-DSI project must develop methods for efficient and scalable replication of source objects in content channels. As the base of application classes that I2-DSI must support grows, replication mechanisms will be deployed a range of source object models (files, database objects, or other object stores). In our work we have begun development on an efficient replication approach for file-based channels by leveraging previous development of a powerful open-source tool for file mirroring.

The mirror experiment presented in Section 4 validates the correctness of our *rsync+* code base in at least one long-duration test. It also has produced encouraging performance data on *rsync+* ---as an example, local processing time for the batch-mode options completed quickly (e.g., about 1 minute in most cases) for the local mirror site. Given the replication scenario envisioned in Figure 2.2, *rsync+* replication could offer a channel provider a form of atomic updates at the remote replicas by blocking access to the replica contents during the update process. During this "black-out" at the local replica, requests could simply be redirected to other I2-DSI servers. Given the timing data from the experimental mirror, we conclude that atomic update could be a supported feature of the I2-DSI replication service. In contrast, during the local *rsync* session with the busy *metalab.unc.edu* server, *rsync* sessions took many (e.g., 2-10 in most cases) times longer to complete, and access delays under an unmodified *rsync* model would be much less attractive.

Our work will continue in the near term with creating and supporting WWW content channels on the now-emerging I2-DSI WAN testbed [16]. In particular, the Linux materials in our local mirroring experiment will be expanded and distributed as a public "Linux Channel" within I2-DSI. Operational experience with this and other file-oriented content channels will expand and refine our understanding of the *rsync+* transport as the basis for a replication solution. As more sophisticated models of I2-DSI replicable service emerge from existing work [17], our view of *rsync+* as a replication transport will also evolve. We believe, however, that *rsync+* offers a very general platform for innovation given its focus on file-based storage, a powerful and pervasive paradigm for many applications.

6 References

[1] http://www.mfsdatanet.com:80/MAE/east.stats.html

[2] Steven Blake, David L. Black, Mark A. Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss, “An Architecture for Differentiated Services”, RFC 2475, Informational RFC, December 1998.

[3] M. Beck and T. Moore, “The Internet2 Distributed Storage Infrastructure Project: An Architecture for Internet Content Channels”, in Computer Networking and ISDN Systems, 1998, 30(22-23):pp. 2141-2148.

[4] G. Carpenter, G. Goldszmidt, M. Beck, T. Moore, B. Dempsey, D. Weiss, “Improving the Availability of Internet2 Applications and Services”, (submitted).

[5] Internet2 DSI Applications Workshop, http://dsi.internet2.edu/apps99.html, UNC-CH, March 1999.

[6] Rsync, available at http://samba.anu.edu.au/rsync

[7] Felix von Leitner, Mirror 1.0, http://www.oasis.leo.org/perl/scripts/net/infosys/ftp/Mirror.dsc.html.

[8] Lee McLoughlin, ftp-mirror, available at ftp://src.doc.ic.ac.u/packages/mirror.

[9] Tridgell, A., and Mackerras, P., “The Rsync Algorithm”, Technical Report, Australian National University Department of Computer Science, June 18, 1996.

[10] Netlib, http://netlib.org/.

[11] E. Grosse, “Repository Mirroring”, ACM TOMS 21:1 (Mar 1995) 89-97.

[12] http://www.internet2.edu/html/multicast.html/wg-plans.html

[13] J. Donnelley, “WWW Media Distribution via Hopwise Reliable Multicast”, 3rd International WWW Conference, Darmstadt, Germany, April 1995.

[14] Ken Miller, K. Robertson, A. Tweedly, and M. White, StarBurst Multicast File Transfer Protocol (MFTP) Specification, Internet Draft, Work in progress, draft-miller-mftp-spec-03.txt, April 1998.

[15] Tridgell, A., and Mackerras, P., “The Rsync Algorithm”, Technical Report, Australian National University Department of Computer Science, June 18, 1996.

[16] See http://dsi.internet2.edu/dsi_map.jpg for testbed status as of April 1999.

[17] M. Beck, T. Moore, B. Dempsey, R. Chawla, "Portable Content Representation of Internet Content Channels in I2-DSI", 4th International Web Caching Workshop (WCW ’99), San Diego, CA, March 30-April2, 1999.

[18] Matthew Lucas, Bert Dempsey, and Alfred Weaver, "MESH-R: Large-Scale, Reliable Multicast Transport", IEEE International Conference on Communication (ICC '99), Vancouver, BC, June 1999.