High Performance Reliable File Transfers Using Automatic Many-to-Many Parallelization*

Paul Z. Kolano

NASA Advanced Supercomputing Division, NASA Ames Research Center
M/S 258-6, Moffett Field, CA 94035 U.S.A.
paul.kolano@nasa.gov

Abstract. Shift is a lightweight framework for high performance local and remote file transfers that provides resiliency across a wide variety of failure scenarios. Shift supports multiple file transport protocols with automatic selection of the most appropriate mechanism between each pair of participating hosts allowing it to adapt to heterogeneous clients with differing software and network access restrictions. File system information is gathered from clients and servers to detect file system equivalence and enable path rewriting so that multiple clients can be automatically spawned in parallel to carry out both single and multi-file transfers to multiple servers selected according to load and availability. This improves both reliability and performance by eliminating single points of failure and overcoming single system bottlenecks. End-to-end integrity is provided using cryptographic hashes at the source and destination with support for partial file retransmission of only corrupted portions. This paper presents the design and implementation of Shift and details the mechanisms utilized to enhance the reliability and performance of file transfers.

1 Introduction

In high-end computing environments, remote file transfers of very large data sets to and from computational resources are commonplace as users are typically widely distributed across different organizations and must transfer in data to be processed and transfer out results for further analysis. Local transfers of this same data across file systems are also frequently performed by administrators to optimize resource utilization when new file systems come on-line or storage becomes imbalanced between existing file systems. In both cases, files must traverse many components on their journey from source to destination where there are numerous opportunities for performance optimization as well as failure. The focus of this work is to support both scenarios with an automated, high performance, high reliability tool that is simple to use and deploy.

A number of tools exist for providing reliable and/or high performance file transfer capabilities, but most either do not support local transfers, require specific security models and/or transport applications, are difficult for individual users to deploy, and/or are not fully optimized for highest performance. This paper presents Shift, which is a new framework for **S**elf-**H**ealing **I**ndependent **F**ile **T**ransfers. Shift provides high performance and resilience for local and remote transfers through a variety of techniques.

* Supported by Task ARC-013 (Contract NNA07CA29C) with Computer Sciences Corporation

Proc. of the 5th Wkshp. on Resiliency in High Performance Computing,
Rhodes Island, Greece, Aug. 28, 2012

These include end-to-end integrity via cryptographic hashes, throttling of transfers to prevent resource exhaustion, balancing transfers across resources based on load and availability, and parallelization of transfers across multiple source and destination hosts for increased redundancy and performance. In addition, Shift was specifically designed to accommodate the diverse heterogeneous environments of a widespread user base with minimal assumptions about operating environments. In particular, Shift is unique in its ability to provide advanced reliability and automatic single and multi-file parallelization to any stock command-line transfer application while being easily deployed by both individual users as well as entire organizations.

Shift consists of a client and a manager component. Figure 1 shows the position of these components within a basic sequential file transfer. A single transfer may consist of many different file operations such as creating directories, copying files, changing attributes, computing checksums, etc. The original client computes the operations that comprise the given transfer and initializes them on the manager. This client, together with any others spawned dynamically, then requests a set of operations from the manager, attempts those operations, and finally reports the results back to the manager. Clients may utilize different applications to carry out file operations depending on availability, performance, and underlying system characteristics. The remainder of the paper will discuss these transfer processing steps in detail.

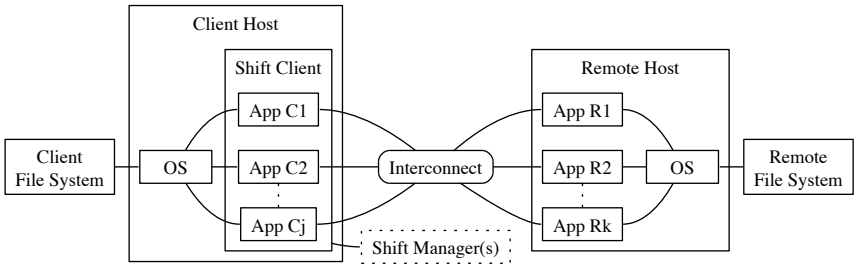

Fig. 1. Shift components within a transfer

Shift's core dependencies are Perl and direct or proxied SSH access to remote hosts using a non-interactive authentication type such as SSH public key authentication. Client hosts are not required to provide SSH access, but Shift's client parallelization features are not possible without it. Shift supports both manually-configured SSH key pairs as well as single sign-on SSH frameworks. The client has initially been tested with a lightweight authentication and authorization framework called Mesh [7], which provides single sign-on using standard SSH key pairs. With minor changes to temporary credential handling, the client can support other authentication frameworks such as the Grid Security Infrastructure (GSI) through GSI-OpenSSH [6].

This paper is organized as follows. Section 2 presents related work. Sections 3 and 4 describe the Shift manager and client. Section 5 details transfer parallelization and load balancing. Finally, section 6 presents conclusions and future work.

2 Related Work

The cp and scp utilities are the de facto standards for local and remote file transfer, respectively. A number of other file transports exist, however, that provide greater performance and/or reliability. BbFTP [3] is a remote transfer utility that supports multiple TCP streams and configurable buffer/window sizes for improved performance as well as a simple retry mechanism for improved reliability. Rsync [18] supports both local and remote transfers and can synchronize files that exist at both the source and destination using partial transfers. This increases performance by minimizing data transfer and improves reliability by correcting corruption. GridFTP [1] offers many of the features of BbFTP and Rsync with a more configurable retry mechanism and additional performance enhancements including UDP-based data streams, partial transfers, and striped transfers across multiple servers. Mcp [8] is a high performance local copy utility that supports multi-threaded single and multi-file copies, processing across multiple nodes, double buffering, and integrated parallel hashing. Shift can utilize any of these tools to take advantage of their enhancements when available.

Several projects modify existing transports to provide enhanced performance and/or reliability. Lim et al. [13] use NaradaBrokering as a more reliable communication medium between client and server within GridFTP. Sultana et al. [20] provide a technique for detecting the location of corruption in files transferred via FTP using a set of signatures appended to each file. If corruption is detected, only a small subset of the file need be transferred again (a feature Shift also provides). The need to modify both client and server software within these projects, however, makes them more difficult to deploy. HPN-SSH [16] is a performance enhancement to OpenSSH that achieves dramatic performance improvements using dynamically adjusted SSH receive windows and a multi-threaded implementation of the AES-CTR cipher. While HPN-SSH requires client and server modification to realize peak performance, either side may be modified without affecting compatibility with stock SSH installations.

Other projects (including Shift) utilize existing transports as building blocks with which to build enhanced capabilities. The Reliable File Transfer (RFT) service of the Globus Toolkit [15] adds reliability using third-party GridFTP transfers initiated from a centralized server. Since the RFT service itself becomes a single point of failure in the initial design, Basney and Duda [2] provide fault tolerance for the RFT service using multiple RFT instances with failover and a synchronized RFT transfer database. The gLite File Transfer Service (FTS) [10] is a reliable transfer service that can be layered on top of GridFTP, RFT, and other services. FTS tracks and monitors all file operations, which are carried out using third-party transfers based on lower-level services. While both RFT and FTS improve the resiliency of transfers, the use of third-party transfers will not fit into the security models of many organizations. Stork [9] is a reliable data placement framework that provides features similar to Shift including support for local transfers, automatic selection between multiple transports, and end-to-end integrity. Stork requires a long-running server accessible with GSI authentication, however, so deployment by individuals is not practical and may be difficult even for organizations.

The source file system will always be a single point of failure when only a single copy of a file exists. Replica management services such as Reptor [11] facilitate the tracking and transfer of multiple copies of the same file to provide data redundancy and

to optimize the source of a particular file. Replica Aware RFT [12] is an extension to the RFT service that allows it to utilize multiple replica servers in the transfer of a single file, thereby increasing fault-tolerance. Peer-to-peer file sharing protocols such as BitTorrent [4] offer similar functionality where clients can utilize multiple data streams for a single file to maximize network utilization from low bandwidth sources and support parallel hashing to verify the integrity of each piece. GridTorrent [21] combines the peer-to-peer functionality of BitTorrent with the speed of GridFTP to achieve high performance file sharing. Since data from individual users and/or already on local file systems is not generally replicated to other locations, Shift focuses on finding multiple access points to the same data source rather than finding multiple sources.

3 Shift Manager

Shift provides a lightweight command-line manager application that facilitates centralized tracking of file operations via two basic functions discussed throughout the remainder. *Put()* adds operations for processing or sets the state of existing operations while *get()* retrieves operations for processing. The location of the manager is shown dotted in Figure 1 as it can be deployed on any of the client host, the remote host, or a dedicated host in a redundant or standalone configuration depending on the needs of the organization or individual user. In addition to tracking, the manager also provides transfer status and performance through manual user inquiries and automated email notifications.

Shift uses a log-structured [17] flat file model for storing tracking data, which keeps dependencies on other components, such as databases, to a minimum, thereby curbing complexity and reducing points of failure. Each file operation is stored as a single line of text containing operation type (e.g. cp, mkdir, etc.), arguments, originating host, run time, size, state, and a message field. The state of a file operation is recorded in put() by appending the operation to the end of one of five log files corresponding to the states *do/doing/redo/done/error*. The same operation can appear in multiple logs or multiple times in the same log but the storage cost is bounded by the number of operations and configured retries possible. This model achieves the put() of one operation in O(1) time and minimizes corruption due to outages/glitches by eliminating data overwrites.

A small metadata file for each transfer records items such as the last used position in each log, counts of operations in each state, etc., which supports the get() of one operation in O(1) time by seeking *do/redo* to their last recorded position and returning the next file operation. Table 1 shows the cost of put() and get() for a single update of varying size. The cost of a million operations in a batch is high, but would not be used in practice since large updates reduce the effectiveness of checkpointing. Get() is more expensive than put() because it performs a number of advanced computations as described in Section 5. In a redundant manager configuration without a shared file system, tracking data must be kept synchronized. This can be achieved through standard mechanisms such as Rsync called from the manager's configurable synchronization hook. Table 2 shows the synchronization cost using Rsync for recording varying numbers of file operations via put() with different numbers of existing operations, which is minimal at even a million existing operations.

Op \\ Files	1-100	1k	10k	100k	1M
put	0.094	0.10	0.20	1.3	14
get	0.15	0.19	0.56	4.8	48

Table 1. Tracking cost (secs)

Exists \\ Puts	1	10k	100k	1M
10k	0.27	0.27	0.35	1.1
100k	0.32	0.33	0.41	1.2
1M	0.89	0.91	0.94	1.8

Table 2. Sync. cost (secs)

Source Location	Path Files	/usr/bin 1841	/usr/lib 15,858	/usr 163,799
local		0.13	0.43	2.9
LAN		0.44	3.5	24
WAN		2.7	210	-
LAN w/ helper		0.39	0.61	3.4
WAN w/ helper		5.4	11	66

Table 3. Init. cost (secs)

4 Shift Client

The user interface to Shift's functionality is provided by a lightweight command-line client that can function as a drop-in replacement for both cp and scp with identical usage conventions and support for the most commonly used options. This provides a known standard interface that makes usage trivial for users of Linux/Unix systems.

4.1 Initialization

A transfer begins when the client is invoked to copy files from a source to a destination. File operations are computed in a separate initialization phase that operates concurrently with file processing and allows directory traversal to complete rapidly instead of binding traversal and transfer together. Local file operations are computed using direct recursive traversal while remote operations are computed using the file manipulation features of the SFTP protocol [5]. Table 3 shows the cost of initializing a recursive transfer under various scenarios. The cost is low for local sources, but can be much higher for remote sources with the high latency of a WAN link. This cost can be greatly reduced using an optional helper script invoked over SSH to compute remote operations locally.

Operations are submitted to the manager via put(), which creates a unique identifier for the transfer, synchronizes the operations to any backup managers, if applicable, and returns the identifier back to the client for use in later operations. After initialization, the client inserts an entry into the user's crontab that periodically invokes itself to check on the status of the transfer. If a processing thread is not detected for the transfer, the cron-invoked client begins processing the transfer itself. Hence, if a client or its associated system crashes, it will eventually be restarted to continue the transfer. The client removes the crontab when directed to do so by the manager. On systems without cron, the client parallelization discussed in Section 5 can provide similar resilience.

4.2 Multiple Transports

During initialization, a client process is forked to process batches of file operations retrieved from the manager via get() using one or more transport protocols. Shift was designed to support a variety of transports. With its basic Perl and SSH dependencies met, Shift is self-contained and can perform local copies using an integrated Perl equivalent of the cp command and remote copies using equivalents of the SFTP and FISH [14] protocols. Shift will automatically take advantage of higher performance options

when available. These currently include Mcp for local copies, BbFTP and GridFTP for remote transfers, and Rsync for both.

For each supported transport, Shift understands how to efficiently transfer a batch of files, how to detect errors, and whether errors are likely recoverable or not. Shift can support other command-line transfer applications in the same manner. The transport is selected dynamically for each batch of files in order of highest estimated performance, which will vary depending on the sizes of the files in the batch. Before any transport is used, a small test transfer is performed to ensure its availability and correct operation with the built-in options available as a last resort. Transfer performance with the various transports will be shown in Section 5.

4.3 End-to-End Integrity

To detect corruption that may occur in the many components a file may traverse, Shift supports optional end-to-end integrity by computing file hashes at the source and destination after successful transmission. A matching hash value provides significant assurance that the file has arrived without corruption. If supported by the transport (e.g. Mcp and the built-in transports), Shift allows the source hash to alternatively be computed as part of the transfer itself, which can result in significant performance gains [8] as the source buffer read from disk for the transfer can be reused for the hash computation. Unlike other transfer applications that only verify bits received over the network, Shift verifies the actual bits stored on the target disk for true end-to-end verification.

Traditionally, differing hash values indicate some unknown portion of a file is corrupt. Shift instead provides a hash tree capability that indicates where the corruption is located at a configurable granularity. The embedded implementation of this capability is roughly equivalent in performance to the standard md5sum utility and is compatible with Msum [8], which is used by Shift when available as it provides significantly enhanced hashing performance. Only the portions of the file deemed corrupted are retransmitted. Partial file transfer is supported natively by Mcp and GridFTP and is also implemented in the built-in transports to augment other transports with this capability. Partially transferred files are supported in a similar manner. Namely, on an aborted transfer, Shift will determine where the source and destination differ and continue the transfer accordingly. Integrity-verified transfer performance will be shown in Section 5. Figure 2 shows a local transfer in which partial corruption and various other failures were induced to illustrate Shift's recovery mechanisms. As can be seen, Shift is able recover from a number of different scenarios automatically.

5 Multi-Host Parallelization

The systems originally chosen by the user to carry out a transfer may be non-optimal for both reliability and performance. Predetermined hosts introduce single points of failure that may not need to exist. By parallelizing a transfer across multiple equivalent hosts, the transfer may still be able to complete even if a failure has occurred along its original path. Parallelization also increases the aggregate CPUs, memory, I/O bandwidth, and network bandwidth available for the transfer, which provides increased performance when all components are functioning properly.

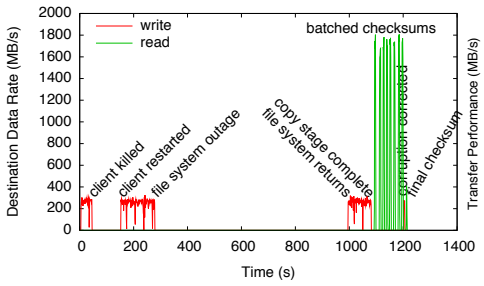

Fig. 2. Automated failure recovery

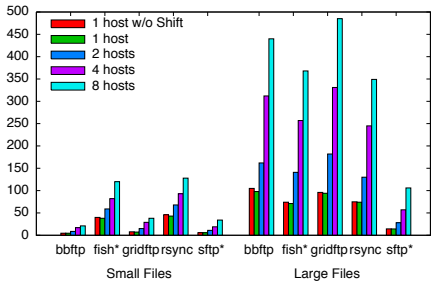

Fig. 3. Remote transfer performance

5.1 Multi-File Parallelization

Any alternate client/remote hosts used must share access to the file system utilized by the original client/remote host. Remote file system equivalence is derived from information supplied by the deploying user/organization via the periodic invocation of an included tool. The client collects similar information incrementally via the mount command during every initialization and sends it to the manager to make client parallelization decisions. In a typical cluster environment where parallelization is most effective, user logins will be distributed across a set of equivalent front-ends. Hence, over time, a complete view of the client environment will be built by simply using the Shift client.

Client parallelization occurs during get() processing. If enough work remains, the manager searches for hosts with equivalent access to the client file system based on the information transmitted during each initialization. The client is then directed to spawn itself on as many such hosts as there is work available by invoking itself on the given host(s) via SSH. All spawns are typically performed during the first get() when an SSH agent from the user's interactive session is often available for authentication if host-based authentication is not supported. Spawned clients immediately add themselves to cron and detach from the SSH session before processing operations normally via get() and put(). Since mount points may differ on each host, paths returned to clients are rewritten based on the manager's file system information. Remote hosts are also parallelized at this point by returning alternate remote paths.

Figures 3 and 4 show the performance of the various remote and local transports (built-ins denoted with '*') for a small file case of 1024 4MB files and a large file case of 64 1GB files without Shift on one host and with Shift on varying numbers of parallel hosts. Figures 5 and 6 show the same cases for integrity-verified transfers using Msum for hash calculations. Remote transfers were between GPFS and Lustre file systems of two cluster front-ends with 8-core 2.8/3.0 GHz Xeon Westmere/Harpertown CPUs and 1 Gb/s NICs over a 10 Gb/s WAN link with HPN-SSH installed at both sites. Local transfers were between Lustre file systems on 3.0 GHz Xeon Harpertowns.

Shift adds minimal overhead except in the local Mcp small file case where the transfer is over before the other hosts even have time to join it. Transfer parallelization achieves significantly better performance than is possible with the underlying ap-

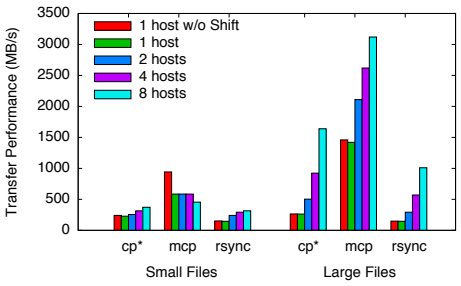

Fig. 4. Local transfer performance

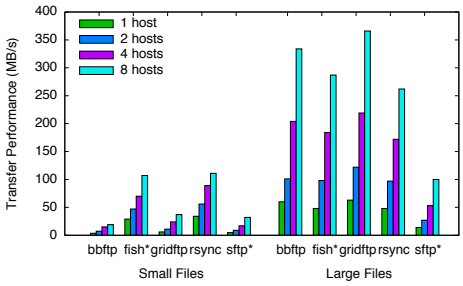

Fig. 5. Remote verified transfer performance

plications on their own or the single hosts on which they normally run. This benefit was achieved automatically over the sequential case as additional host information was made available to the manager during each client initialization. Verification is CPU-intensive so can add significant overhead, but has less of an impact on slow transports where the cost of transmitting the data dwarfs the cost of hashing.

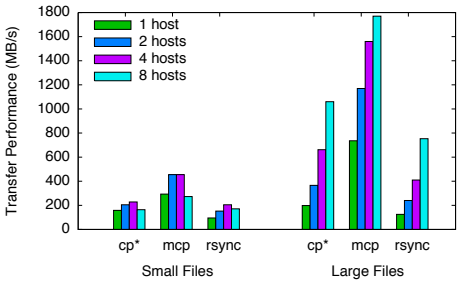

Fig. 6. Local verified transfer performance

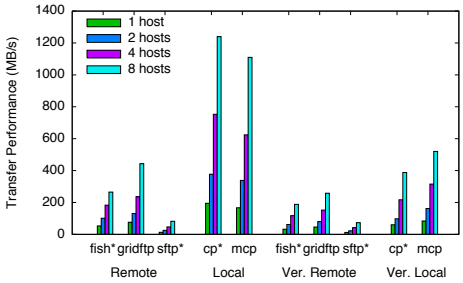

Fig. 7. Single file parallelization performance

5.2 Single File Parallelization

The techniques of the previous section work best for transfers with multiple files that can be broken up into batches of roughly the equivalent size to maximize the utilization of parallel client resources. Transfers of a small number of very large files can create imbalances where some clients are stuck with the bulk of the work while the others remain idle. For this reason, Shift also supports single file parallelization where a single file can be broken up into smaller chunks that can be transferred in parallel.

This capability works by using partial file transfers. Mcp and GridFTP both have the ability to transfer a specific subset of a file beginning at a given offset with a given

length. This functionality was also added to the built-in local and remote transports using standard system seek, read, and write calls and corresponding capabilities within SFTP/FISH. Files above a given split size are broken up by the manager into smaller sized chunks that can be independently transferred using the appropriate partial transfer. Figure 7 shows the benefits of using single file parallelization on a single 64GB file. As can be seen, the single file case achieves roughly 80% of the performance of the original 64 1GB file case with the exception of Mcp. Mcp performance is significantly reduced due to the disablement of direct I/O during partials transfers, which was found to exercise an undesirable Lustre bug, but still scales well in relation to single host performance. In general, Shift's single file parallelization capability allows client workload to always be kept in balance and fully utilized regardless of the sizes of the original files. This capability also allows efficient checkpointing of large files on a single host.

5.3 Load Balancing

While parallelism greatly increases the performance that can be obtained by a single user, the use of too much parallelism by all users can quickly lead to resource contention. Shift provides several different forms of load balancing to distribute and/or reduce the load across a site. All balancing is performed during get() processing based on global transfer activity and load information sent by clients during each manager invocation. The manager first decides if the client should sleep based on user/administrator-defined thresholds for CPU, I/O, network and/or disk utilization applied to the received load information. During parallel transfers, highly loaded clients will be throttled, thereby shifting the balance of work to clients with a lighter load.

Next, the manager decides if the client should sleep based on global load and fairness criteria. Shift's basic strategy is to allow clients to initially spawn in parallel on as many resources as requested since the credentials for doing so (e.g. an SSH agent only available during a user login) may not be available a short time after invocation. The amount of work that a client actually receives, however, is governed by the manager, which, in a centralized deployment, has information on all transfers and the load they are generating on each host based on client information. If the total load is more than the site can handle, enough clients will be directed to sleep until the load is at an acceptable level. To ensure fairness at high loads, clients are throttled in round-robin fashion with each user guaranteed at least one active client before another user gets two.

Once it is determined that a client should proceed, the manager determines which remote hosts have equivalent access to the paths in the next batch of operations. The host from this set with the lowest load is then chosen and the original remote path is switched to the mount point of the selected host before being returned to the client. Using this approach, Shift allows maximum performance when resources are underutilized, but prevents degradation of performance due to overutilization.

6 Conclusions and Future Work

This paper has described Shift, a framework for **S**elf-**H**ealing **I**ndependent **F**ile **T**ransfers. Shift uses a transfer manager to centrally track the status of all file operations throughout the life of a transfer. The manager utilizes remote file system information as well

as information forwarded by clients to find hosts with equivalent access to the client and/or remote file system. If found, multiple clients may be spawned to multiple remote hosts in a many-to-many configuration. Shift adapts to many client configurations by supporting multiple file transports with automated transport selection and validation.

Shift replaces traditional sequential transfers, which are highly vulnerable to failures at every point along the path between the client and remote file systems, with a highly parallel model that is resistant to failures throughout via multiple forms of redundancy and recovery. Outages to the client/remote file systems and the network interconnect between client/remote hosts, which are the remaining single points of failure, are tolerated through an intelligent retry mechanism that classifies failures by recoverability. Via multiple techniques, Shift provides high reliability together with high performance through aggregate resource utilization using client and manager components that are easily deployed by both organizations and individuals. Shift will be released as open source software in the near future [19].

There are a variety of directions for future work. The hash calculations for verified transfers are currently carried out sequentially where each batch of files is hashed first at the source and then verified at the destination. Ideally, all hashes would be computed in parallel and compared afterward to halve the verification cost. Support for other transports should also be investigated.

References

1. W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster: The Globus Striped GridFTP Framework and Server. ACM/IEEE Supercomputing 2005 Conf., Nov. 2005.
2. J. Basney, P. Duda: Clustering the Reliable File Transfer Service. 2nd TeraGrid Conf., Jun. 2007.
3. BbFTP. `http://doc.in2p3.fr/bbftp`.
4. B. Cohen: Incentives Build Robustness in BitTorrent. 1st Wkshp. on Economics of Peer-to-Peer Systems, Jun. 2003.
5. J. Galbraith, O. Saarenmaa: SSH File Transfer Protocol. IETF Internet Draft, Jul. 2006.
6. GSI-Enabled OpenSSH. `http://grid.ncsa.illinois.edu/ssh`.
7. P. Z. Kolano: Mesh: Secure, Lightweight Grid Middleware Using Existing SSH Infrastructure. 12th ACM Symp. on Access Control Models and Technologies, Jun. 2007.
8. P.Z. Kolano, R.B. Ciotti: High Performance Multi-Node File Copies and Checksums for Clustered File Systems. 24th USENIX Large Installation System Administration Conf., Nov. 2010.
9. T. Kosar, M. Livny: A Framework for Reliable and Efficient Data Placement in Distributed Computing Systems. Jour. of Parallel and Distributed Computing, vol. 65, no. 10, 2005.
10. P. Kunszt, P. Badino, R. Brito da Rocha, J. Casey, A. Frohner, G. McCance: The gLite File Transfer Service. 1st EGEE User Forum, Mar. 2006.
11. P. Kunszt, E. Laure, H. Stockinger, K. Stockinger: File-Based Replica Management. Future Generation Computer Systems, vol. 21, no. 1, Jan. 2005.
12. Y. Lee, E. Kim, H.Y. Yeom: Replica Aware Reliable File Transfer Service for the Data Grid. 4th IEEE Intl. Conf. on eScience, 2008.
13. S. Lim, G. Fox, S. Pallickara, M. Pierce: Web Service Robust GridFTP. 10th Intl. Conf. on Parallel and Distributed Processing Techniques and Applications, Jun. 2004.
14. P. Machek: FIles transferred over SHell protocol (V 0.0.2). `http://cvs.savannah.gnu.org/viewvc/*checkout*/mc/mc/vfs/README.fish`.
15. R.K. Madduri, C.S. Hood, W.E. Allcock: Reliable File Transfer in Grid Environments. 27th IEEE Conf. on Local Computer Networks, Nov. 2002.
16. C. Rapier, B. Bennett: High Speed Bulk Data Transfer Using the SSH Protocol. 15th ACM Mardi Gras Conf., Feb. 2008.
17. M. Rosenblum, J.K. Ousterhout: The Design and Implementation of a Log-Structured File System. ACM Trans. on Computer Systems, vol. 10, no. 1, Feb. 1992.
18. Rsync. `http://samba.org/rsync`.
19. Shift. `http://shiftc.sourceforge.net`.
20. A. Sultana, M.F. Bashir, M.A. Qadir: CFiTT - Corrupt Free File Transfer Technique Over FTP. 1st IEEE Intl. Conf. on Information and Emerging Technologies, Jul 2007.
21. A. Zissimos, K. Doka, A. Chazapis, N. Koziris: GridTorrent: Optimizing Data Transfers in the Grid with Collaborative Sharing. 11th Panhellenic Conf. on Informatics, May 2007.