Parallelism in System Tools

Kenneth D. Matney, Sr. *and* ***Galen Shipman****,*
Oak Ridge Leadership Computing Facility,
Oak Ridge National Laboratory.

ABSTRACT: *The Cray XT, when employed in conjunction with the Lustre filesystem, has provided the ability to generate huge amounts of data in the form of many files. Typically, this is accommodated by satisfying the requests of large numbers of Lustre clients in parallel. In contrast, a single service node (Lustre client) cannot adequately service such datasets. This means that the use of traditional UNIX tools like cp, tar, et alli (with have no parallel capability) can result in substantial impact to user productivity. For example, to copy a 10 TB dataset from the service node using cp would take about 24 hours, under more or less ideal conditions. During production operation, this could easily extend to 36 hours. In this paper, we introduce the Lustre User Toolkit for Cray XT, developed at the Oak Ridge Leadership Computing Facility (OLCF). We will show that Linux commands, implementing highly parallel I/O algorithms, provide orders of magnitude greater performance, greatly reducing impact to productivity.*

KEYWORDS: Cray XT, parallel, system tools, cp, spdcp, tar, pltar

1. Introduction

Users of the Cray XT at the Oak Ridge National Laboratory Leadership Computing Facility (OLCF) have succeeded in creating parallel applications that perform I/O in parallel highly effectively. This is a result of enabling technologies provided by Lustre[1] and the **Spider** center-wide filesystems[2]. In turn, this has caused the creation of huge datasets consisting of thousands of files and many, many gigabytes of data. The problem with data is that it cannot simply lie at rest. Sometimes, it needs to be copied from one place to another for further processing and yet, at the same time, avoiding change to the original data. Sometimes, it needs to be archived for safekeeping or future reference. Sometimes, the user finds it more convenient to process the data with standard UNIX/Linux system tools/utilities than to write a custom application.

2. Motivation.

The typical user of the OLCF Cray XT really would like to perform his research in a timely manner. But, using Linux *cp* to copy a huge dataset, Linux *tar* to archive a huge dataset, or some other UNIX/Linux command (e.g., *grep*, *awk*, etc.) to process the information of a huge dataset does not yield performance consistent with this objective. In addition, with each new generation of computer system that we install, we find that the amount of data grows exponentially. This really should come as little surprise, since the amount of system resources (memory, disk) has grown exponentially as well. As a result, users solve larger problems, create larger, more complicated models, and in turn, need to process greater amounts of data. In the OLCF, we already have users with datasets that exceed 10 TB of storage. It is just a matter of time until this grows to 100 TB and beyond.

The problem with large numbers like 1 TB of storage and 100 TB of storage is that they just do not translate into human terms very well. However, by looking at the time that it takes to copy a dataset or to archive a dataset, we can understand impacts on productivity made by the serial system tools of today. In general, our benchmarks will show that Linux *cp* gets about 235 MB/s on the **widow1**[3] filesystem. This translates to about 8500 seconds (about 2.4 hrs) for a 1 TB dataset. (Bear in mind that, first, we read 1 TB and then, we write 1 TB.) In human terms, 2½ hours is a minor inconvenience. However, at 10 TB, this time will have grown to about 24 hrs and we start to see a real impact to user productivity. And, when the dataset has grown 100 TB, the user finds himself waiting around for 10 days for the copy to complete. At this point, impact to productivity may be an issue.

Creating a tarball displays similar performance characteristics. On the **widow1** filesystem, we can create an archive at up to about 270 MB/s, extract at up to 280 MB/s, and list at up to 310 MB/s. For a 1 TB dataset, these translate to times of about 7300 seconds, 6400 seconds, and 7100 seconds, respectively. However, by the time that the dataset has grown to 100 TB, these times will have increased to 204 hrs, 179 hrs, and 196 hrs, respectively. These are between 7 - 9 days and again, represent a substantial impact to productivity.

We will demonstrate, in this paper, that by using system tools which implement parallel I/O algorithms, a degree of sanity can be restored functions like *cp* and *tar*. It certainly is not reasonable to expect a user to wait around for a week while his dataset is being processed by a system tool; especially when the dataset was generated in less than a day. Even if this were reasonable, it definitely would not be desirable. To this end, we have created parallel, Lustre-aware versions of the Linux *cp* and *tar* commands. The parallel version of *cp* is *spdcp* and the parallel version of *tar* is *pltar*. Description of the algorithms employed by *spdcp* and *pltar* is beyond the scope of this paper. Details of *spdcp* algorithms have been published[4]. Furthermore, *spdcp* has been released under GPL[5] and we anticipate that *pltar* will be released under GPL, as well.

3. Performance vs. Benchmarks

In order to get some measure of the impact to productivity caused by serial vs. parallel system tools, we need to evaluate the performance of each. Another way of putting this is that the impact to productivity is directly proportional to the size of the dataset and the performance of the tool. Of course, if we could eliminate the data, what a deal! So, we will require some sort of benchmark that will provide representative values for the performance of *cp* and *tar*, and, their parallelized versions, *spdcp* and *pltar*.

For the purpose of this discussion, we shall define two synthetic datasets. But, before describing the datasets, a clarification on units is in order. Herein, we always refer to sizes and rates in base 10, except as explicitly noted. So, 1 MB/s is 1 x 10^6 bytes per second. The allocation units of the **widow1** filesystem are multiples of 1 MB(2) or 1,048,576 bytes. However, a dataset size of 100 TB specifies 1 x 10^{14} bytes of data.

The first dataset consists of small files. In this context, small is relative and the file size will be 256 MB(2) with 2688 files distributed among 12 directories. The second dataset consists of large files, with a size of 2048 MB(2) and has 336 files distributed among 12 directories. The files in the first dataset are striped to 1 OST and the files in the second dataset are striped to 8 OSTs. It will be observed that the benchmarks are constructed such that each dataset places the same amount of data on each OST used. Thus, performance variations are due to the ability of Lustre to handle large distributed (among OSTs) files vs. files that reside wholly on a single OST. Moreover, each dataset contains roughly 722 GB of data.

For *cp* and *spdcp*, benchmarking is fairly straightforward. We just measure the time for each tool to process each of the two datasets. However, for *tar* and especially *pltar*, the situation is more complicated. First, the archival tools can either create, list, or extract from an archive. Second, the performance of *pltar* will be seen to be a function of the stripe width of the archive. Thus, a parametric study will be required.

For parallel tools, the number of clients used can be a factor in performance. However, because the number of servers and targets in any filesystem is finite and limited, and because modern cluster-like systems, as used with the Cray XT, provide so many potential clients; by choosing to employ a sufficient but negligible number of clients, this factor readily is eliminated for functions that are I/O bound. For example, **widow1** has 96 OSSes, or servers by which it can transfer data. In contrast, the OLCF workhorse, **jaguarpf** has 18684 nodes, each with 12 cores for a total of 224,208 cores and each of which can move data. Thus, the 32 nodes (384 cores) that we use for the *spdcp* and *pltar* benchmarks consist of a negligible fraction (0.17%) of the total compute resource available and impact to computation job scheduling is minimal.

The *spdcp* tool previously was demonstrated[6] on an earlier implementation of the Lustre filesystem. The differences in filesystem configurations, in fact, will lead to differences in performance for the *spdcp* tool. At the same time, we would like to show that the investment in additional resources has provided some sort of benefit. So, in order to take advantage of the additional hardware offered by the configuration of the **widow1** filesystem, we necessarily must run larger benchmarks than used for the previous demonstration.

4. Linux Command Performance

4.1 The Performance of cp

The basis of comparison for copy performance is *cp*. Herein, we present the results for copy of the small file dataset and copy of the large file dataset. The time required to copy the small file dataset on **widow1** was 6152.61 seconds for a bandwidth of 235 MB/s. The time required to copy the large file dataset on **widow1** was 4863.40 seconds for a bandwidth of 297 MB/s. These values yield an average bandwidth of 262 MB/s. If we translate this performance average to the length of time that it would take to copy a 100 TB dataset, we get around 212 hrs or about 9 days. The good news is that, if we are lucky, we will not see any datasets this large for a few years. The bad news is that they are coming.

4.2 The Performance of tar

The basis for comparison used will be GNU *tar* using the POSIX 1003.1-2001 format. By way of this format, *pltar* can store and restore Lustre striping data via the PAX[7] attribute block. Without going into the details, this also can provide *pltar* a subtle performance advantage over GNU *tar* when the members of the archive use tuned striping meta-data values. At the same time, the additional overhead incurred over and above the legacy *tar* format is very small for 1 MB(2) and larger sized members (usually less than 0.1%). For these benchmarks, the overhead does not exceed 0.0004% [1 kB(2) / 256 MB(2)].

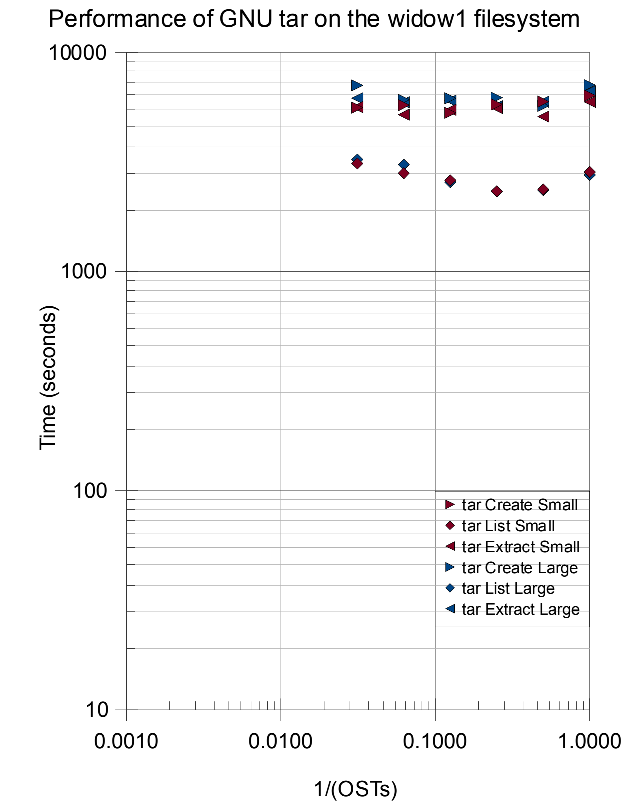

Figure 1. Performance characteristic of GNU *tar* as seen on the **widow1** filesystem.

As seen in Figure 1, the performance of GNU *tar* does not change appreciably over a wide range of OST stripe counts. This is just another way of saying that increasing the stripe width of the archive does not contribute to improved performance for *tar*. We provide performance results for archives striped to up to 32 OSTs. We also see that the time required to create or extract members from an archive is roughly the same; while the time required to list the archive is about half of that for creation/extraction. If we base bandwidth on the size of the archive, tar gets a peak (read + write) of about 280 MB/s (small files). If we use the large striped files, the bandwidth is reduced to about 255 MB/s. In either case, the listing bandwidth is about 310 MB/s. Note that except for listing the tarball, these bandwidths cover both read and write operations.

To give this some perspective, assume that there exists a 100 TB dataset that needs to be placed into a tarball. We can use the average of the bandwidths seen here to make an estimate of how long this would take. As in the *cp* case, first tar reads the data, then it writes the data. So, we have 2 x 100 TB / 267 MB/s for a reasonable approximation and running the numbers yields roughly 9 days. This is basically the same story as for *cp*.

5. Parallelized Command Performance

5.1 The Performance of spdcp

At this point, it might be best to take a step backwards. It is unfair to introduce a command, like *spdcp*, without some explanation of its design goals. Looking at *cp*, one of the shortcomings is that the Lustre meta-data is not preserved in the copy. Let's consider how this can hurt application performance.

If I have one of those 100 TB datasets, an application could have generated a 1 - 10 TB shared file for part of its restart procedure. While this sounds like a lot of data, if spread across 200,000 cores it only amounts to 5 – 50 MB per core. So, if it originally were striped across 64 OSTs, copying to the OLCF default puts it across 4 OSTs and would make the application restart take as much as **16X** as long. On the other hand, if the dataset also contains a great many smaller files, setting the default stripe width to 64 OSTs can cause other performance issues. So, there is motivation to provide a version of *cp* that preserves the Lustre meta-data upon copy. The simplest way to implement this would have been to adapt GNU *cp* to Lustre, but we still would be limited to single-node data movement performance.

The time required for *spdcp* to copy the small file dataset on was 66.37 seconds, for a total bandwidth of 21,700 MB/s. This is a speedup of almost **93X** over *cp*. The previous small file benchmark (run on the Lustre implementation[6]) only achieved a bandwidth of 9,300 MB/s. So, we see over twice the bandwidth by spdcp as previously demonstrated. In the large files test, the time required for *spdcp* to copy the dataset was 80.29 seconds, for a bandwidth of 18,000 MB/s. This is a speedup of almost **61X** over *cp*. In the previous test[6], *spdcp* only got 7,300 MB/s. So, again we see that *spdcp* on **widow1** is demonstrating between 2 and 2½ times the bandwidth as previously achieved. This provides a good degree of confidence that *spdcp* will be able to scale with the size of the dataset/Lustre installation.

5.2 The Performance of pltar

The GNU *tar* command has the same performance and parallelism issues as does GNU *cp*. It does not retain Lustre striping information and it is serial. But, it has an additional issue. While the buffer size can be reset by an argument, it is only very seldom that anyone does it. Basically, a default *tar*, run in parallel, would beat the tar out of Lustre. So, what is *tar* doing that is so bad? First, when creating an archive, the default block size of 10 kB(2) would create a mini-lock-storm for extending the file (archive). The only saving grace is that tar is serial; so, it cannot beat up the MDS (Meta Data Server) that badly. On the other hand, this does generate an excessive amount of meta-data traffic. Second, when reading files to include as members in the archive, it reads a default

10 kB(2) buffer throughout the file. In a production environment, this will lead to increased thrashing on the OSTs. So, while it would be fairly simple to modify GNU *tar* to use larger defaults with Lustre and to implement Lustre adaptations, we still would have the issue of parallelism.

The GNU *tar* command processes all members of the archive serially. Unfortunately, it is not simple to make GNU *tar* parallel. Indeed, in order to make a parallel *tar*, we must lose the concept of serial tapes entirely. This necessitates a complete design change. However, given that we are willing to make these concessions, radical performance improvement is possible. Note that *htar*[8]' used for HPSS[9] operations, employs a similar concept, but does not have multi-node capability and is not Lustre-aware.

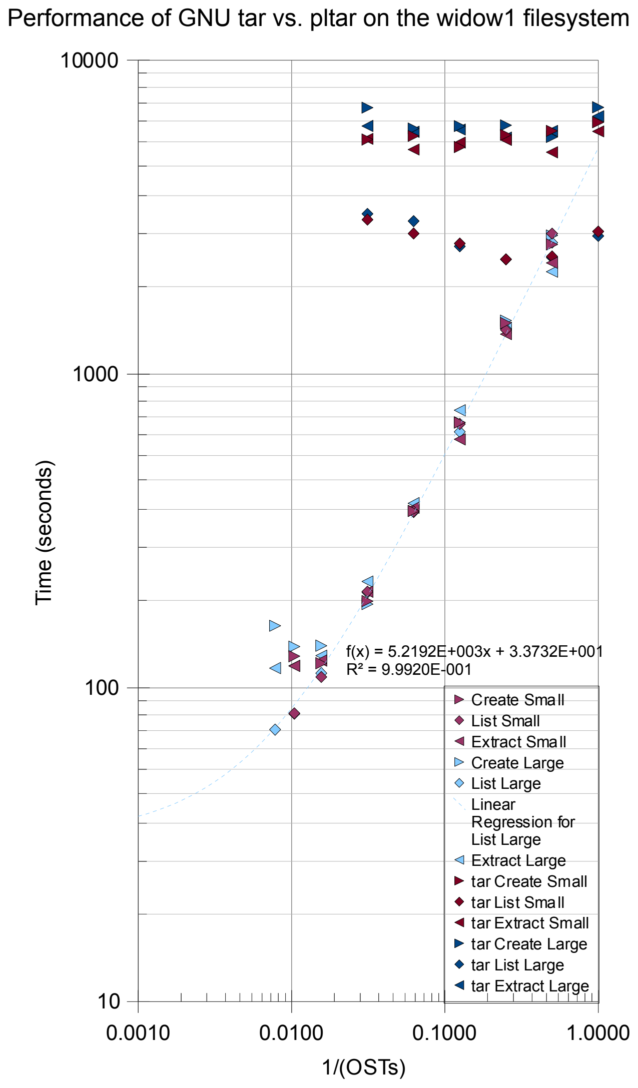

Figure 2. Performance characteristic of *pltar* relative to GNU *tar* on the **widow1** filesystem.

As seen in Figure 2, the performance of *pltar* scales very linearly with the multiplicative inverse of the number of OSTs across which the archive is striped. Data is provided for archives striped through 128 OSTs. Translating the times to bandwidth, when listing the archive, *pltar* gets up to 9800 MB/s. This is almost **32X** the bandwidth enjoyed by GNU *tar*. Note that, for listing, the scaling continues through 128 OSTs.

Upon extraction, *pltar* gets a bandwidth of up to 12,300 to 12,500 MB/s. In fact, it is relatively insensitive as to whether the archive contains large files or small files. This is an average of about **46X** the bandwidth attained by GNU *tar*. Unfortunately, the scaling holds up only through 64 OSTs.

Upon creation, *pltar* only gets a bandwidth of 10,700 to 12,100 MB/s. Even so, the bandwidth is no less than **42X** of what is seen with GNU *tar*. Again, the scaling holds up only through 64 OSTs.

To put this into perspective, let's return to that 100 TB dataset that needs to be placed into a tarball. Using the average of the bandwidths here to estimate the time for *pltar* to extract the archive, our reasonable approximation becomes 2 x 100 TB / 11,400 MB/s. But, this time, when we run the numbers, we get roughly 2½ hours instead of 9 days. Or, to look at this another way, if I am restoring my data from an archive and using *tar*, I will get to work on doing science in about a week and an half. But, using *pltar*, I can start work in a couple of hours. We definitely an make a positive impact on the worker's productivity.

It also is useful to examine the results of the curve fit to the *pltar* performance data. Specifically, as the number of OSTs approaches , the time to list the archive approaches 33.7 seconds. This time represents the total non-scalable overheads in *pltar*, including time to launch the parallel job, synchronizations between parallel processes, and time to clean up after the parallel job completes. Expressed as a fraction of the total work done, 0.64%, it demonstrates that *pltar* is very highly parallel.

6. Further Work on pltar, and Beyond

pltar, as a system utility, provides a lot of power and performance. However, the filters that GNU *tar* may employ have not yet been implemented, In addition, compressed archives are not yet supported. There is a lot of potential performance possible with compressed archives. For one thing, depending on the compressibility of the data, the archive might be much smaller. For example, assuming a compression factor of 2.5, the archive would occupy only 40% of the storage. While we would have to use many more compute nodes (cores), this means that we potentially could list/write/extract a 100 TB dataset as a tarball in about an hour.

With *spdcp* and *pltar*, we only have begun to address the peta-scale to exa-scale data gap. There are many system utilities that have not been parallelized. Examples include[10,11]: *bzip2*, *grep*/*awk*, *cut*/*paste*, and *sort*. So much remains to be done for these systems (hardware and software) to be capable of unfettered processing at larger scales. As a result, at present, the best that we can provide is a partially crippled operating system. Anytime these huge datasets must be processed by a system utility, performance will suffer so badly that the user will need to write a custom post-processor.

7. Conclusions

The commands of the UNIX/Linux operating system have long served to assist in scientific and engineering investigations. There are many system utilities to aid in this effort. However, as systems have increased in scale, parallel filesystems have been called into service to satisfy the I/O needs of the applications that run on these systems. As a result, applications have access to filesystem capabilities that far exceed those of a single node. In turn, the datasets created by these applications can and have become huge. These datasets are so large that it becomes impractical to process them with standard system commands. Thus, systems commands that implement parallel I/O across multiple nodes become a necessity. The requisite parallelism, through spdcp and pltar, has been demonstrated to exist. Furthermore, performance gains of **30X** to **100X** have been demonstrated, as well. The performance gain that is achievable depends upon the amount of parallelism inherit in the operations performed. The only practical conclusion is that parallel system tools are vital to progress.

Acknowledgments

The authors would like to thank the staff and colleagues who have contributed material to this paper. Research sponsored by the Mathematical, Information, and Computational Sciences Division, Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

About the Authors

Kenneth D. Matney, Sr. is a researcher in the Technology Integration Group which is part of the Oak Ridge Leadership Computing Facility at Oak Ridge National Laboratory. E-Mail: matneykdsr@ornl.gov. Galen Shipman is the Group Leader for the Technology Integration Team,. E-mail: gshipman@ornl.gov.

Selected References and Notes

1. Lustre™ is an Open-Source parallel filesystem and a trademark of Oracle.
2. Spider is a Lustre implementation, based on 1344 OSTs and 192 OSSes. It is accessible to all OLCF computational facilities and organized into two separate filesystems.
3. The widow1 filesystem is one of the two filesystems in Spider and consists of 672 OSTs and 96 OSSes.
4. Ken Matney, Shane Canon, and Sarp Oral, A first look at scalable I/O in Linux commands, *Proceedings of the 9th LCI International Conference on High-Performance Clustered Computing*
5. Source code to spdcp is available. Web Page http://www.nccs.gov/user-support/center-projects/lustre-user-toolkit/
6. Prior to Spider, the implementations of Lustre on OLCF computational facilities all were directly attached to the computer system. In addition, the particular implementation on which spdcp was benchmarked consisted of 80 OSTs and 20 OSSes.
7. The PAX header is an extension to USTAR (tar) format, IEEE Std. 1003.1, 2001 pax format. This permits archival/restore of filesystem-specific meta-data.
8. Htar is a tar utility to communicate directly with archives in HPSS. Web Page http://www.mgleicher.us/GEL/htar/
9. HPSS is an hierarchical storage system, commonly back-ended by tape library. Web Page http://www.hpss-collaboration.org/
10. There are parallel implementations of bzip2, however they limit scalability since parallel I/O is not implemented. Web Page http://compression.ca/mpibzip2/
11. Jeff Gilchrist and Aysegul Cuhadar. Parallel Lossless Data Compression Based on the Burrows-Wheeler Transform, *In 21st International Conference on Advanced Networking and Applications (AINA '07)*, pp. 877-884, 2007.