# PSI – A High Performance File Transfer User Interface

Mark A. Roschke
*High Performance Computing Division,*
*Los Alamos National Laboratory*
*mar@lanl.gov*

C. David Sherrill
*High Performance Computing Division,*
*Los Alamos National Laboratory*
*dsherril@lanl.gov*

## Abstract

*Transferring and maintaining large datasets requires parallel processing of both data and metadata for timely execution. This paper describes the work in progress to use various processing techniques, including multi-threading of data and metadata operations, distributed processing, aggregation, and conditional processing to achieve increased transfer performance for large datasets, as well as increased rates for metadata queries and updates.*

## 1. Introduction

Ever-increasing computing capabilities result in ever-increasing data sets to be transferred. Such data sets can consist primarily of large files, many small files, or both. Transferring data sets with large files requires an emphasis on parallel file transfer, utilizing as much bandwidth as possible. And it is in this area that the majority of data parallel transfer development has occurred. But, it is no longer rare for a user to generate data sets of 100,000 to one million files. And when data sets reach this size, it is imperative that support be provided for high performance metadata operations, not only in support of file transfer, but also to support browsing and maintaining the data set.

## 2. Overview of PSI

The Parallel Storage Interface (PSI) is a data transfer user interface designed to provide high speed transfer for large data sets, with a special emphasis on utilizing as many resources as possible for a single user request. Developed by the authors, PSI is the main user interface to the High Performance Storage System (HPSS) at Los Alamos National Laboratory. This paper describes the efforts to provide a full-featured data transfers capability for archival transfer, local transfer, and wide area host-to-host transfer, providing both high-speed data transfer as well as high-speed metadata processing.

PSI uses a parallel workflow model for processing both data and metadata. Work is parallelized and scheduled on available server and client resources automatically, using a priority and resource-based approach. Optimization is performed automatically, including areas such as parallelization, optimized tape transfer, load leveling, etc.

## 3. Unix syntax and Semantics

PSI utilizes UNIX-like syntax and semantics. For example, the following commands are available for data transfer and manipulation of file attributes: **cd**, **chmod**, **chgrp**, **cp**, **du**, **find**, **grep**, **ls**, **mkdir**, **mv**, **rm**, **rmdir**, and **scp**.

## 4. Multi-mode Operation

PSI offers three modes of operation, providing the same syntax, semantics, and look and feel for the three most frequently used data transfer situations, which are 1) local transfer, 2) archival transfer, and 3) host-to-host transfer. The particular interface command determines the context of the specified commands. For example,

| | | |
|---|---|---|
| **sh** | **cp –R a b** | copy files locally |
| **psiloc** | **cp –R a b** | parallel copy locally |
| **psi2ccc** | **cp –R a b** | parallel copy on cluster **ccc** |
| **psi** | **cp –R a b** | parallel copy in the archive |

This approach provides a consistent look and feel, allowing the user to move between the 3 major transfer situations, eliminating the time necessary to learn the command set for each situation.

## 5. Automatic Optimization

The general design approach for PSI is that the user simply specifies the files to be operated on, PSI determines the resources available to the command, and then executes the command, with all optimization being performed automatically, including such features as adjusting all types of thread counts dynamically, optimizing the order of any data transfers to/from tape, assignment of operations across multiple hosts (including load leveling), and splitting large transfers across hosts when appropriate.

To support automatic optimization, all activity within PSI is controlled using a priority-based resource management scheme, limiting the amount of bandwidth and memory that each type of activity can consume. Scheduling of activities such as file transfers are performed via an internal job scheduler, which dispatches activities across available hosts in an optimal order, load leveling all activities as necessary.

## 6. Conditional Transfers

To address occurrences of interrupted transfers as well as that of newly arriving (or updated) data within a data set, PSI can scan both the input tree and output tree, examining file attributes to determine which files need to be transferred. This feature alone can routinely save hours of time that would be spent on re-transferring the entire tree.

## 7. Parallel Archival Tarring Option

When transferring to the archive, the user can select the PSI tar option, which automatically utilizes parallel tar transfers to/from the archive. The parallel tar capability in PSI typically constructs one or more tar files per directory, preserving the original tree structure. Large files are normally transferred un-tarred to the archive. Multiple tar processes are load leveled across available hosts, providing scalable multi-host performance, even for small files.

The archive namespace is extended into the tar files present, utilizing the index file that is stored with each tar file. This namespace extension prevents the archive from becoming a large black box of data. The user can browse through the original tree, and execute such commands as **ls**, **find**, **grep**, **rm**, and **scp** with references to files within the tar files, and can also utilize globbing (i.e. wild cards) in such references. Conditional transfers are also supported, so that newly arrived files can be placed within new tar files in a directory. In addition, commands such as **scp**, **chmod**, **grep**, **ls**, and **rm** are specially aware of the tar files, and can take advantage of operating on whole tar files when feasible.

## 8. Techniques to Increase Performance

The general approach chosen involves the use of parallel data and metadata processing, automatic optimized file aggregation and de-aggregation, and conditional operations when feasible. Combining these three features provides a variety of performance increases. For example, multi-threading to a degree of 40 threads might increase performance by a factor of 30, while operating on a file aggregate of 1000 files can provide a performance boost of up to 300. Conditional operations can provide a factor of 20 or so. By combining these three features, performance gains of over 1,000 have been observed, as outlined below.

## 9. Multiphase Parallel Work Flow

To facilitate efficient control of the various steps required to execute user requests in a parallel fashion, For example, tasks are organized into phases, e.g. 1) stat source files, 2) stat destination files, 3) transfer files. Work progresses though each phase. Each phase can consist of many threads, each requiring different resources. Achieving high performance in processing metadata requires a reasonably high degree of parallelism; typical thread counts for all three phases is 100 to 200, depending upon the mix of metadata and file transfer operations being performed.

## 10. Areas of Performance Increase

Work at increasing performance has fallen in two general areas – increasing parallelism, and decreasing latency with the latter area receiving the most effort. Increasing parallelism generally falls in the predictable categories of more threads, and more hosts,
with some miscellaneous optimization applied to areas such as when to transfer large files across nodes, etc.

Effort to decrease latency has been largely in the area of various types of aggregation, namely 1) data aggregation, 2) control aggregates, 3) metadata query aggregates, and 4) metadata update aggregates.

Metadata query work has involved experiments with striping directory queries across multiple hosts, with an eye toward support of massive directories (directories with greater than 50,000 files).

Since aggregation is largely connected with latency, the benefits from aggregation tend to be shared across the areas of faster scheduling, more scalability, and faster WAN operations.

## 11. Conclusion

Combining the techniques of multi-threaded processing of data and metadata with the concept of small file aggregation can result in significant performance increases. These increases can be further improved by adding techniques such as conditional updates or conditional file transfers. Performance increases above factors of 1000 have been observed. In addition, using user-generated aggregates can result in significant decreases in archival system metadata.

## 12. Performance Results

The following results were obtained on a cluster of 4 client nodes connected to to a Panasas file system,
For various files sizes and commands.

**Local Mode**

| **Mode** | **cp 1KB (files/s)** | **cp 10MB (MB/s)** | **cp 1GB (MB/s)** | **conditional transfer (files/s)** | **chmod (Files/s)** | **find (Files/s)** | **rm (Files/s)** |
|---|---|---|---|---|---|---|---|
| sh | 32 | 15 | 55 | - | 47 | 312 | 247 |
| psi (local) | 1,813 | 398 | 431 | 2364 | 2,090 | 1,743 | 2,999 |

Also, in a recent large scale test on 16 nodes connected to a Panasas file system, 295 TB of data (consisting of 967,000 files) were copied at an average rate of 2.85 GB/sec.

The following results were obtained on a cluster of 4 nodes, from a Panasas file system to Los Alamos HPSS.

**Archive Mode (HPSS)**

| **Mode** | **cp 1KB (Files/s)** | **cp 10MB (MB/s)** | **cp 1GB (MB/s)** | **cond (files/s)** | **chmod (Files/s)** | **find (Files/s)** | **rm (Files/s)** |
|---|---|---|---|---|---|---|---|
| psi (HPSS) | 80 | 1,071 | 580 | 102 | 295 | 599 | 155 |
| psi (HPSS, TAR) | 1,139 | 256 | 269 | 2,365 | 31,188 | 2,999 | 15,210 |

The following results were obtained on a cluster of 4 nodes, from a local Panasas file system to a remote Lustre file system, with a round trip time of 38 ms (Los Alamos, NM to Livermore, CA)

**Host-to-Host Mode**

| **Mode** | **cp 1KB (Files/s)** | **cp 10MB (MB/s)** | **cp 1GB (MB/s)** | **cond (files/s)** | **chmod (Files/s)** | **find (Files/s)** | **rm (Files/s)** |
|---|---|---|---|---|---|---|---|
| psi (h2h) | 1,933 | 423 | 480 | 2,396 | 2,433 | 3,012 | 229 |