LA-UR- 08-7314

Approved for public release;
distribution is unlimited.

Title:	Parallel Log Structured File System (PLFS)
Author(s):	Gary Grider John Bent James Nunez
Intended for:	PDSW Meeting, Austin, TX, November 17, 2008

Los Alamos
NATIONAL LABORATORY
EST. 1943

Form 836 (7/06)

Parallel Log Structured File System (PLFS)

Gary Grider, HPC-DO

Abstract

Currently and in the foreseeable future, the way that large HPC applications cope with interrupts of the application is checkpointing of state of the application to stable storage. This is typically done by writing application state to a global parallel file system. Given the anticipated reliability decrease in future HPC systems, it becomes vital that this checkpointing action be done rapidly to keep the computation to checkpoint time ratio high. Checkpointing has the property of being a write mostly application. Typically, many more checkpoints are created/written than actually ever read since this is a defensive I/O mechanism. Checkpointing typically takes one of three forms

- Each process writes its own state to its own file: N processes to N files (N to N)
- Multiple processes write data into several files: N processes to M files (N to M)
- All processes write data into one single file: N processes to 1 file (N to 1)

The N to 1 method for checkpointing has many advantages for users of HPC applications but is most difficult for HPC parallel file systems. The N to N method for checkpointing is easiest on HPC parallel file systems but most inconvenient for HPC application users. The PLFS work in this paper is an attempt to take the best properties of N to N I/O and the best properties of N to 1 I/O and make them available to users/applications in a completely transparent way to the applications (requiring no modifications/re-linking etc.) applications that use the N to 1 dump method.

Parallel Log Structured File System (PLFS)

Gary Grider, John Bent, James Nunez
{ggrider, jnunez, johnbent @ lanl.gov}
Los Alamos National Laboratory (LANL)

Work supported by:
DOE/NNSA ASC Program
DOE/Office of Science SciDAC Petascale Data Storage Institute
LANL/CMU Institute for Reliable High Performance Information Technology (IRHPIT)

11/05/2008

Background/Motivation

The need for increasing scale in scientific computation drives the need for rapidly increasing scale in storage capability for scientific processing. Individual storage devices are rapidly getting denser while their bandwidth and agility is not growing at the same pace.

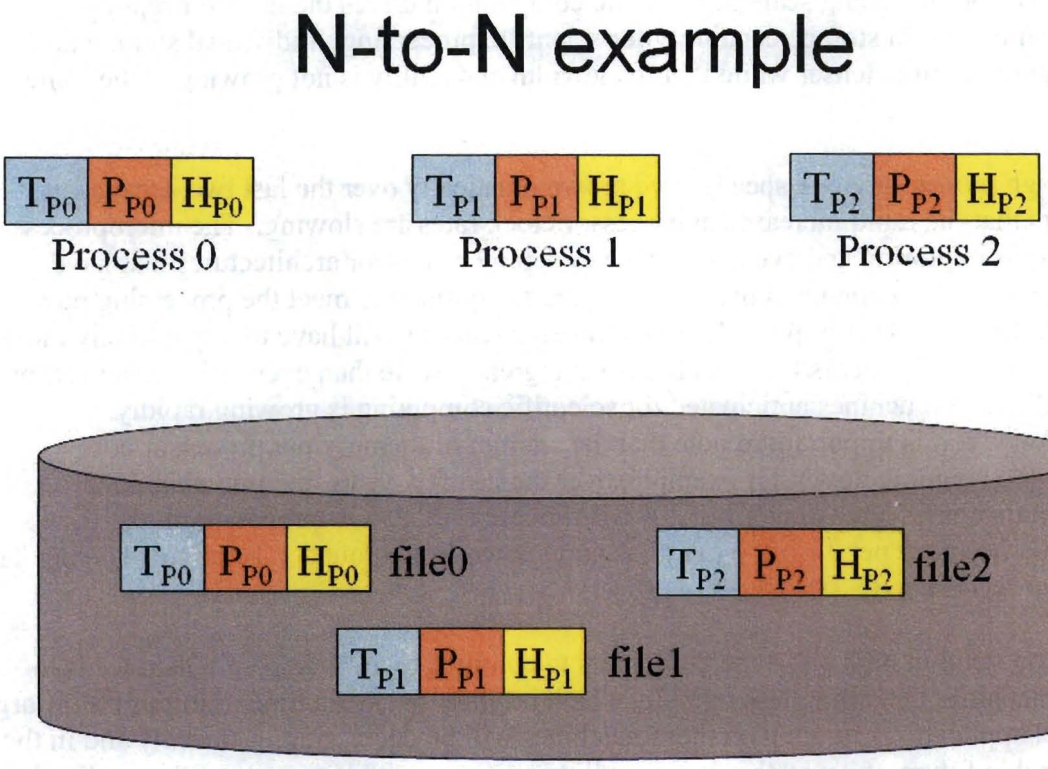
Although processor clock speeds have grown drastically over the last two decades, it appears that the rapid increases in processor clock rates are slowing. The microprocessor industry is exploring and even beginning to deploy processor architectures that have many more processing units per chip or board to continue to meet the processing power growth demand. This implies that scientific applications will have to begin to rely more heavily on multi-process/task parallelism at a greater scale than ever before. The compute capabilities of machines anticipated for scientific computing is growing rapidly. Additionally, it is important to note that the amount of memory per processor/core appears to be going down, for example, over the last few years, memories in ASC progam machines have gone from 2-4 GB/processor to .25 GB/processor on the Blue Gene/L. Memory per Teraflop (TF) has gone down from around 1 Terabyte (TB) per TF to about .1 TB per TF [HECFSIO-2007].

Also, the trend in high end supercomputers to continue to grow in parallelism leads to these machines becoming less reliable, which implies that mean time to interrupt for large parallel applications running on these machines will be decreasing. Currently and in the foreseeable future, the way that large applications cope with interrupts of the application is checkpointing of state of the application to stable storage. This is typically done by writing application state to a global parallel file system. Given the anticipated reliability decrease, it becomes vital that this checkpointing action be done rapidly to keep the computation to checkpoint time ratio high. Checkpointing has the property of being a write mostly application. Typically, many more checkpoints are created/written than actually ever read since this is a defensive I/O mechanism. Checkpointing typically takes one of three forms [Nunez-ISW2008].

- Each process writes its own state to its own file: N processes to N files (N to N)
- Multiple processes write data into several files: N processes to M files (N to M)

- All processes write data into one single file: N processes to 1 file (N to 1)

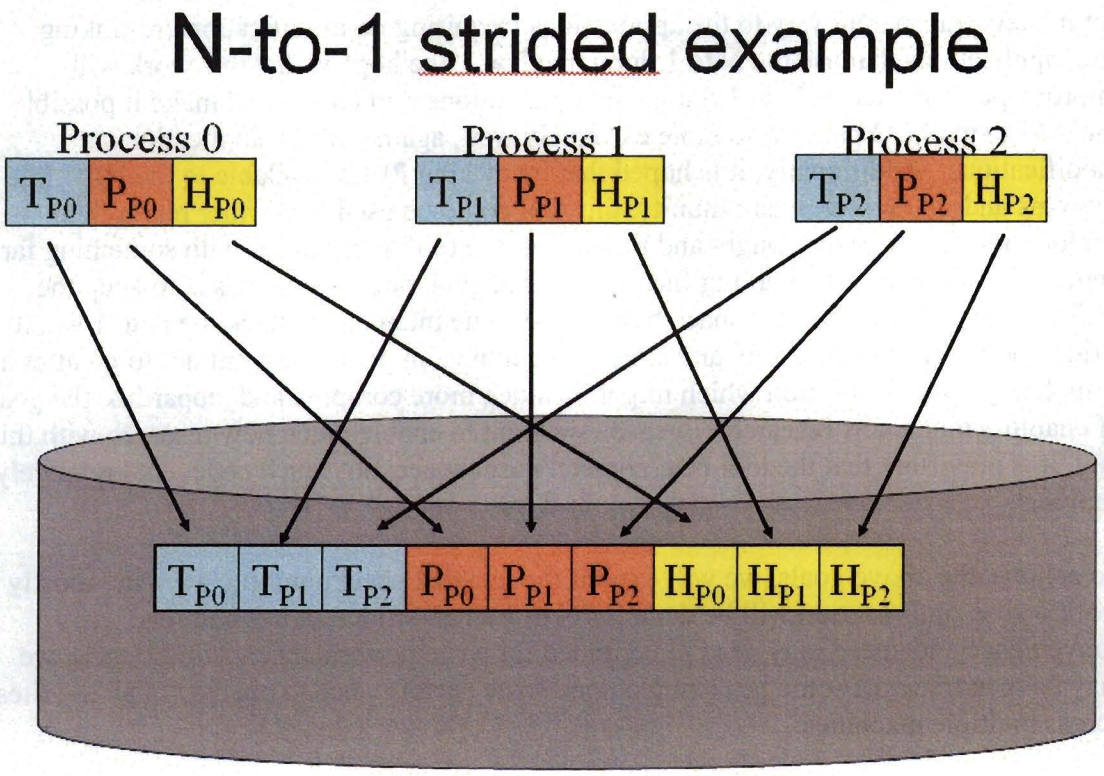
N to N:
While N to N offers the highest concurrency for writing data which is the easiest thing for file systems to handle, it also stresses the metadata portion of file systems for concurrent inserting of file entries. As the number of processes grows to utilize the large number of processor/cores in future supercomputers, the N to N checkpoint method becomes difficult to manage as the number of files for the user to manage gets enormous. Take a future Blue Gene machine with 1 million processes taking checkpoints every few hours as an example. N to N checkpointing in this case would generate millions of files per day and billions of files for a significant scientific study. Additionally, N to N makes restart of a job from checkpoint files trivial for restarting on the same number of nodes as generated the checkpoint, but restarting from differing number of processes is quite complex. In the following example, each process has 3 arrays data. Each process writes all of its data to a separate file.



N to M:
N to M checkpointing has some nice properties in that it reduces the metadata concurrence workload for the file system but it also reduces the I/O concurrency which is bad for performance. It also can be quite complex for restarting on a different number of nodes.

N to 1:

N to 1 checkpointing offers the best situation for metadata inserts, is easiest for the user to cope with file management, can simplify restart on different number of nodes, but suffers from the biggest issues for I/O concurrency. Additionally, as memory per process goes down, if data is interleaved from each process into the file to further assist restart on different number of processes operations, the concurrency creates hot spots in the single file. In the following example, each process has 3 arrays of data, just as in the N to N example. In this N to 1 strided example, the first arrays are co-located together with all processes writing to a single file.



Additionally, high level scientific data management libraries such as the Hierarchical Data Format (HDF) [HDF] and Network Common Data Format (NetCDF) [NETCDF] use the N to N method. These libraries help scientific users document the structures in the data. These libraries use the N to 1 model to make the users life easier.

Also, the MPI-IO parallel I/O library has immense power to describe complex derived parallel/distributed data types and complex views of the data in the data files. Often, N to 1 is the file outcome from users of this powerful capability [THAKUR-1999].

Initial Goals for the PLFS

As you can see, the N to 1 method for checkpointing has many advantages. The major disadvantage to the N to 1 method is the issues with concurrent writing of data into a single file and the difficulties this presents to modern global parallel file systems. All parallel file systems have a difficult time with this I/O pattern. Most of these parallel file systems have special modes for handling the N to 1 small strided I/O patterns which are seldom used properly by users/applications. A great deal of work has also been done in middleware like MPI-IO to help file systems deal with this N to 1 small strided I/O pattern. The PLFS work is an attempt to take the best properties of N to N I/O and the best properties of N to 1 I/O and make them available to users/applications in a completely transparent way to the applications (requiring no modifications/re-linking etc.) applications that use the N to 1 dump method. The hope is that this work will improve performance of N to 1 dumps for applications immensely and make it possible for N to 1 small I/O patterns to scale extremely well, again with no application modifications. Additionally, it is hoped that by making PLFS available to the HEC File Systems and I/O research community, this tool could be used for further research to explore new frontiers of thought and to improve the tool or replace it with something far better. To facilitate this enabling further research goal, one of the goals is to keep the PLFS simple, small, and user space based code. The intent is to attack the N to 1 small strided problem fundamentally and simply to get huge improvement but not to go after a completely optimal solution which might be much more complex and jeopardize the goal of enabling much new research. Because we want to enable much new research with this tool, it is important that the tool be completely user space, not much code, and relatively simple.

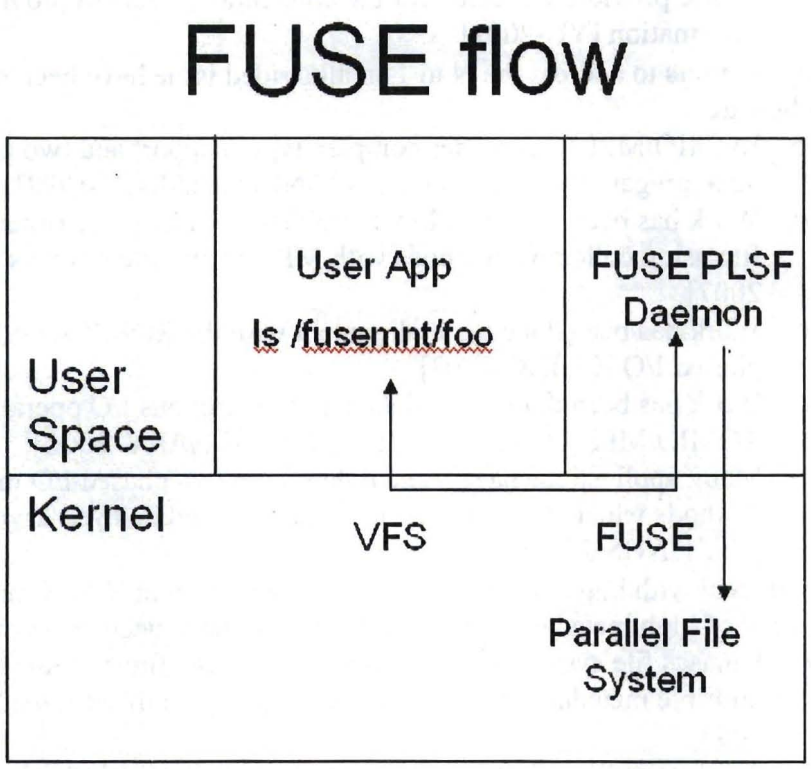
To achieve the above goals, we will exploit the fact that checkpointing is a write mostly workload so optimization will be done for N to 1 small strided writes and read convenience and speed may need to be traded for write performance. Additionally, we may have to trade off some peak single node write performance to enable scalable writes across multiple machines.

Some Related work

There has been much work in the area of dealing with large parallel checkpointing for HEC, so much work that it would not be practical to mention all of the work. For this reason we have chosen to mention a few relatively recent works organized in groups of methods of attack.

- Global Parallel File System vendors have tried to help with the N to 1 small strided concurrence problem by offering special modes of operation that users of these systems can request. These include
  - Panasas provides parallel RAID10 and other layout control operations which allow for more concurrency on N to 1 patterns and concurrent write open flags which require the user to promise to not overlap I/Os which allow for better concurrency [BENT-2008] [WELCH-2008].
  - GPFS provides a method for applications to provide layout information to the file system for optimizations [ANTYPAS-2007] [SCHMUCK-2002].
  - Lustre provides a method for the applications/users to provide layout information [YU-2007].
- Many attempts to address the N to 1 small strided issue have been done via middleware
  - ROMIO/MPI-IO provides complex types support and two phased I/O to do aggregation of I/O operations (ANL) [THAKUR-1998].
  - Work has been done to allow for assisting with open storms on a single file using collective methods with MPI-IO and file systems [LATHAM-2007]
  - Work has been done to do alignment with the ROMIO/MPI-IO two phased I/O [CHING-2003]
  - Work has been done to do delayed/asynchronous I/O operations with ROMIO/MPI-IO two phased I/O [CHOUDHARY-2006]]
  - Many applications have written their own two phased I/O middlewahre methods which do aggregation, alignment, and delayed/asynchronous I/O [GITTINGS-2008]
- To help deal with huge numbers of files typically seen in N to N dump operations a number of high metadata concurrence schemes have been devised
  - Panasas file system offers the ability to spread file system metadata over multiple metadata servers to increase aggregate insert rates [WELCH-2008]
  - The Lustre file system team designed a directory split/hash scheme to split up directory entries across multiple metadata servers, this capability is not yet available in general release however [LUSTRE-LITE]
  - The GPFS file system offers directory splitting across multiple SAN volumes [SCHMUCK-2002]
  - The PVFS file system offers splitting of metadata operations across multiple servers [PVFS]
  - CMU has recently done work in using the file system in user space FUSE software to do directory splitting/hashing [GIGAPLUS]

- Log Structured File Systems have been used in the past to cope with high seek workloads
    - The work by Ousterhout and Douglas on defining log structured file systems to deal with high write seek workloads is of course legendary and has become the standard way to optimize for random write workloads. [ROSENBLUM-1990] [ROSENBLUM-1992]
- The file system in user space (FUSE) is a loadable kernel module for Unix-like computer operating systems, that allows a user space daemon to provide file system functions. The following diagram shows how the FUSE software transfers file system requests from user applications to the FUSE daemon, in our case PLFS and the FUSE daemon interacts with the real file system. This enables the FUSE daemon to act on behalf of the user app to restructure its file system requests. [FUSE]. The following diagram describes the request flow using FUSE to front-end a file system.

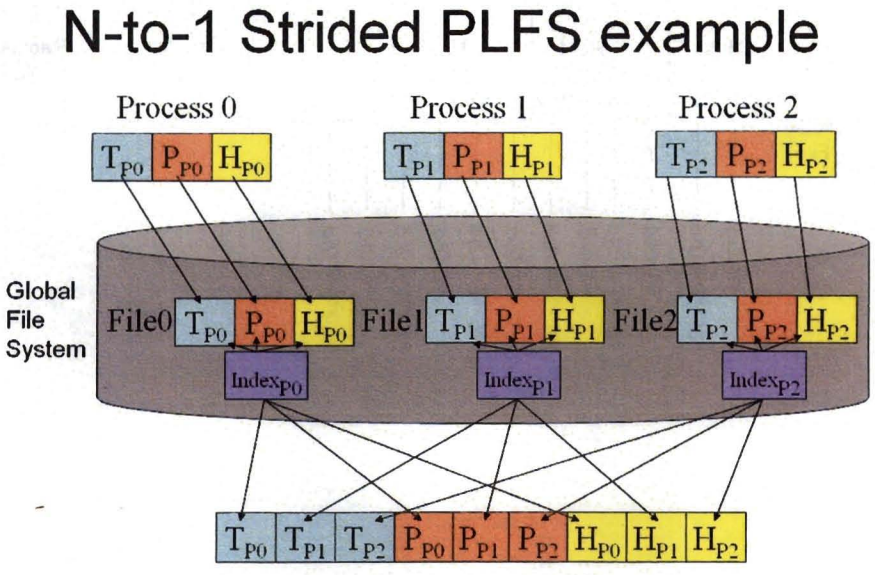


PLFS Design

To achieve the goals stated above for the PLFS, FUSE was chosen as the technology that underlies the PLFS. FUSE allows all the PLFS code to reside in user space and be quite small and simple. FUSE is an amazing gem in the open source community which allows for user space file system development to be quite simple. Additionally, in recent years, FUSE has become quite efficient on modern Linux machines.

The N to N checkpointing method benefits from the fact that each process has its own file. There are many benefits that are exploited because of this file per process scheme, high concurrency with little to no locking/consistency issues in the parallel process setting and aggressive caching because of no consistency issues. The high concurrence enables extremely good scaling of write performance. Additionally, the aggressive caching enables latency hiding of application I/O. An even further benefit is that most N to N applications write their own file serially or nearly serially which even further assists the aggressive caching and minimizes file system seeking. Of course the drawback to this N to N pattern is the large number of files created making metadata workload high and making it difficult to manage the files over time.

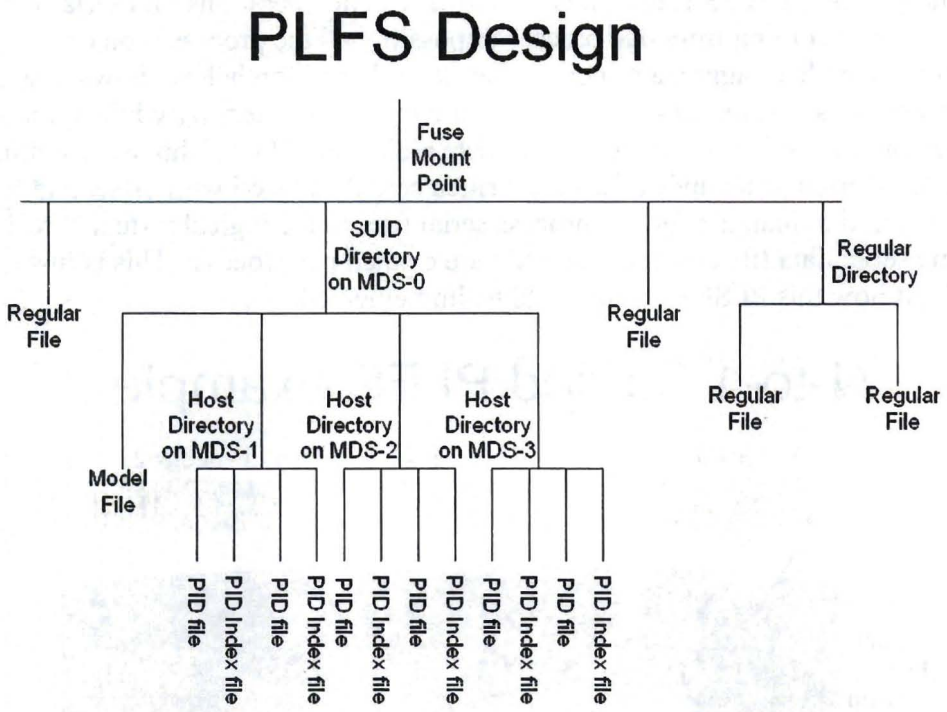
The concept behind PLFS is to turn N to 1 strided write operations into serial write operations only coming from one process or possibly all the processes on one host/compute node or aggregator node. The example cartoon below shows how each process writes its various arrays of data to a file per process serially while at the same time maintaining an index of where each write really should be if this were writing to a single file. Currently the index files are written serially as well with offset and length values required to map the file per process serial files to the logical virtual N to 1 strided file. One serial data file and one index file are created per process. This is just one example of how this PLSF concept could be implemented.



Virtual N to 1 Strided File

Notice in the diagram below that the process data/index files are grouped into a directory per host. This is done to put some structure over the large number of files that will be created. Notice that each directory of a subset of process data/index files is placed on a different metadata server. This is done to enable scaling of metadata operations within this single logical file. This again is one example of an implementation of a method to overcome the bad parts of the N to N workload that this conversion of N to 1 method to serial N to N method has created.

Additionally, notice the model file. This file is a surrogate file for the virtual file that is being represented by this directory structure. This is a place where extended attributes can be stored for the logical/virtual N to 1 strided file. Also notice that the top level directory of this virtual N to 1 file is a SUID directory. This is a directory with the POSIX SUID bit turned on. SUID is currently undefined for directories so this is the method used to tell the difference between a real file, a directory and a virtual N to 1 file. That SUID directory entry represents the logical N to 1 file in the name space of the real file system. Notice that in the highest level directory there can be many entries and each entry can be a normal file, a directory, or an SUID directory with the name of the logical file the user knows the file by.



Current Prototype Implementation

There is currently a prototype implementation of this PLFS we call FUSEPLFS. We use the FUSE capability to hide this SUID directory from the user. The SUID directory appears to the user as a regular file. Write operations occur to the logical file as described in the design section where each process writes serially to a file per process and each process also writes to an index file. The current prototype implements a subdirectory per host just as described above. Metadata operations like getattr to fulfill stat() calls uses the index files to determine the length of the file. Other metadata operations like chown(), chmod(), utime(), etc. use the model file to represent the logical file the user sees. There is an experimental read method being developed to use the index files to look up where read requests should be directed to.

Currently the prototype will not tolerate concurrent writes and read operations on a single logical file. It also will not handle append file operations. Additionally, the current prototype requires that the storage under the FUSEPLFS file system be at least a global file system.

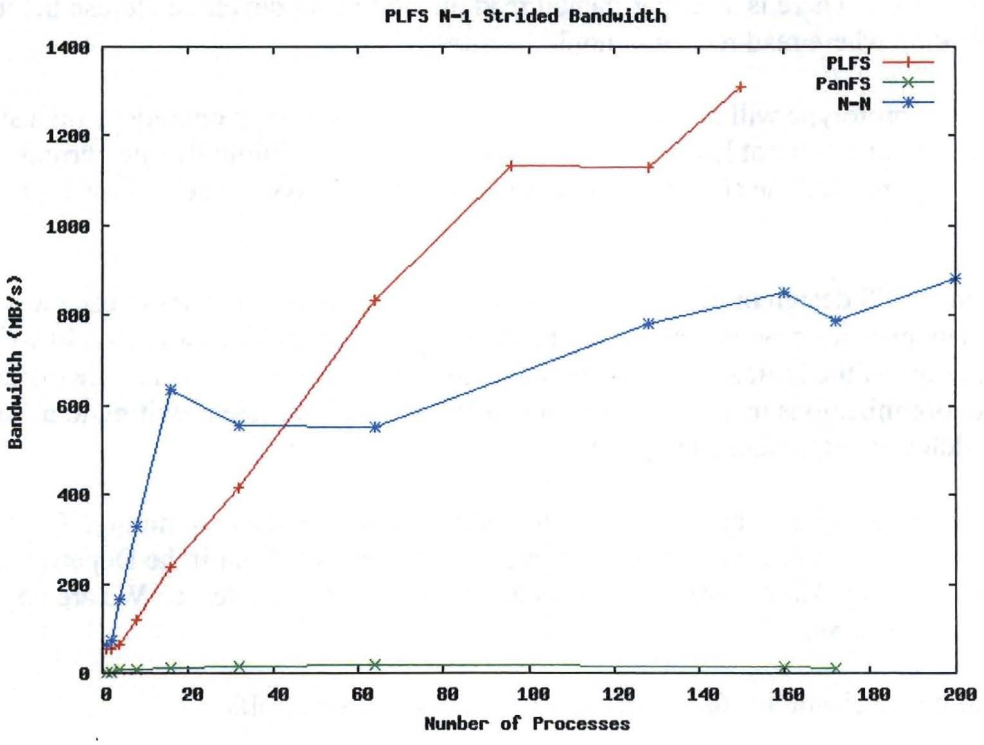
The current PLFS design implies that the users will make some promises to use the facility. The primary promise that has to be made by the user is that they will not write the same range of the logical file with more than one process ever. This is little different than other optimizations that have been done in the area of concurrent writing to a single file in middleware and other file systems.

There are plans to release this PLFS software into the open source community. Currently the code can be given out selectively with appropriate paperwork until the Department of Energy and the Los Alamos National Laboratory approve public release. We are hoping that will be very soon.

There will be a web site for this work at http://institute.lanl.gov/plfs

Initial Results

Some simple testing of the PLFS prototype have been done. The Los Alamos MPI-IO test [LANL-MPIIOTEST] was used to do synthetic workload performance measurements for N to 1 strided writing and reading. A small 128 node very old Xeon class cluster with gigabit Ethernet interconnect was used for the tests. The storage use in this test was 12 very old Panasas storage shelves attached to the cluster using gigabit Ethernet. The total capability of the storage system/storage area network combination was something less than 2 gigabytes/sec peak. The graph below shows the difference between PLFS on a small strided N to 1 workload. The PanFS line is the same workload without PLFS. The N-N line is similar write sizes but using an N to N pattern.



You can see that with a synthetic N to 1 small strided workload, PLFS performance scales nicely as you would expect.

Much more analysis of the PLFS is needed. Please see future work for planned follow on activities.

Possible Future Work

As was stated earlier, PLFS is a prototype currently. There are certainly many avenues of R&D that can be followed using the PLFS.

The preliminary nature of results leaves much room for improvement, adding more file systems, cluster systems, scale, and parameter sweeps in the results could further motivate investment in the PLFS. Additionally, trying more real applications especially those that use popular but problematic N to 1 small strided methods like those that use HDF and NetCDF as well as MPI-IO collectives.

Currently, the PLFS uses the FUSE direct-io mode to allow for large writes. In the near future, the FUSE package will allow large writes without using the direct-io mode. This may open up the ability for the PLFS to perform even better for smaller strided writes.

Of course, given the prototype nature of the PLFS, there is much room for hardening of the code as well as improving the very experimental read method. Support for concurrent read/write operations and append could also be added. Additionally, it is quite possible that one file per host and one index per host might perform reasonably well and have produce even less files. Various PLFS layouts could be explored with various workloads.

It is unclear if reconstruction of a real N to 1 strided file in an offline fashion before the file is needed for read is necessary or desired. For some read patterns, the indexed PLFS format might serve read requests adequately. A study as to the appropriateness of rebuilding the real N to 1 strided file, versus building an efficient single index from the distributed index, versus just leaving the file in PLFS format would certainly need to be done to determine how to best handle differing read workloads against PLFS files.

The indexing scheme used by the prototype PLFS is extremely simplistic. Study of indexing schemes in memory during writing, offline reconstruction of indexes, honoring read requests during a write operation, index types to take advantage of popular N to 1 strided patterns, different index distribution schemes etc. are all excellent follow on work. Additionally, the PLFS concept could also be merged with ideas like Gigaplus making N to N operations. This entire

The entire PLFS concept starts to head HPC file system storage towards file formats in the file system. It is quite possible that other file types besides N to 1 strided might be served well by similar thinking to PLFS. Decades ago, file types and access methods were used and were supported within a single file system. The IBM MVS storage systems allowed for many different file types, partitioned data sets, indexed sequential, virtual sequential, and sequential to name a few. Storage for modern HPC systems may benefit from a new parallel/scalable version of file types. There is much research to be done in this area to determine the usefulness of this concept and how such a thing would work with modern supercomputers and future HPC languages and operating environments.

References

[HECFSIO-2007] High End Computing File Systems and I/O 2007 Workshop Report, http://institute.lanl.gov/hec-fsio/docs/HECIWG-FSIO-FY07-Workshop-Document-FINAL-FINAL.pdf

[Nunez-ISW2008] LANL Data Release and I/O Forwarding Scalable Layer (IOFSL) Background and Plans, James Nunez, Gary Grider, John Bent, ISW 2007 presentation, http://www.dtc.umn.edu/disc/resources/nunez_isw2008.pdf

[HDF] Hierarchical Data Formats, HDF Group, http://www.hdfgroup.org/HDF5/

[NETCDF] Network Common Data Format, http://www.unidata.ucar.edu/software/netcdf/

[THAKUR-1999] Data Sieving and Collective I/O in ROMIO, Rajeev Thakur, William Gropp, Ewing Lusk, Proc. of the 7th Symposium on the Frontiers of Massively Parallel Computation, February 1999, pp. 182–189. c1999 IEEE, http://209.85.173.104/search?q=cache:Evux653LecAJ:www.mcs.anl.gov/~thakur/papers/romio-coll.ps+romio+two+phased&hl=en&ct=clnk&cd=1&gl=us

[BENT-2008] Storage for Petascale Computing, John Bent, James Nunez, Gary Grider, IEEE Mass Storage Conference 2008 Presentation, http://storageconference.org/2008/presentations/2.Tuesday/AM/2.Bent.pdf

[ANTYPAS-2007] Parallel IO Library Benchmarking on GPFS, Katie Antypas, SciComp07 Presentation, http://www.nersc.gov/news/presentations/kantypas_GPFSBenchmarking_SciComp07.pdf

[SCHMUCK-2002] GPFS: A Shared-Disk File System for Large Computing Clusters, Frank Schmuck, Roger Haskin, Proceedings of the Conference on File and Storage Technologies (FAST'02), 28–30 January 2002, Monterey, CA, pp. 231–244. (USENIX, Berkeley, CA.), http://www.almaden.ibm.com/StorageSystems/projects/gpfs/Fast02.pdf

[YU-2007] Exploiting Lustre File Joining for Effective Collective IO, Yu, Weikuan; Vetter, Jeffrey; Canon, R. Shane; Jiang, Song, Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on
Volume , Issue , May 2007 Page(s):267 - 274
Digital Object Identifier 10.1109/CCGRID.2007.51

[WELCH-2008] Scalable Performance of the Panasas Parallel File System, Brent Welch, Marc Unangst, Zainul Abbasi, Garth Gibson, Brian Mueller, Jason Small, Jim Zelenka, Bin Zhou, USENIX FAST-08 Conference Paper, http://www.usenix.org/event/fast08/tech/full_papers/welch/welch_html/index.html

[THAKUR-1998] A Case for Using MPI's Derived Datatypes to Improve I/O Performance, Rajeev Thakur, William Gropp, Ewing Lusk, In Proceedings of SC98: High Performance Networking and Computing, November 1998. Copyright 1998 IEEE, http://www.mcs.anl.gov/~thakur/dtype/

[CHING-2003] A. Ching, A. Choudhary, W. Liao, R. Ross, and W. Gropp. Efficient structured data access in parallel file systems. In Proceedings of Cluster 2003, Hong Kong, November 2003, http://www.mcs.anl.gov/~thakur/papers/scalable-mpi-io.pdf

[LATHAM-2007] The Impact of File Systems on MPI-IO Scalability, Rob Latham, Rob Ross, Rajeev Thakur, http://www.mcs.anl.gov/~robl/papers/latham:scalable_ops.pdf

[CHOUDHARY-2006] Choudhary, Alok N., Mahmut T. Kandemir and Rajeev S. Thankur "Collaborative Research: Scalable I/O Middleware and File System Optimizations for High-Performance Computing." High End Computing University Research Activity NSF 06-503 (2006)

[GITTINGS-2008] The RAGE radiation-hydrodynamic code, Michael Gittings_, Robert Weaver, Michael Clover, Thomas Betlach, Nelson Byrne, Robert Coker, Edward Dendy, Robert Hueckstaedt, Kim New, W Rob Oakes, Dale Ranta, Ryan Stefan, Physics.Comp-ph April, 2008, http://arxiv.org/PS_cache/arxiv/pdf/0804/0804.1394v1.pdf

[LUSTRE-LITE] Lustre Lite announcement, includes plans for multiple metadata servers, http://lwn.net/Articles/25247/

[PVFS] PVFS description, http://www.parl.clemson.edu/pvfs/desc.html

[GIGAPLUS] GIGA+: Scalable Directories for Shared File Systems, Garth Gibson, Swapnil Patil, http://www.pdl.cmu.edu/PDSI/gigaplus/index.html

[ROSENBLUM-1990] Rosenblum, Mendel and Ousterhout, John K. (June 1990) - "The LFS Storage Manager". Proceedings of the 1990 Summer Usenix. pp315-324.

[ROSENBLUM-1992] Rosenblum, Mendel and Ousterhout, John K. (February 1992) - "The Design and Implementation of a Log-Structured File System". ACM Transactions on Computer Systems, Vol. 10 Issue 1. pp26-52.

[FUSE] File System in User Space, http://fuse.sourceforge.net/

[LANL-MPIIOTEST] The Los Alamos MPI-IO Test, http://institute.lanl.gov/data