# Automatically Encapsulating HPC Best Practices Into Data Transfers*

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

This paper presents the Shift automated transfer tool and the mechanisms it employs to achieve better performance while preserving the stability of HPC environments. Shift encapsulates best practices understood by domain experts during transfers so that scientists can focus on their science without the need to study file transports, resource management, and file systems as well. Shift understands how to utilize the variety of transports that might be deployed throughout a widely distributed user base, how to maximize the performance achievable by each, and the scenarios in which each is most effective. Shift understands which resources are available in a particular HPC environment and how to utilize them for significant performance increases while preventing resource exhaustion. Finally, Shift understands the file systems to which and from which files may be transferred and the nuances to their use that affect performance and stability behind the scenes.

## Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Applications; D.4.3 [**File Systems Management**]: File organization; D.4.4 [**Communications Management**]: Network communication; H.3.4 [**Systems and Software**]: Performance evaluation

## General Terms

Performance; Reliability; Measurement

## Keywords

high performance computing; data transfer; data archival; best practices; parallelization

## 1. INTRODUCTION

In an ideal world, HPC users would have unlimited, automatically backed up storage for their data sets with unlimited computational power at their disposal so they would never need to move them. In the real world, however, users are given a finite allocation of scratch space that is not backed up and a finite allocation of CPU time. Hence, users must transfer data to/from archive systems to make room for new data or to backup/restore old data, and must transfer from/to remote systems for less costly pre-/post-processing.

Users accept that data transfers are a part of life in HPC environments, but their requirements of data integrity, ease of use, and turnaround time may often be at odds with each other and with the requirements of system owners and administrators, such as environment stability and cost. For instance, the transfer tool that is easiest to use or that provides the most assurance of data integrity may not necessarily be the one the provides the fastest turnaround time. A file structure that may be easier for a user to manipulate while archived may degrade the performance of the tape system and require more online disk storage to accommodate the reduced speed. Achieving the best turnaround time possible for one user may have a negative impact on stability for all other users. Resolving these and other similar conflicts in a manner acceptable to all requires expert knowledge in how and when to use transfer tools most effectively, the properties of the local environment and its resources, and the peculiarities of individual file systems.

A variety of tools exist for providing reliable and/or high performance file transfer capabilities, but most of them do not fully address such conflicts so do not perform optimally in all transfer scenarios. This paper presents a number of transfer-related considerations and the accepted best practices for addressing them together with how they are encapsulated within Shift, which is a framework for **S**elf-**H**ealing **I**ndependent **F**ile **T**ransfers. By embedding these best practices within Shift, users get the performance and integrity achievable by domain experts in an easy to use interface that is drop-in compatible with cp/scp while preserving the stability of the environment for other users and administrators.

Shift consists of a lightweight command-line client that interacts with and performs actions as directed by a manager component. The manager is a command-line application that is invoked by the client either directly or via SSH, and that facilitates centralized tracking of various file operations such as traversing and creating directories, copying files, changing attributes, and computing and verifying checksums. After the client initializes the basic attributes of the transfer, it enters a loop of requesting operations from

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



*HUST2015,* November 15-20, 2015, Austin, TX, USA
© 2015 ACM. ISBN 978-1-4503-4000-7/15/11...$15.00
DOI: http://dx.doi.org/10.1145/2834996.2834997

```
> shiftc --create-tar /nobackup/user1/dataset1 archive1:dataset1.tar
Shift id is 36
Detaching process (use --status option to monitor progress)

> shiftc --status
id | state |        dirs |       files |     file size |  date |        run |     rate
   |       |        sums |       attrs |      sum size |  time |       left |
---+-------+-------------+-------------+---------------+-------+------------+---------
34 | error |         0/0 | 23121/23121 | 39.5TB/39.5TB | 10/02 | 2d14h32m5s |  175MB/s
   |       | 46222/46242 | 23111/23121 |     79TB/79TB | 10:26 |            |
35 | done  |         1/1 |   5131/5131 |   303GB/303GB | 10/05 |      1m35s | 3.19GB/s
   |       | 10262/10262 |   5132/5132 |   605GB/605GB | 12:28 |            |
36 | run   |       24/24 | 26656/26656 | 1.78TB/1.78TB | 10/06 |   2h48m37s |  176MB/s
   |       | 15463/53312 |    10/26684 | 1.02TB/3.56TB | 12:11 |   1h47m55s |
```

Figure 1: Shift transfers are initialized using cp/scp syntax and run in the background with on-demand status.

the manager, attempting those operations, and reporting the results back to the manager. Clients may utilize different underlying built-in and/or external transport applications to carry out file operations depending on availability, performance, and underlying system characteristics. By supporting multiple transport applications, each with different advantages and disadvantages, Shift is able to provide better performance than any of these transports individually while augmenting them with advanced capabilities and low-level optimizations that most do not support on their own.

Shift provides high performance and resilience for local and remote transfers through a variety of techniques including 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. The details of Shift's basic operation have been described in previous work [9]. Shift has been in production at the NASA Advanced Supercomputing (NAS) facility for three and a half years and in the past year alone has facilitated transfers of over 14PB of data including approximately 8PB in local transfers, 4PB in local area network (LAN) transfers, and 2PB in wide area network (WAN) transfers. Shift is used for everything from user data transfers, to disaster recovery backups to/from remote organizations, to rebalancing entire multi-PB Lustre file systems.

Figure 1 shows the main components of Shift's user interface. Transfers are initialized using easily understood cp/scp syntax with similar basic options together with Shift-specific options to access its extended capabilities. In this case, Shift is being directed to create a tar file named "dataset1.tar" in the user's home directory on the remote system "archive1" containing the contents of the local directory "/nobackup/user1/dataset1". Unlike cp/scp, which remain in the foreground until the transfer completes, Shift, by default, backgrounds itself immediately after printing a transfer identifier, which may be used to stop, restart, or show detailed status of the transfer. The main window into Shift's operation is through the `--status` option, which shows all of the user's transfers in a given time period. Displayed items include the transfer state, the number of operations completed, the number of operations total, the start and run time, the estimated time remaining, and the transfer rate.

This paper is organized as follows. Section 2 presents related work. Section 3 describes the tuning of individual transports and the conditions under which each is most effective. Section 4 discusses Shift's resource management for parallelizing transfers while avoiding resource exhaustion. Section 5 details optimizations related to specific file systems. 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. Rsync [24] 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. Parsync [20] is a wrapper around rsync that utilizes parallel rsync instances to improve performance and an external utility for faster directory traversal during transfer initialization.

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. 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. Bbcp [8] is a remote transfer tool that offers other optimizations such as compression and direct I/O as well as a checksum capability for enhanced data integrity. Fast Data Transfer (FDT) [6] is a remote transfer tool that utilizes multi-threading and multiple TCP streams together with checksum and restart capabilities for greater reliability and integrity.

Mcp [11] 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. Streaming parallel distributed cp (spdcp) [16] has similar goals as mcp and achieves very high performance on clustered file systems using MPI to parallelize transfers of files across many nodes. Dcp [5] utilizes MPI to distribute local copies across many nodes without any centralized state and with integrity verification via checksums. Pcp [21] can also parallelize local copies using MPI, but in addition to integrity verification, it provides the abilities to split the

processing of large single files, stripe files automatically on Lustre file systems, and restart interrupted transfers.

Ong et al. [18] describe the parallelization of cp and other utilities using MPI. Their cp command, however, was designed to copy one file to many nodes unlike mcp, spdcp, and pcp, which were designed to allow many nodes to copy parts of the same file. Desai et al. [4] use a similar strategy to create a parallel rsync utility that can synchronize files across many nodes at once. HPSS Tar [7] optimizes tar by allowing archives to be created/extracted directly to/from HPSS, bypassing local storage. It uses multi-threading, buffer management, and HPSS network striping capabilities to significantly increase tar performance. The pltar utility [17] uses MPI to parallelize archive creation and extraction and achieves very high performance on Lustre file systems.

Several projects modify existing transports to provide enhanced performance through better optimization of TCP window sizes. Thulasidasan et al. [27] modify GridFTP to dynamically adjust the TCP window and buffer sizes based on latency values embedded within data packets. This technique improves performance at a fraction of the memory usage of static values. Prasad et al. [22] modify GridFTP to differentiate between congested and non-congested network paths based on out-of-band UDP packets so TCP send and receive windows can be adjusted optimally. Yildirim et al. [28] additionally take parallel streams into account and attempt to balance the TCP window size and number of streams. HPN-SSH [23] 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. The gLite File Transfer Service (FTS) [13] is a reliable transfer service that can be layered on top of GridFTP, RFT, and other services. FTS monitors all file operations, which are carried out using third-party transfers based on lower-level services. Globus Online [2] provides a similar service as RFT and FTS using centralized tracking and third-party GridFTP transfers with an easy to understand web interface. While all three projects improve the usability and resiliency of transfers, the use of third-party transfers will not fit into the security models of many organizations. Stork [12] 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.

# 3. TRANSPORT TUNING AND SELECTION

The average scientist typically has several transport choices available on the systems they utilize, but may not understand how to use all of them, how to get the best performance out of each, the different scenarios under which each is most effective, or what idiosyncrasies each may have that could come into play during their transfers. Shift encapsulates all of this knowledge in a single tool that presents a simple, well-understood cp/scp-style interface while automatically optimizing and selecting the most appropriate transport to achieve the lowest turnaround time for the user.

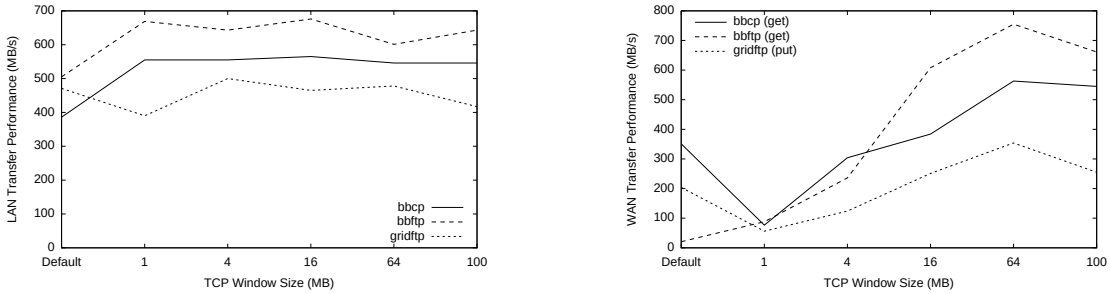
Shift supports a number of the transports discussed in Section 2. In particular, Shift incorporates the external tools `bbcp`, `bbftp`, `gridftp`, `mcp`, and `rsync`, has built-in perl equivalents of `cp` and `sftp` (denoted in figures as `cp-perl` and `sftp-perl`) based on the perl File::Copy and Net::SFTP::Foreign modules, respectively, and a built-in custom perl variant of the `fish` protocol [14]. Other potential transports discussed in Section 2 are not supported because they are either not currently available or do not have the ability to transfer two files to two different directories in the same invocation, which is necessary to efficiently implement Shift's batching of arbitrary files. The batching capability is especially important for remote transfers to avoid incurring authentication overhead on every file.

Unless otherwise indicated, all local and LAN performance tests were run between Lustre file systems mounted on dedicated 2x12-core 2.5 GHz Intel Xeon Haswell nodes with 128 GB DDR4 memory connected via dual Infiniband 4x FDR channel adapters. WAN tests were run from a GPFS file system mounted on shared 2x6-core 2.8 GHz Intel Xeon Westmere front-ends with 192 GB DDR3 memory over a 10GE WAN link to a Lustre file system mounted on shared 4x8-core 2.6 GHz Intel Xeon Sandy Bridge front-ends with 64 GB DDR3 memory. Note that WAN measurements were gathered on a production link with an unknown and varying level of other activity so may not represent the absolute maximum performance achievable by each transport.

## 3.1 TCP Window Size

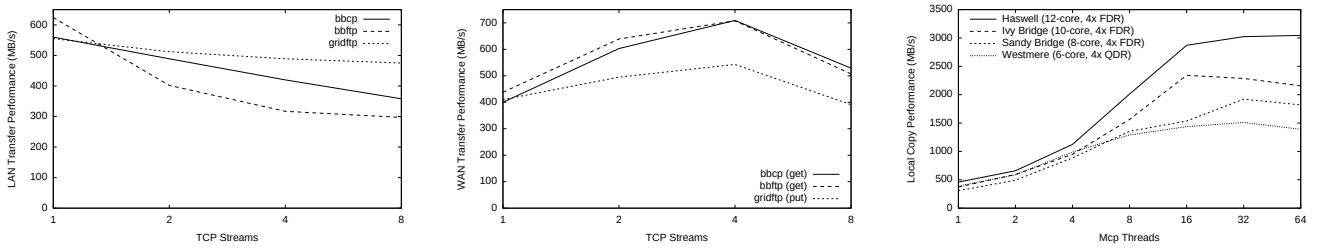
The non-SSH remote transports supported by Shift (`bbcp`, `bbftp`, and `gridftp`) initiate direct TCP links between source and destination. All three of these transports expose options in the underlying TCP layer that can have a significant impact on performance. In particular, the two settings that impact performance the most are the TCP window size and the number of parallel TCP streams. The TCP window size specifies the amount of data the sender/receiver is willing to buffer before it must wait for an acknowledgment. To maximize the performance of a single TCP stream, generally accepted wisdom states that the window size should be set to the bandwidth delay product (BDP) (e.g. [27]), which is the link's bandwidth multiplied by the round-trip time between hosts. Each transport has its own default value, which is further constrained by the operating system settings (e.g. in Linux, the sysctl `net.core.[wr]mem_max` values, which specify the maximum send/receive buffer sizes, and the sysctl `net.ipv4.tcp_[wr]mem` values, which specify the min/default/max settings for TCP window auto-tuning).

Figures 2a and 2b show the performance of transferring a 64GB file over the LAN and WAN, respectively, using the `bbcp`, `bbftp`, and `gridftp` transports with 4 streams over default and varying window sizes. The bandwidth of the LAN connection is approximately 40 Gb/s using IPoIB with




(a) LAN TCP window performance (b) WAN TCP window performance

Figure 2: Adjusting the TCP window used on the WAN can significantly increase performance over transport defaults.




(a) LAN TCP stream performance (b) WAN TCP stream performance (c) Mcp multi-threading performance

Figure 3: Internal transport parallelization increases performance by more greedily consuming resources.

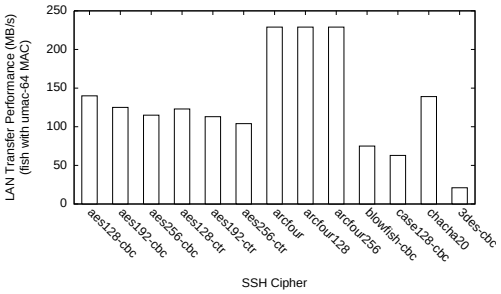
a round-trip latency of 0.035ms for a BDP of 175kB. The bandwidth of the WAN connection is 10 Gb/s with a round-trip latency of 66ms for a BDP of 83MB. As can be seen, adjusting the window size on the LAN has minimal impact as even the default window sizes are greater than the LAN BDP. On the WAN side, however, adjusting the window size has a significant impact over the default value. An increase of almost 37x was observed for `bbftp` due to extremely poor performance with its default setting. Both `bbcp` and `gridftp` performed much better with default values, but still saw significant increases of 1.6x and 1.7x, respectively.

Shift utilizes this knowledge of TCP behavior to try to set the window size automatically for the user, which is absolutely essential in the case of `bbftp`. While determining the latency to the destination is easily achieved using the standard ping tool, determining a site's bandwidth automatically a priori is non-trivial and even determining the speed of the network interface card (NIC) is difficult as a non-root user. Since the amount of information available is limited, Shift uses a heuristic-based approach to classify systems into either a default case or a 10GE WAN case, the idea being that while 1GE NICs are readily available for even residential users who have lower bandwidth, 10GE NICs are still costly enough to be limited to sites that have the bandwidth to utilize them effectively. For the default case, a static window of 4MB is used, which lies in the middle of being somewhat oversubscribed for LANs and residential WANs or somewhat undersubscribed for longer haul 1GE WANs. For WAN transfers, if a host is determined to have a 10GE NIC using the `lspci` utility, the latency to the destination will be measured and the window will be chosen as if the link had 10 Gb/s bandwidth up to the OS sysctl limits. While unlikely to pick the precise ideal window size, the computed window size should generally be on the correct order to provide better than default performance in most cases without the more complex calculations of other work [22, 27, 28].

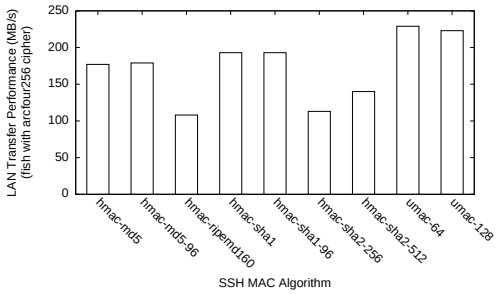
## 3.2 Transport Parallelism

Four of Shift's supported transports (`bbcp`, `bbftp`, `gridftp`, and `mcp`) have configurable parallelism that allow additional resources to be brought to bear when sequential operation is not utilizing available resources completely. In the case of `bbcp`, `bbftp`, and `gridftp`, parallelism takes the form of using multiple TCP streams. Using multiple streams can have several benefits. First, it allows increased performance on hosts whose operating systems do not have properly configured TCP window maximums and/or TCP auto-tuning values. While each stream may run at less than maximum throughput, combining multiple streams together may still allow the transfer to utilize all available bandwidth. Similarly, it can overcome undersubscribed default or specified TCP window sizes in the transports themselves. This is especially relevant to Shift, which may estimate window size incorrectly. Finally, multiple streams may help overcome interference by cross traffic on links by more greedily trying to consume bandwidth.

Figures 3a and 3b show the performance of transferring a 64GB file over the LAN/WAN, respectively, using the `bbcp`, `bbftp`, and `gridftp` transports with a 4MB/64MB window size and varying numbers of streams. As can be seen, in the LAN case, using parallel streams has a detrimental effect on transfers, with a particularly high impact on `bbftp`, as there




(a) SSH cipher performance




(b) SSH MAC algorithm performance

Figure 4: The default SSH cipher and MAC algorithm settings may not always be optimal for performance.

is less cross traffic to interfere with the transfer so additional streams simply add more overhead. In the WAN case, transfers fared better overall at 4 streams although even with a single stream, periods were observed where the full 10 Gb/s bandwidth was being consumed. So while a single stream is enough to achieve maximum performance on an idle WAN link, the additional streams likely helped overcome interference by other activity on the network. Shift encapsulates these results and selects one stream during LAN transfers and four streams during WAN transfers.

In the case of `mcp`, parallelism takes the form of using multiple threads. Figure 3c shows the performance of transferring 64 1GB files using `mcp` with a varying number of threads across different node types. As can be seen, maximum performance can be increased about 4x on Westmere nodes and above 6x on the others compared to a single thread. While performance per thread decreases as more threads are added, total performance increases up to the point at which the system is saturated and the overhead from more threads ends up decreasing performance. Like other transport parameters, Shift allows the number of threads to be centrally configured on the manager. Within the NAS environment, `mcp` is configured with 4 threads as higher thread counts were thought to overwhelm some front-end systems when multiple users were running local transfers simultaneously, or the same user kicked off multiple transfers or multi-client transfers on the same host. An alternative to limiting parallelism at the transport level is Shift's global throttling mechanism, which will be discussed in Section 4.2.

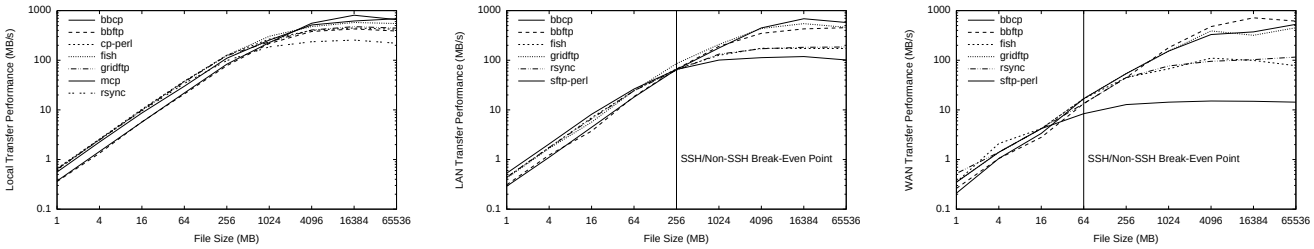
## 3.3 SSH Cipher and MAC Algorithm

Three of Shift's supported remote transports (the built-in `fish` protocol, `rsync`, and the built-in `sftp-perl` protocol) utilize an SSH pipe as their underlying communication medium, so their performance is directly proportional to SSH performance. While SSH does not have configurable TCP windows or parallelism settings, users can install the HPN-SSH [23] patches, which improve TCP window handling and increase cipher parallelism. Regardless of whether these patches are installed or not, however, the performance achievable by SSH depends greatly on the underlying choice of cipher and message authentication code (MAC) algorithm. By default, the client and server will agree on a cipher and MAC algorithm based on their preferred/supported lists in the `ssh_config` and `sshd_config` files, respectively. If a user does not specify them explicitly, they may be left with selections that are non-optimal for performance.

Figures 4a and 4b show the performance of transferring a 16GB file over the LAN with the `fish` transport while varying the SSH cipher and MAC algorithm (note that OpenSSH 7.1 was used to characterize current ciphers and MAC algorithms instead of OpenSSH 5.8, which was used for all other tests). As can be seen, there are significant differences in performance. The arcfour cipher variants achieve over 63% greater performance than the next fastest cipher (aes128-cbc) while the umac-64 MAC algorithm achieves over 18% greater performance than the next fastest MAC algorithm (hmac-sha1). While the arcfour variants are being phased out due to security concerns, they can still be used when the confidentiality of the data is not critical or in more secure environments such as in LAN transfers. Also note that these options still offer more privacy than the default modes of the non-SSH transports, which have completely unencrypted data streams. Shift allows the preferred order of ciphers and MACs to be centrally configured in the manager and the Shift client will automatically check which are available on the client host before transfers begin.

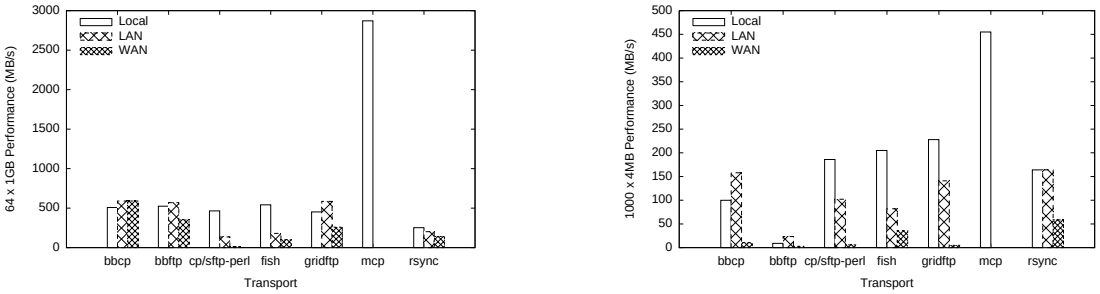
## 3.4 Transport Selection

While the performance of each individual transport has been optimized, it must still be determined if there are scenarios in which some transports perform better than others. Figures 5a, 5b, and 5c show the performance of the transports supported by Shift on single files of various sizes transferred locally, over the LAN, and over the WAN, respectively. Note that these figures use a logarithmic scale to show differences at low file sizes more clearly. In the LAN and WAN cases, there are clear break-even points at which the SSH-based transports switch from having greater performance to less performance than the non-SSH transports. For LAN transfers, this point is 256MB, while for the WAN, it is approximately 64MB. This is the point at which the overhead of creating multiple processes and establishing multiple streams catches up to the lower overhead but slower performance of the SSH-based transports. Surprisingly, the network-based transports perform quite well locally with `bbcp` surpassing even `mcp` at 4GB and 16GB. In the local case, there is a clear break-even point for `bbcp` at 1GB. `Rsync` does not perform as well locally due to checksumming that cannot be parallelized between source and target as in the remote case.




(a) Local transport performance (b) LAN transport performance (c) WAN transport performance

Figure 5: Transport speed varies by file size with SSH-based transports outperforming non-SSH transports at small file sizes.




(a) Large file batch performance (64 x 1GB) (b) Small file batch performance (1000 x 4MB)

Figure 6: Relative performance between transports may change when transferring multiple files due to per file startup overhead.

While single file performance gives a good indication of the raw speed of each transport, it does not fully characterize the overhead when transferring multiple files and performance can be skewed by file system limitations. Figures 6a and 6b show the local, LAN, and WAN performance of each transport when transferring 64GB in a batch of 64 1GB files and 4GB in a batch of 1000 4MB files, respectively. As can be seen, `mcp` fares significantly better than the others in these tests since its performance is no longer inhibited by the single file write limitations of Lustre. In the small batch WAN case, all of the non-SSH transports have poor performance with small files. `Bbftp` in particular has extremely bad performance with small files over all mediums, likely due to its use of new sockets for every single file. Shift uses the break-even points and batch data to automatically select the most effective transport based on availability on the client and the average size of each batch being transferred. By dynamically choosing the transport that is most efficient for each batch of files, Shift can achieve higher performance than individual transports can achieve on their own.

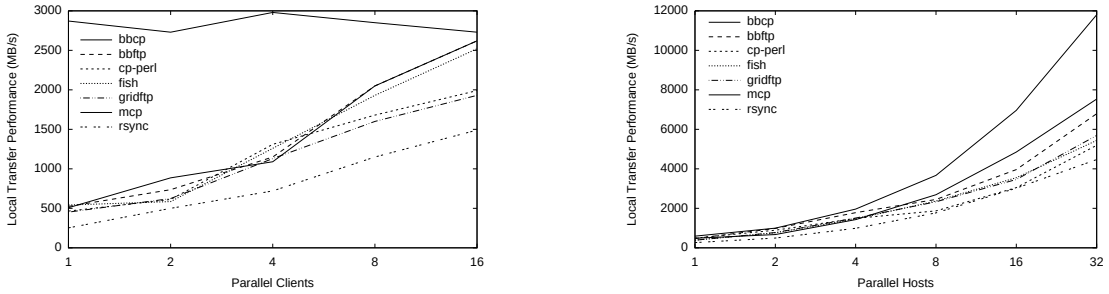
# 4. GLOBAL RESOURCE MANAGEMENT

While optimizing individual transports has significant benefits on dedicated resources, in a production environment, transfers are typically running on resources that are shared among all users for transfer as well as non-transfer activity so there is a high probability of interference with other processes. In HPC environments, while resources are shared, there is also likely to be significant spare capacity at various times, which provides an opportunity to increase performance dramatically. Shift provides the ability to both take advantage of excess capacity as well as limiting interference.

## 4.1 Parallelization

Shift supports two types of parallelization that may be used individually or in tandem: client parallelization and host parallelization. Client parallelization allows multiple Shift clients to run on the same host when a transport does not have its own parallelism or when a single client cannot fully utilize all the host's resources. Figure 7a shows the local performance of each supported transport when transferring a 64GB batch of 64 1GB files across varying numbers of clients. As can be seen, `mcp` gets little benefit from client parallelization in this case as its built-in parallelism already maximizes single host performance. The other transports, however, see significant performance gains.

While client parallelization can help transports achieve the maximum performance on a single host, this performance is only a fraction of what is available across an entire HPC environment. To overcome single system bottlenecks and allow transfers to aggregate the resources of many hosts, Shift also supports host parallelization where clients are spawned on additional hosts with access to the appropriate file systems. Figure 7b shows the local performance of each transport when transferring a 512GB batch of 32 16GB files across varying numbers of hosts. As can be seen, the performance benefits of host parallelization are significantly better than those of client parallelization due to the increase in aggregate CPU, memory, and file system bandwidth. To enhance the applicability of both parallelization mechanisms, Shift also supports splitting of single files (both local and remote) across clients/hosts. This capability relies on transports that support partial file transfer, so cannot be used with `bbcp`, `bbftp`, or `rsync`.




(a) Client parallelization performance

(b) Host parallelization performance

Figure 7: Using parallel clients/hosts takes advantage of spare resource capacity on a single host/across multiple hosts.

While the precise details of Shift's parallelization implementation have been discussed in previous work [9], the basic mechanism is that the Shift manager is provided with local file system mount information across user-accessible front-end systems by administrators (through automated scripts). Shift understands how to utilize this information to determine equivalency of a variety of file system types including CXFS, DMF, GPFS, Lustre, and NFS. Shift utilizes file system equivalency to determine onto which hosts a transfer can be parallelized. Shift automatically learns about the file systems on the user's remote client systems as they run remote transfers. Alternatively, a host list or host file (e.g. `$PBS_NODEFILE`) can be given to explicitly specify the client hosts to use in the transfer. Once Shift has this information available, parallelizing a transfer on N hosts is as simple as adding the option "`--hosts=N`".

## 4.2 Global Throttling

With the ability to trivially throw massive resources into a single transfer comes the promise of significantly lower turnaround time for users, but the danger of easily overwhelming components of the environment relied upon by other users and processes. For instance, highly loaded CPUs on front-end systems can cause slow interactive response time, file system overload can decrease I/O rates within jobs, thereby wasting computational resources, or tape-backed file systems may be filled up faster than tapes can be written, thereby causing a denial of service for users trying to retrieve files. In general, the more users that take advantage of the parallelization capability, the greater the potential for resource exhaustion becomes.

To allow users to utilize parallel resources while preserving the stability of the environment, Shift supports several varieties of throttling. Client hosts within individual transfers can be throttled at a given CPU percentage, target disk usage, I/O rate, and/or network rate. These can be specified by users directly so they don't take up too much of their own system's resources or may be specified centrally for all transfers. More importantly, Shift supports throttling globally across all transfers involving any/all users, any/all hosts, and/or any/all file systems. Transfers involving users or hosts may be throttled at a given I/O and/or network rate while transfers involving file systems may be throttled at a given disk usage and/or I/O rate. These mechanisms allow all three of the example scenarios above to be handled efficiently. CPU utilization on front-ends can be limited by client throttling. File system overload can be prevented by specifying the portion of the file system bandwidth that may be taken up by transfers. Finally, transfers to tape-backed file systems can be suspended and resumed automatically at specific disk utilizations.

Global throttling is based on information returned from clients on the various loads they are generating while requesting a new batch of operations to process. When a client requests a new set of operations from the manager, the manager examines the current loads of all transfer to determine if any limits relevant to the invoking transfer are being violated. If so, the manager divides up the corresponding resource equally among the users of all relevant transfers. When the collective load of the invoking user's transfers are under that user's share, those transfers will proceed at full rate with any unused resources temporarily divided up amongst other users. If the user's transfers exceed their share, that share is divided up equally among all their transfers. Transfers that are over their portion of the user's share are directed to sleep for the time needed to bring their average load down to their share. This approach allows all users to have a fair share of available resources during periods of heavy activity, but allows individual users to consume up to the full threshold when resources are idle.

Figure 8 shows a 512GB parallel local file transfer without throttling and with write throttling on the destination file system at 4 GB/s. As can be seen, without throttling, the transfer operates at maximum speed until completion. When the destination file system is throttled, however, the transfer oscillates above and below the threshold in an attempt to keep its average performance at the specified limit. The synthetic test in the figure represents a worst case where many transfers of the same size are initiated at the same time so their periods of activity and inactivity are synchronized, thereby producing higher highs and lower lows than would happen in a normal production setting where transfers will have varying sizes and start times leading to activity that more closely straddles the throttling limit.

## 4.3 Load Balancing

In the case of local transfers, all file system activity occurs on the client host. In the case of LAN/WAN transfers, however, two hosts are each performing one half of the total I/O.

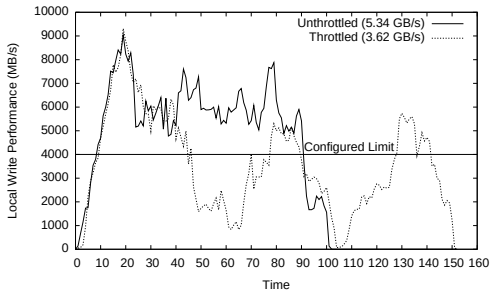



Figure 8: Global throttling ensures fairness and preserves the stability of the environment.

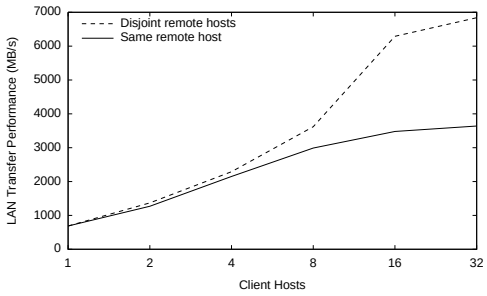
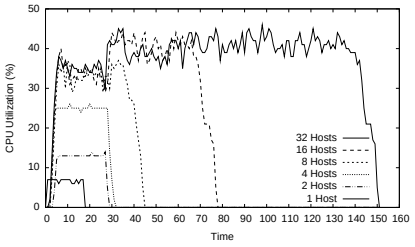
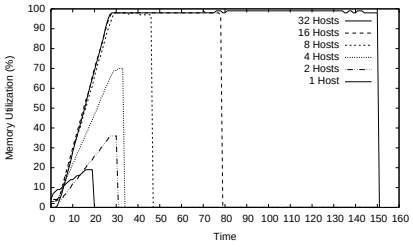


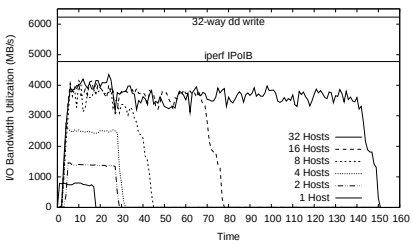

Figure 9: Load balancing avoids single system bottlenecks in multi-user and parallel transfers.




(a) CPU utilization




(b) Memory utilization




(c) I/O utilization

Figure 10: IPoIB bandwidth is the likely bottleneck on the destination host during a parallel all-to-one remote transfer.

Users are creatures of habit and have a tendency to specify the same host for every login or transfer destination. For example, at the NAS facility, even though a load balancing alias exists for front-end systems, inevitably the lowest numbered front-end, which is also featured in examples in the site user guide, always ends up having the most users logged in. While basic login sessions do not typically consume significant system resources, file transfers stress all aspects of the system including CPU (especially during integrity-verified transfers), memory, network interconnects, and file system I/O. If users behave similarly with transfers and all end up running their remote transfers to the same front-end, it may degrade the performance of their transfers unnecessarily as other front-ends may have resources available.

Figure 9 shows the aggregate performance of a varying number of client hosts each performing a 16GB `bbcp` LAN transfer to the same or disjoint remote hosts. As can be seen, when the remote hosts are independent from each other, the performance continues to increase as more remote transfers are initiated. When the remote host is the same across transfers, however, aggregate performance quickly levels off. To get an idea of the bottleneck involved in this case, Figures 10a, 10b, and 10c show the CPU, memory, and I/O utilization on the target during the transfers. As can be seen, even at 32 source hosts, the destination CPU has a maximum utilization of only 46%. Memory utilization does hit 99% above 4 hosts, but I/O hits its maximum much earlier, hence memory is unlikely to be the bottleneck.

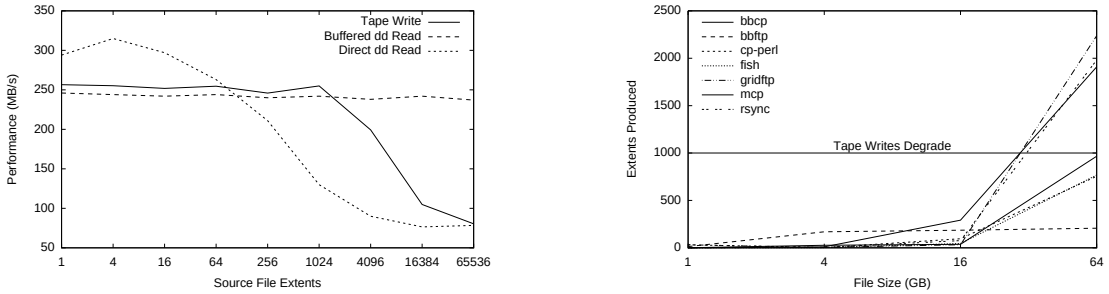
In the testing environment, the destination host is connected to the file system using 4x FDR Infiniband, which has a maximum bandwidth of 7 GB/s. As shown in Figure 10c, the maximum performance of a 32-way dd write was measured at 6.23 GB/s, which is significantly higher than the maximum write performance achieved of 4.35 GB/s. Nodes communicate with each other over a separate 4x FDR Infiniband link using IPoIB to support standard TCP/IP applications. Using iperf to measure the IPoIB performance, however, resulted in a maximum of 4.775 GB/s, which is only 10% more than the maximum write performance achieved. Hence, the limiting factor is likely IPoIB performance. Because Shift understands the file systems mounted on each host, it can map the remote host initially specified by the user into one or more equivalent hosts that have lower load. In the case of remote transfers, Shift also ensures that all clients are using different remote hosts when available to minimize single system bottlenecks.

# 5. FILE SYSTEM OPTIMIZATION

While transfer speed is immediately obvious to users as they are given the number directly in Shift's status output, other performance factors exist that may only show up later during computational jobs or when retrieving files from tape. In particular, different file systems have various idiosyncrasies that affect read/write performance, stability, and operating cost. By addressing these factors within Shift at the time of transfer, users will eventually see a performance gain even though they may not be aware of it.

## 5.1 Tape Optimization

In HPC environments, high end disk arrays store user data for use by thousands of computational nodes. These systems have very high performance but come at very high cost. Since scientific users are constantly creating new and




(a) Tape write/file read performance by extent

(b) Extents produced by transport and file size

Figure 11: Tape write speed decreases significantly beyond 1000 extents, which is easily reached when transferring larger files.

very large data sets or reprocessing old data sets with new techniques, and expanding online disk capacity beyond a certain point becomes cost prohibitive, hierarchical storage systems backed by tape libraries are used to archive older files in a more cost effective manner while still allowing them to be retrieved to disk when needed. While tape libraries provide much higher density at a fraction of the cost, their performance is highly constrained by slow-moving physical components such as robotic assemblies and tape cartridges. Because it is much more difficult to add additional robots than another bank of disk drives, it is imperative that tapes be read/written at the fastest speeds. One phenomenon observed at the NAS facility was the severe degradation of tape speed in some files copied by `mcp`, which uses concurrent threads to increase performance, but was found to sometimes create large numbers of file extents when doing so that were impacting tape write speed.

Figure 11a shows the tape write and file system read performance for varying numbers of file extents. Write measurements were obtained from a dedicated SGI CXFS file system to a dedicated Spectra Logic TFinity tape library with LTO-6 drives. File system read numbers were obtained using `dd` from the same file system with buffered and direct I/O. As can be seen, after 1024 extents, tape write performance drops significantly. On the file system side, buffered reads produce consistent results across all numbers of extents, but direct reads drop in very similar fashion to tape writes. While the measurements were not perfectly aligned, there was strong evidence that the cause of the degradation was the use of direct I/O in the tape subsystem, which was later confirmed by SGI.

To understand the potential for the problem to occur, Figure 11b shows the number of extents produced by different transports for varying file sizes. While the extents will vary depending on other activity and current free blocks on the file system, the figure shows that transports can easily cross the threshold at which tape write performance degrades with a fairly small file by archival standards (i.e. 64GB). To minimize the number of extents in transferred files to the degree possible, Shift supports preallocation of files, which will thereby maximize tape utilization during writes.

The downside of preallocation is that sparse files, whose size is greater than their disk utilization due to holes that are not stored on disk, are forced to store all bytes on disk. This can both surprise users whose disk quotas can rapidly change after a sparse file is made regular and increases disk usage unnecessarily, which is especially detrimental to tape-backed file systems that may not have as much online space available. To get the benefits of preallocation while minimizing its drawbacks, Shift supports a configurable sparsity threshold at which preallocation will not be performed, allowing sparse files to stay sparse while preallocating for the vastly more common regular files. Note that preallocation is not effective for `bbftp` due to its use of temporary files without the ability to overwrite existing files.

Once a file has been migrated to tape and removed from disk, it must be retrieved back to disk from tape before it can be transferred. In the case of SGI's Data Migration Facility (DMF), files will be retrieved automatically when accessed if they are not already online with the calling program blocked until the file is available. Letting DMF's automatic mechanism take care of retrieval during a transfer of multiple files is non-optimal, however, as the transfer may enter a cycle of retrievals and copies where DMF does not have a chance to minimize tape movement and/or may even unmount the tape by the time one file has been copied and the next retrieval begins. By manually initiating a retrieval of all files to be transferred, DMF is given a chance to retrieve files on the same tape in the most efficient manner.

Figure 12 shows the difference in retrieval time between batched and sequential retrievals of varying numbers of 1GB files. As can be seen, retrieval time is significantly greater when files are retrieved one after the other compared to when they are retrieved in a single batch. When Shift determines that a transfer is being initiated from a DMF file system, it automatically issues a request to retrieve all files in the transfer so that hopefully, most files will be online by the time it tries to copy them. Shift also reissues the retrieval of each batch of files it is about to transfer in case the files have been pushed back offline by the time it gets to them. While Shift currently only supports DMF (since that is the only tape system available to the author), the approach taken for optimizing tape retrievals is applicable to many such systems by using the equivalent to DMF's `dmget` command (e.g. in IBM's TSM and HPSS systems, the equivalents would be "`dsmc retrieve`" and "`hsi in`", respectively).

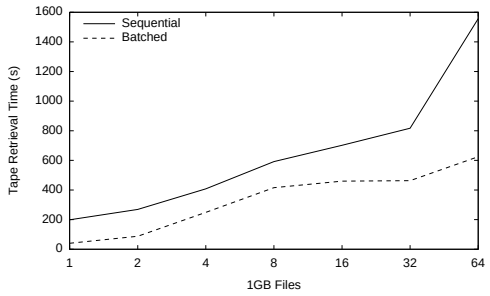



Figure 12: Batched tape retrievals are more efficient due to minimized tape movement.

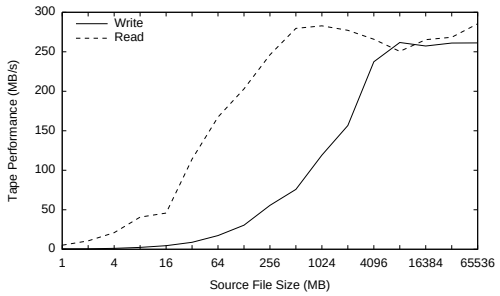
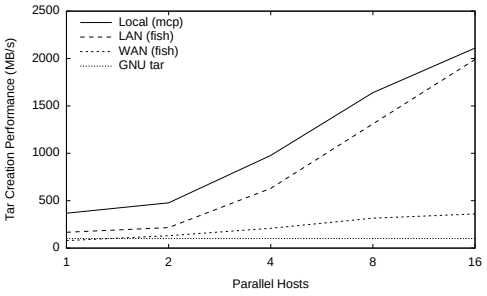
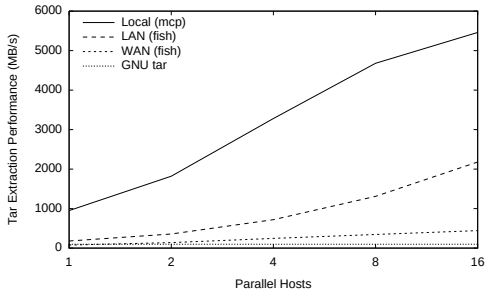



Figure 13: Tape I/O is more efficient with larger files where tapes can reach full speed.




(a) Tar creation performance




(b) Tar extraction performance

Figure 14: Shift can reach significantly higher speeds than GNU tar using high performance transports and parallelization.

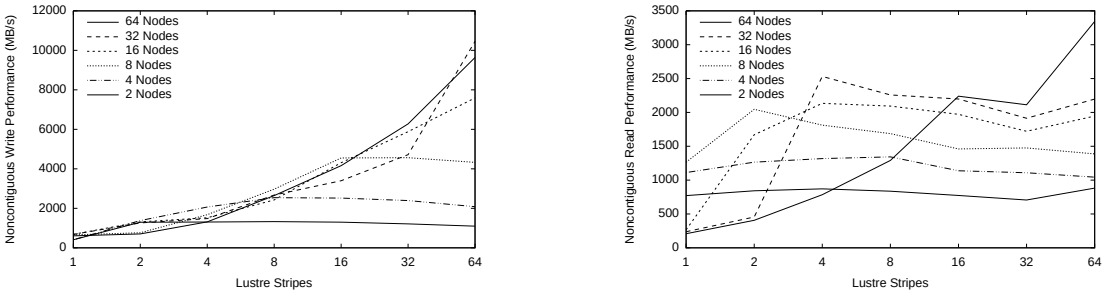
## 5.2 Tar Creation/Extraction

From the user's perspective, directory hierarchies from scratch space could ideally just be copied as is to tape-backed storage so every file could simply be accessed individually as needed. From a stability perspective, however, doing so is very detrimental to the file system. First, tape-backed storage typically has a size threshold below which files are kept on disk instead of being migrated to tape. So a directory structure that is large in aggregate size but that consists of many small files may eat up significant portions of the file system and never be migrated to tape. Even when the files are above the size threshold, however, it is still detrimental to the system when the files are not large enough to be efficiently written/read to/from tape.

Figure 13 shows tape read and write performance for files of varying size. As can be seen, read performance is only maximized for files above 512MB and write performance is only maximized for files above 8GB. The standard solution for this problem is to have users aggregate data into tar files with enforcement using quotas on the number of files to prevent huge directory copies. This is non-optimal for usability since both tar creations and extractions are extremely slow at around 100 MB/s. In addition, users may not know what a migrated archive contains unless they retrieve it from tape to list its contents, they have no assurance whether individual files were copied intact, and may have to retrieve huge archives when only a small number of files are needed from within. If scratch directories are not mounted on the archive system, users may also not have enough quota left to even create a tar file to free up space needed for other activities.

Shift has built-in support for tar creation and extraction without the need to use the tar utility itself, which allows all of the above issues to be addressed. Shift can utilize high performance transports in addition to its parallelization capabilities to copy data in and out of tar files at rates far beyond the tar command even on a single node. Shift can automatically create index files of tar contents that can be kept online due to their small size so users know exactly which archives contain which files. Shift can verify the integrity of files copied into or out of tar files and record checksums for later use to give users assurance that their long term archives are intact before deleting the originals. Shift can automatically break tar creations into multiple tar files of a given size, allowing users to retrieve smaller amounts of data from tape when only a subset of the files are needed. Finally, Shift can create and extract tar files directly over the network so users can always create archives regardless of their quota on the scratch file system.

Figures 14a and 14b show the tar creation and extraction performance to/from a Lustre file system, respectively, of Shift over different numbers of hosts locally as well as remotely across the LAN/WAN. On a single local host, Shift can achieve over 3.6 times the creation performance of GNU tar up to 21x on 16 hosts. Remote creation over the LAN achieves almost 1.7x on one host up to almost 20x on 16 hosts. Extraction is faster due to Lustre single file read performance being faster than single file write performance. Local extraction can be done at almost 10x GNU tar on one host to over 56x on 16 hosts. LAN extraction is at almost 2x on one host up to over 22x on 16 hosts. Tar operations




(a) Lustre striped write performance

(b) Lustre striped read performance

Figure 15: Lustre I/O rate increases with stripes up to a maximum, but too many stripes per reader can decrease read speed.

over the WAN are currently limited by SSH performance and only achieved 80% of GNU tar on one host up to 3.6x/4.4x for creation/extraction on 16 hosts.

## 5.3 Lustre Striping

Lustre is a high performance file system that is used by over 60% of the top 100 supercomputers in the world [26]. Lustre utilizes parallel striping across large numbers of disks to achieve higher aggregate performance than is possible on a single-disk file system. Unlike other parallel file systems such as CXFS and GPFS, however, Lustre striping must be specified explicitly before a file is first written. Striping can significantly impact I/O performance. A greater number of stripes provides more available bandwidth but also produces greater resource contention during metadata operations. A large file stored over a smaller number of stripes produces imbalances in the underlying Lustre OSTs. A file can only be restriped by copying it in its entirety, so it is critical that striping be set correctly before it is first written.

Figures 15a and 15b show the effects of striping on Lustre write and read performance, respectively, using the contiguous memory, non-contiguous file access case of the noncontig benchmark [19] across varying numbers of nodes and stripes using a file size equal to one GB per node. This benchmark simulates noncontiguous I/O access patterns that may be found in scientific applications. As can be seen, the higher the number of nodes, the higher the number of stripes needed to maximize write performance with little drop-off when using more stripes once the maximum is reached. Read performance also benefits from more stripes, but levels off much more quickly and increasing the number of stripes once the maximum rate is reached for a given number of readers actually decreases performance in many cases. Hence, it is beneficial to stripe files in proportion to the number of readers who are going to access it. A more detailed treatment of striping is addressed in previous work [10].

The benefits of striping in file transfers are twofold. First, using greater stripes for destination files increases write performance during parallel transfers. Second, later parallel jobs that read the files will achieve higher performance than they would at default striping, thereby allowing better utilization of computational resources. In previous work [10], stripe-awareness was added to a set of standard system tools so that files would be striped automatically as users performed day-to-day operations. With Shift's ability to parallelize transfers across many hosts with many different tools, a more general approach is used where destination files on Lustre file systems are created with the appropriate striping before being filled with data by the transport. Note that this technique is not effective for `bbftp` due to its use of temporary files without the ability to overwrite existing files. By default, Shift will restripe if the source file is not on Lustre or has default striping, and will preserve striping when the source is on Lustre with non-default striping. This allows users who have specifically striped files for highest application performance to retain striping settings.

# 6. CONCLUSIONS AND FUTURE WORK

This paper has presented the Shift automated transfer tool and the mechanisms it employs to achieve better performance while preserving the stability of HPC environments. Shift encapsulates best practices understood by domain experts during transfers so that scientists can focus on their science without the need to study file transports, resource management, and file systems as well. Shift understands how to utilize the variety of transports that might be deployed throughout a widely distributed user base, how to maximize the performance achievable by each, and the scenarios in which each is most effective. In particular, Shift automatically tunes the TCP window size and number of TCP streams used by `bbcp`, `bbftp`, and `gridftp`, automatically selects the highest performance SSH cipher and MAC algorithm available for `fish`, `rsync`, and `sftp`, and automatically selects the highest performing transport for batches of files based on average file size, transfer type, and availability.

Shift understands which resources are available in a particular HPC environment and how to utilize them for significant performance increases while preventing resource exhaustion. These include automatically parallelizing transfers on the same host and/or across multiple hosts to take advantage of excess resource capacity, automatically throttling transfers according to various administrator-defined local and/or global criteria to prevent system overload, and automatically load balancing transfers across parallel clients and remote hosts to eliminate single system bottlenecks.

Finally, Shift understands the file systems to which and from which files may be transferred and the nuances to their use that affect performance and stability behind the scenes.

Reads and writes of tape-backed file systems are optimized for maximum tape performance by automatically retrieving migrated files in batches and automatically preallocating files below a configured sparsity, a high speed tar creation and extraction capability increases usability by significantly reducing turnaround time of archive operations and allowing direct remote archival to overcome quota limitations, and files transferred to Lustre file systems are automatically striped according to size to maximize read performance in subsequent batch jobs. Together these 10 automatically encapsulated best practices help Shift achieve significantly higher performance than any of its supported transports on their own. Shift is available as open source software [25].

There are variety of directions for future research. One area of investigation within the NAS environment is using a small cluster of nodes dedicated to Shift transfers, which would allow automatic parallelization to be ramped up without bogging down front-end systems. Shift's remote tar capabilities are performed using partial transfers of files directly into or out of tars during creation and extraction, respectively, using differing source and destination offsets. Currently, only the built-in `fish` or `sftp-perl` transports support this capability, which are both limited by the speed of SSH. To overcome the SSH bottleneck and substantially improve remote tar performance, an attempt will be made to augment `mcp` with a remote transfer capability. Some transports, such as `bbcp` and `rsync`, have options to limit the bandwidth utilized to a specific rate during transfers. These options could be used by Shift during throttling as an alternative to idling for a specific period of time. This could potentially allow more consistent throughput without as much oscillation around the threshold as can occur in the current mechanism.

## 7. ACKNOWLEDGMENTS

Thanks to M. Cary, L. Cox, S. Emery, and A. Meyer for suggesting most of Shift's tape optimizations and providing assistance during tape benchmarking. Thanks to J. Chang and S. Cheung for valuable discussions about I/O patterns found in scientific applications. Finally, thanks to D. Talcott for clarifying various Infiniband performance characteristics.

## 8. 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] B. Allen, J. Bresnahan, L. Childers, et al.: Globus Online: Radical Simplification of Data Movement via SaaS. Preprint CI-PP-5-0611, Computation Institute, Univ. of Chicago, Jun. 2011.

[3] BbFTP. `http://doc.in2p3.fr/bbftp`.

[4] N. Desai, R. Bradshaw, A. Lusk, E. Lusk: MPI Cluster System Software. 11th European PVM/MPI Users' Group Meeting, Sept. 2004.

[5] Dcp. `https://github.com/hpc/dcp`.

[6] Fast Data Transfer. `http://monalisa.cern.ch/FDT`.

[7] Gleicher Enterprises: HPSS Tar Man Page. `http://www.mgleicher.us/GEL/htar/htar_man_page.html`.

[8] A. Hanushevsky, A. Trunov, L. Cottrell: Peer-to-Peer Computing for Secure High Performance Data Copying. 12th Intl. Conf. on Computing in High Energy Nuclear Physics, Sept. 2001.

[9] P.Z. Kolano: High Performance Reliable File Transfers Using Automatic Many-to-Many Parallelization. 5th Wkshp. on Resiliency in High Performance Computing, Aug. 2012.

[10] P.Z. Kolano: Transparent Optimization of Parallel File System I/O via Standard System Tool Enhancement. 2nd Intl. Wkshp. on High Performance Data Intensive Computing, May 2013.

[11] 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.

[12] 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.

[13] 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.

[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] K. Matney, S. Canon, S. Oral: A First Look at Scalable I/O in Linux Commands. 9th LCI Intl. Conf. on High-Performance Clustered Computing, Apr. 2008.

[17] K.D. Matney, G. Shipman: Parallelism in System Tools. 52nd Cray User Group Conf., May 2010.

[18] E. Ong, E. Lusk, W. Gropp: Scalable Unix Commands for Parallel Processors: A High-Performance Implementation. 8th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, Sept. 2001.

[19] Parallel I/O Benchmarking Consortium. `http://www.mcs.anl.gov/research/projects/pio-benchmark`.

[20] Parsync. `http://moo.nac.uci.edu/~hjm/parsync`.

[21] Pcp. `https://github.com/wtsi-ssg/pcp`.

[22] R.S. Prasad, M. Jain, C. Dovrolis: Socket Buffer Auto-Sizing for High-Performance Data Transfers. Jour. of Grid Computing, vol. 1, no. 4, 2003.

[23] C. Rapier, B. Bennett: High Speed Bulk Data Transfer Using the SSH Protocol. 15th ACM Mardi Gras Conf., Feb. 2008.

[24] Rsync. `http://samba.org/rsync`.

[25] Shift. `http://shiftc.sourceforge.net`.

[26] S. Simms: Choose Lustre. Lustre User Group 2015 Conf., Apr. 2015.

[27] S. Thulasidasan, W. Feng, M.K. Gardner: Optimizing GridFTP through Dynamic Right-Sizing. 12th IEEE Symp. on High Performance Distributed Computing, Jun. 2003.

[28] E. Yildirim, D. Yin, T. Kosar: Balancing TCP Buffer vs. Parallel Streams in Application Level Throughput Optimization. 2nd Intl. Wkshp. on Data-Aware Distributed Computing, Jun. 2009.