Petabyte File Systems Based on Tertiary Storage

Marc J. Teller marc_teller @emass.com
Paul G. Rutherford paul_rutherford@emass.com
EMASS, Inc.
10949 E. Peakview Ave.
Englewood, CO 80111
Tel: +1 303 792-9700
Fax: +1 303 792-2465

Abstract

In a matter of a few short years, the computing industry has moved from terabyte storage systems to petabyte environments. While manufacturers and implementors alike are struggling with access to 1000's of gigabytes, planners are requesting bids for systems in excess of 1,000 terabytes of data. The requirements for petabyte systems are much the same as for terabyte systems. Access must be the same as disk-based file systems; consequently, it is transparent to applications. In addition, performance should be predictable and meet minimal application requirements. Finally, system management should require few resources. In addition to these operational requirements, the size of the system requires strict attention be paid to the cost of each component. As a vendor of storage solutions, EMASS was interested in validating its AMASS product against these requirements.

This paper documents a project for benchmarking a petabyte storage system residing in our Denver facility. In addition to profiling the ability of AMASS to scale to a petabyte, the study looks at the operation of a large file system, alternate tools for dealing with large file systems, and additional development and research to support petabyte file systems.

Results of the project indicate a high degree of scalability for AMASS up to the initial project-imposed limit of 16 million files and identified specific areas for enhancement of the AMASS product.

1. Introduction

We used to worry about storing and retrieving terabytes (TB) of data. Today, magnetic disk can be used for small numbers of terabytes. In fact, most operating systems can manage these large disk farms as standard file systems. Therefore, these operating systems have become the de facto standard against which higher capacity storage systems are measured.

At higher capacities, tape technology has met the challenge by providing us with high density recording. Standard half-inch cartridges now contain up to 50 gigabytes (GB). A small desk side automated library, using low-cost 30 GB tapes, can easily hold the 20 to 50 cartridges needed for a terabyte of data and fit within the budget of most small departments. Because hundreds of terabytes has become the normal mode of operation for data centers, it is now time to push toward a thousand terabytes (a petabyte) of online data.

While still costly, a petabyte (PB) system requires 20 to 50 thousand cartridges. The system

can be automated using available tape libraries from one of the high storage capacity providers such as EMASS or Storage Technologies. Given that the hardware is available, the real question is management software. Is there software that can scale to this capacity? Some management software is precluded because the total number of bytes stored is inadequate; other software is excluded because the number of files managed is inadequate.

This paper presents the results obtained in creating a petabyte file system, the scalability of the metadata and operational characteristics of a large file system, and future work needed for petabyte file systems.

2. The Problem

At the 1996 NASA Goddard Storage Conference, a panel of large system users commented on the lack of storage management software that could address their needs. Many were still considering in-house development or were feeling forced into less than adequate systems. Most were using Hierarchical Storage Management [2] (HSM) products such as DMF from Silicon Graphics (formerly Cray Research), UniTree from UniTree Software Inc., or FileServ from EMASS. The panelists were concerned both with long term availability and with scalability of the products for petabyte file systems.

A list of some of the requirements, by no means intended as exhaustive or addressed by this project, for a system solution to the issues on this area can be summarized as follows:

- High capacity media.
- Large number of files (tens of millions).
- Large amount of total data (multiple petabytes).
- Stable, reliable, and easy-to-use software. For example, file system view of storage allows easy programming and portable applications.
- Data is kept online. Offline storage, except for backup copies, is unacceptable.
- Relatively predictable data access times.
- Multiple host platform support. This allows server platform selection by the end-user.
- Performance able to meet, or exceed end-user expectations.

2.1. Media Technology

Given current technology, the least number of cartridges that can reasonably be automated yet still provide petabyte storage capacity is 6,667. Table 1 provides examples of the media types applicable to this size system.

Vendor & Drive Model	Media Type	Capacity /Cartridge†	Cartridges /PB	Native Transfer Rate	$/GB*
Ampex DST312/M	DD-2	150GB	6,667	15 MB/s	0.84
StorageTek Redwood	SD-3	50GB	20,000	11 MB/s	1.80
Quantum DLT7000	CT-IV	35GB	28,572	5 MB/s	2.83
Sony SDX-300C	AIT-1	50GB	20,000	3 MB/s	2.00
magnetic disk drive	n/a	23GB	43,479	13 MB/s	150.00
magnetic disk array	n/a	176GB	5,682	18 MB/s	660.00

†This represents native capacity, so does not include any compression the drive may support.

***This cost reflects only the media for tape devices. Because a tape drive is used to service all media, there is not a one-to-one correlation of drive to capacity as with a disk drive or disk array.

Table 1. Media Possibilities

For comparison purposes, the table shows standalone disk drives as well as drive arrays. But even if we used a 1 TB-sized array, we would still need a thousand of them to store 1 PB of data. In any case, attaching the 1,000 arrays would pose a significant problem.

Cost, access profile, and transfer rate must be considered. Not all automated library vendors support the different types of drives. So while the actual library choices might be limited, solutions are available.

2.2. Number and Size of Files

System administrators are concerned about the amount of disk space needed to represent the files in a petabyte file system. Although one cannot predict the exact mix of file sizes, an estimate of the range of files a petabyte system must handle can be made. If the average file size is 1 GB, the number is a million files. For this project, 16 million files were created using a single 62.5 MB set of data blocks.

In a regular file system the actual space occupied by files must take into account the space used by the inodes and indirect blocks needed to access the file data. The size of the inodes range from 128 bytes to 512 bytes depending on the vendor. If HSM is added, database information increases to track the file location. Typically this is between 512 and 1024 bytes of data per file. In an archive file system like AMASS, the database contains the file metadata and totals 120 bytes per file. In a regular file system with an HSM, about 20 GB will be needed to hold the metadata. With AMASS, about 4GB will be needed. However, even 20 GB is not unreasonable.

Another issue in the computing industry is the size of the largest file that can be stored in a file system. Extended file systems support at least 40 bits of file address, or 1 TB. File systems are moving toward full 64-bit addressing in the next year or so. This gives us the ability to address 18.5 exabytes in a single file.

2.3. Data Access

While magnetic disk has a relatively uniform access time, tertiary storage devices (specifically tape, a sequential access device with slower positioning time) have a decidedly non-uniform access time. Access to data on tertiary storage is governed by a combination of the following:

- Time for a drive to become available.
- Time to unload media in the selected drive.
- Time for the robotic arm to fetch the appropriate media and load it into the selected drive.
- Time to position the media to where the data resides [1].
- Time associated with accessing metadata to obtain addressing information.

In order to minimize the impact of the above times, systems can employ a cache (usually based on magnetic disk) to hide the overhead by managing *readahead* and *residency* of data. The caching behavior is important due to the difference between disk and tape access characteristics. Contention for tape drives becomes a significant issue in a heavily loaded system. For reads, there is not much that can be done. The data is on tape so the requestor must wait for a drive to become available. For writes, a disk based cache can be used to hold data until tape I/O can be completed.

2.4. Issues With Large Datasets and Metadata

When a file system begins to store a very large number of files or datasets, or a very large number of directories, the time it takes to access file data becomes a growing concern. This is usually (with the exception of simultaneous user access interaction) the result of acquiring the metadata that describes the location of the file data on the physical media. It does not matter whether the file system stores metadata in inodes (as in standard UNIX® file systems) or stores metadata in a database (as in AMASS [3]). While the inode-based systems suffer from indirect block lookup and read operations (even extent-based file systems can suffer from extent metadata management), database systems can suffer from scanned lookup times depending upon the nature of database operations and how file keys are used. As a result of these concerns, the project focuses on characterizing metadata access performance as an AMASS file system grows from initialization to 16 million files referencing over a petabyte of data (see Section 4.1.).

3. Project Goals

This project is designed to provide answers to the questions raised in the previous discussion. As a software vendor, we are interested in how well our AMASS product meets the needs of a petabyte system.

AMASS – Archival Management And Storage System – stores and retrieves files using robotic libraries, jukeboxes, and standalone drives. AMASS provides transparent, direct access, to the files stored in the AMASS file system. AMASS makes the drives and media – normally considered offline storage – appear as a single, online, logical device with a single, mounted, file system. Therefore, the extensive storage provided by a library appears as

one large file system. Because AMASS is implemented at the virtual file system (VFS) layer of the server's operating system, it is transparent to application software.

The primary goals of the project are to verify scalability of AMASS to 1 petabyte, to profile operation of user commands with large datasets, to identify product enhancements for petabyte size systems, and to create a base system for exploration of multi-petabyte systems.

4. Project Definition

The project being undertaken is designed to evaluate two scenarios using AMASS as the storage management environment. In phase I, the petabyte ($1x10^{15}$ bytes) storage requirement of very large systems and the performance impact, from an end-user perspective, is analyzed. In phase II (to be described in a future paper, see Section 7.), the data access characteristics of AMASS when executing random access requests to data files that range from 100 GB to 1 TB in size and span media volumes within the library, will be analyzed.

4.1. Large Dataset Storage

In an effort to characterize the performance plateaus in a file system as it approaches large numbers of files, directories, and amount of data stored, a test system has been setup that is designed to ramp to a data storage in excess of 1 PB and over 16 million files. Because the only measure of interest is the relative performance of the file system database against the scale of the file system population, a slow platform is just as useful as a fast platform. This portion of the investigation is running on a Sun Microsystems SPARCclassic with Solaris 2.5 and AMASS Version 4.7.1. The media choice is not germane as data performance is being evaluated in a later phase of the project. In addition, the amount of data being stored is not germane to the scalability of metadata in the file system because each byte of data is directly addressed in AMASS, unlike typical UNIX file systems where no indirect addressing is used. This provides a very consistent access time to any byte of data in the petabyte store except for the issues raised in Section 2.3.

A single set of data blocks is used for all 16 million files. The data descriptors for each file reference the same blocks on the media. However, links are not used so that the system appears to have actually stored a petabyte of data. This saves the $7.1 million for media (100,000 Magstar® cartridges) and 1.6 years to write the files (at 20MB/sec. sustained).

Figure 1 illustrates the namespace created. It consists of 512 directories, under the root directory for the library. Each of these directories has 128 subdirectories each of which contains 256 files. In other words, there are 512 sub-trees with 128 directories and 32,768 files (128 * 256; for a total of 512 * 128 * 256, or 16,777,216 files).

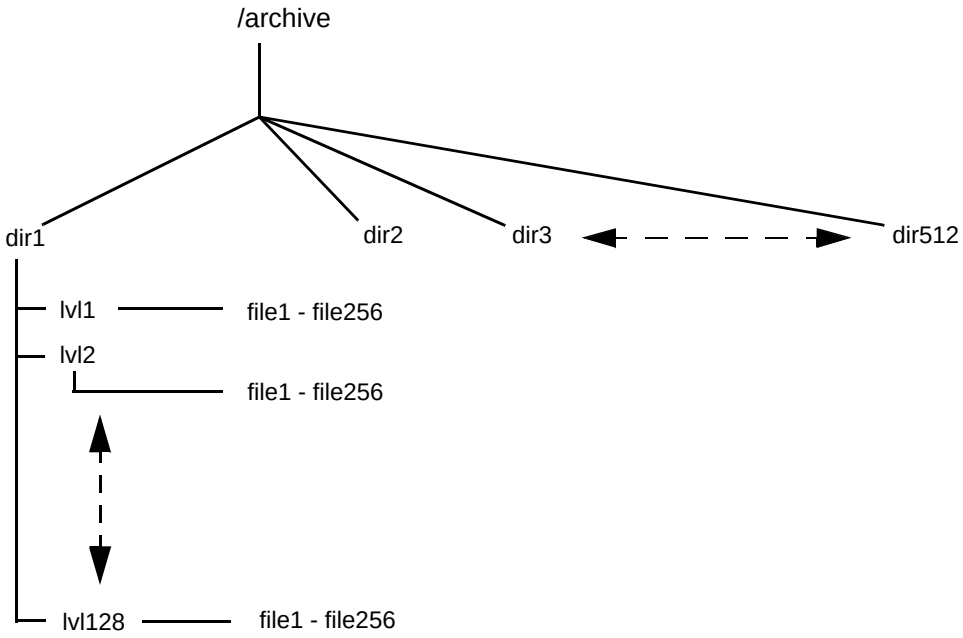

Figure 1. Namespace Organization

The structure of the namespace was chosen as a relatively easy way to increase the number of files to the goal of 16 million. Since the namespace is built using an import facility in AMASS, the ability to create 32,768 files (128 directories each containing 256 files) with each import simplifies the namespace creation.

Because the metadata for the file system is accessed by a keyed lookup, the actual namespace organization does not significantly impact performance and consequently, is not an issue. Although it might be possible to construct an unbalanced namespace layout that could affect the test results, this would not be a real-world scenario since most systems have a more balanced structure as did the test scenario.

Phase one of the project's performance evaluation is oriented towards metadata access since AMASS uses direct access for data, after the metadata has been retrieved. This has been accomplished by using standard UNIX® commands that query the namespace or the file/directory attributes without accessing the actual file data, such as `ls(1)`, `find(1)`, and `pwd(1)`. The performance data is gathered after a group of four trees are created, resulting in the addition of 131,072 files across 512 sub-directories to the namespace.

The characterization script uses a ten iteration loop with a recursive '`ls`' beginning in the 64th subdirectory of the first tree, followed by a '`find`' beginning in the 100th subdirectory of the 3rd tree, and ending with a '`pwd`' in the 127th subdirectory of the 2nd tree. The ten iteration loop is followed by a five iteration loop with the same commands, in the same subdirectories running in the last tree created. All results are averaged and use the UNIX® `time(1)` command for reporting the *real* execution time of the commands. This approach was chosen so that the end-user perspective can be examined and all of the "benchmark" tests that have been received from customers are also based on timing command level scripts.

5. Results

5.1. Metadata Performance

Figure 2 shows essentially flat performance, indicating a very scalable system. The absolute performance values can be reasonably tied to a slow, minimally configured workstation. The relative performance shows the scalable nature of the system. The graph data points include a 90% confidence interval.

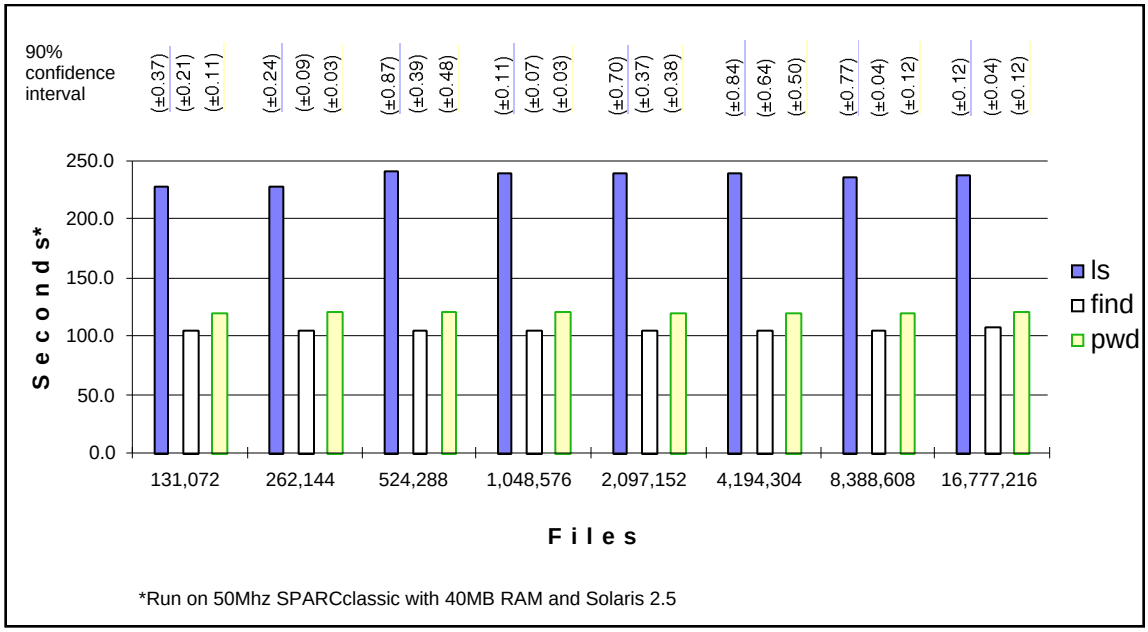

Figure 2. Metadata Test Results

As can be seen, a 'pwd' command for a location 127 levels deep in the tree takes about 120 seconds in real time. This timing was gathered by executing the standard UNIX® `pwd(1)` command that performs recursive `stat(2)` calls. In other words, the standard file system paradigm is at work. AMASS implements a file system view of the library to allow unmodified applications to run using the media stored in the library. While this is extremely beneficial in terms of a customers investment in application code, it does incur some overhead, including kernel <-> user space context switches, to allow redirection of file system requests to the AMASS code.

A future phase of the project will compare the scalability of AMASS to that of a native file system.

5.1.1. Database direct-access for commands

To see if performance could be improved using direct access to metadata, we wrote an AMASS 'print working directory' command. Figure 3 shows the results of comparing the two commands, the standard UNIX password command and the new AMASS password command. As in Figure 2, the data points include a 90 percent confidence interval.

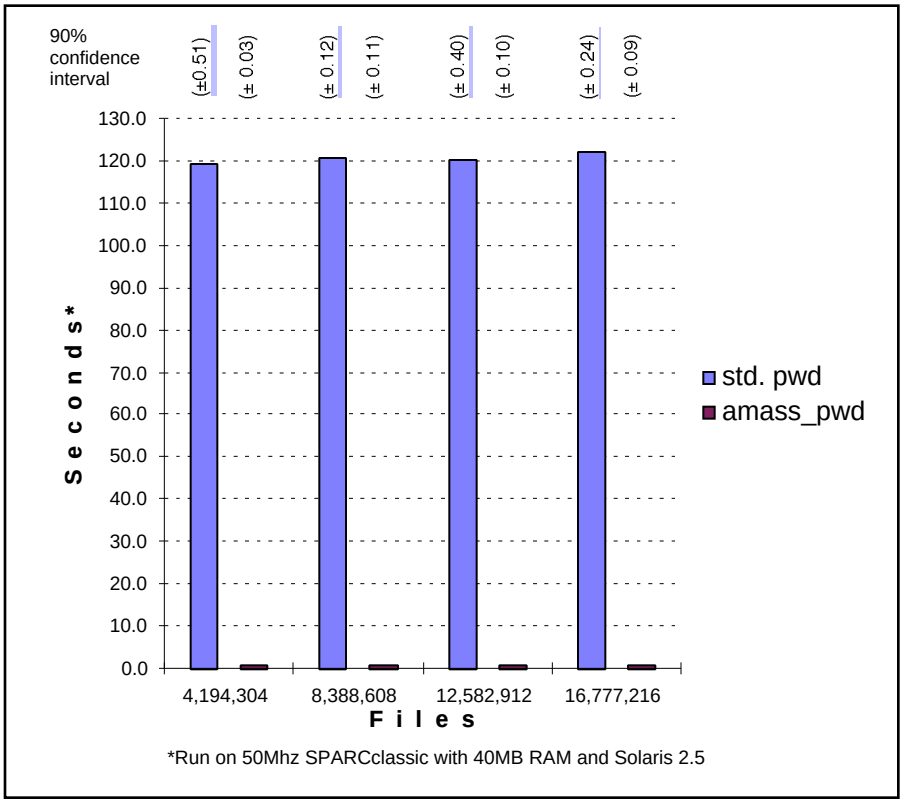

Figure 3. PWD Comparison

As the graph shows, the performance of direct metadata access significantly out-performs the performance of file system access. However, this increase sacrifices standard command transparency. For environments that require increased performance, an Application Programming Interface (API) is available for AMASS.

Analysis of the standard UNIX `pwd` command indicates that the main loop for walking the namespace tree consists of:

- open("./../../ ...")
- fcntl (sets 'close on exec flag' for open file)
- fstat -- 2 calls on open file descriptor
- getdents -- 2 or more, depending on the size of the directory
- close

However, the `amass_pwd` command uses a namespace walking loop consisting of:

- lseek (1 for main record, 1 for name record)
- read (1 for main record, 1 for name record)
- brk -- 2 calls of 0x2000 each

AMASS uses a Virtual File System (VFS) "driver" to re-direct file system calls to a user-level daemon that queries the database for metadata information. The data is returned to the caller after it has been transformed into a compatible format. Therefore, standard file system calls incur the overhead of this redirection and transformation, including context switching between kernel-level and user-level. While this overhead is negligible for I/O operations (due to the implementation), it can become significant in metadata-only opera-

tions due to database lock contention.

In the `amass_pwd` case, the AMASS database points from the directory and file record to its name record, so neither directory searching (via `getdents`) nor `open` calls are required. Bypassing the file system interface and directly accessing the database eliminates the most significant portion of the overhead.

In addition to the `pwd` command, we evaluated the benefits of using a variant of the `find` and `ls` commands. Ultimately, when a file system scales to many millions of files, the standard metadata access paradigm, which uses `stat` and `getdents` to access inode and directory information, has an unacceptable overhead associated with it. Therefore an alternative path to this information is required to achieve high performance goals.

While the user of such a system can fine-tune the applications used to manage the storage system, it is reasonable for the storage software vendor to make variants of the typical UNIX command set available for these sites.

6. Additional Requirements

Based on the operational experience during the project, this section outlines a set of requirements for any solution to address the petabyte storage world, and presents approaches for modifying AMASS to meet these requirements.

6.1. Data Integrity and Technology Migration

With such an enormous amount of data being managed, a method to allow protection of data from loss due to media failure must be provided. EMASS refers to this as Total Data Life Management (TDLM) since it designed to manage data integrity during the entire lifecycle of the data. As each volume of media is unmounted/unloaded, heuristics are updated to capture retry counts and ECC codes. This data is examined and suspect media are selected for data copy operations. When the data is copied, the original, suspect, media would be made inactive and the operator notified so it can be removed from the library. This process helps to ensure the integrity of data stored within the petabyte environment. As archives grow in size, many volumes remain unaccessed. Programs referred to as “tape sniffers” access each volume in the library on a rotating basis to randomly read files. This activity results in collecting tape error rates for each volume in the file system on a periodic basis. The rates determine which volumes should be copied before any data loss is incurred.

As new drive technologies become available, the question of how a site moves existing data from older media to new media becomes increasingly important. When the amount of data is only a couple of terabytes, a batch data transcription process is probably acceptable. On the other hand, when the amount of data exceeds this number or when the site cannot tolerate any data downtime, another method must be found. Although transcribing a petabyte of data is unacceptable as a foreground activity (see Section 6.3. for an estimate of the time involved), every large site will eventually have to transcribe data as older drives are either retired or no longer supported.

One approach would be to treat the older media as read-only. As this data is updated, it is

copied to new media, and the older version is marked as invalid. Over time, much of the data would be transcribed in this manner. Any unmodified, and therefore non-transcribed, data is handled in a batch mode in order to complete the migration.

Another approach, and probably one more likely to be used, is to provide a set of tools to assist administrators in planning and executing the migration. These tools would provide estimates of the time required to migrate all of the data from the older media to newer media and mechanisms for this migration without requiring or causing any service interruption for users. Server platform and drive independence is a great help as the site is not constrained in its decision on which technology to adopt.

This leads into the metadata standard for describing data that is currently under development (see Section 6.7.).

6.2. Extensible Metadata

When a site is managing a petabyte of data, any user-defined keywords for searching the available data is an aid to the user community. One approach is to use well understood catalog tools on top of AMASS. Another approach, is to provide the ability for users/applications to manipulate metadata that is opaque to AMASS. A pair of AMASS API (Application Programming Interface) routines could provide a way for client applications to set and retrieve metadata that AMASS is currently unaware of. The metadata could be correlated with either a file or with a directory, providing flexibility in organization.

6.3. Backup and Recovery

After a petabyte, or more, of data has been stored, how do you back up this file system? Typically a full backup essentially duplicates the storage size. If you assume a sustained throughput, including media exchange times, of 20 megabytes/second, a 1 petabyte backup would take 1.6 years to complete! Add to this the cost of media, plus the usually overlooked impact of having a reduced number of drives available to respond to client requests during this 1.6-year long backup! The validity and scope of this petabyte backup is staggering — to say the least.

For petabyte file systems, a revision history paradigm may be more efficient than the more traditional image backup paradigm. In a revision history paradigm, only modifications to a baseline image are copied. This is essentially a versioned, incremental backup. Unlike a traditional backup, only the changes made to files since the last backup are copied, the entire file is not. The initial backup, which forms the baseline image of each file, could be created with a file replication feature run when the file is first placed in the file system or with a manual copy run in batch mode during system idle time. The net effect of this approach is to reduce the impact of backing up the file system and constrain the amount of data needed to ensure the recreation of the dataset in case of disaster. The nature of incremental backups, however, is that recover is also incremental from the last full backup.

However, from an archival storage perspective, with slow changing data, the real issue is disaster recovery. In this scenario, the use of multiple archives is optimal, along with replication of the data streams to both archives when data is ingested. This provides replication of data, on the fly, as opposed to batch mode, which is too expensive in time and resources.

Another option to consider is tape RAID where data is spread across multiple tapes, with a parity tape, so the loss of any single tape does not result in the loss of any data. The liability of using this option is the mount time needed to retrieve a file. Any option used to address disaster recovery — must pay a price in increased overhead.

6.4. Startup and Consistency Checks

Another feature required for petabyte storage management is startup and recovery times. With a database almost 3.9 gigabytes in size, checking the consistency of the file system against the database takes a long time to complete — even if the database has no errors to correct. It is critical that this process be optimized. Some of the optimizations being investigated (given that the process is CPU-bound) are:

- Multi-threading the utility as most systems with a very large dataset environment will likely be hosted on multiprocessor systems.
- Optimizing the code for larger memory machines. AMASS can execute on systems with as little as 24Mbytes of RAM; however, this results in a coding style that makes use of redundant code loops to avoid a larger memory footprint. Most multiprocessor hosts running AMASS have significantly larger memory configurations.

6.5. Long Runtimes

Because management utilities, like `df`, scan huge amounts of data, running this type of command can raise concern about the health of the system when it seemingly causes the system to hang. The solution may be for the software vendor to provide a set of utilities that duplicate the functionality of the file system-based UNIX tools, yet employ a shortcut to access the file system metadata directly, improving the performance of some commands (see Section 5.1.1. and Figure 3).

6.6. System Utilities

As with some standard UNIX utilities, some of the AMASS commands use fixed-width formats in reporting size, block number, and other information that might overflow in a petabyte storage environment ([5] describes some issues in migrating to 64-bit file system environments). These need to be identified and modified to support the larger environments. The most likely approach is the one followed by most UNIX vendors: a new set of commands, replacing the older ones, can be installed which will handle the larger data fields needed to correctly present data about the dataset store.

6.7. Standards

An aspect of the technology migration issue is being addressed by the AIMM [6] Standards committee C21.1. Using current technology, petabyte systems require many thousands of cartridges. As mentioned in Section 6.1., it is impractical to move this amount of data from one file management system to another and, in the process, force the data to be rewritten in a different format. The C21.1 committee is working to define a metadata standard that describes the data contained on media. EMASS is actively participating in this effort.

When the storage management system at the source site provides a standard metadata, the storage management system at the destination site can import the metadata and have sufficient information to access the data contained on the original media [7].

7. Future Work

To study the data access characteristics of AMASS with large (1TB) file sizes, a second phase of the project will be conducted. In this phase, an AML/S library [4] equipped with a pair of IBM Magstar 3590 drives and loaded with 120 tape cartridges will be connected to a Silicon Graphics Challenge L running IRIX 6.2 and AMASS Version 4.9.1.

A file creation task will be used to write the 1 terabyte file with drive compression disabled (thereby allowing a true data performance characterization). The task will report the amount of data written every hour resulting in a 'write rate' profile of a single writer across the entire terabyte of data. It should be noted that media exchange times are included in the profile (the terabyte file should require 100 cartridges at 10 gigabytes each).

After the file has been completely written, a pair of reader tasks will read random, non-overlapping, regions of the file. These tasks will report data access times, drive utilization, and media exchange wait times. This test scenario does not investigate resource (drive or volume) contention resolution but it does establish a reasonable profile of I/O activity. Variations from the observed performance can be affected by the nature of the cache environment, the number of drives, and the way in which the data is organized within the namespace.

The reader tasks are designed to scatter read requests across media volumes (never the same media volume by both readers). Ideally, each reader would have a drive available in which to load the media volume containing the requested data region (even if previously accessed media needs to be unloaded first). Given that no more than one reader task would be accessing a specific piece of media (due to the gap between regions read), there should be no read request queueing except for media exchange times.

8. Conclusions

As phase one of this project clearly demonstrates, there IS storage management software available that can scale file systems to petabyte capacity. As petabyte-sized file systems are typically used for long-term data storage and retrieval, it is essential that library and media types be vendor-independent to allow for technology migration during the lifecycle of the data. While some operational issues need to be improved, such as data integrity and administrative tools, these issues can be resolved as the file system grows to a petabyte in size.

9. References

[1] Sarawagi, S., "*Database Systems for Efficient Access to Tertiary Memory*", Fourteenth IEEE Symposium on Mass Storage Systems, September 1995.

[2] Woodrow, T., "*Hierarchical Storage Management System Evaluation*", Third

NASA Goddard Conference on Mass STorage Systems and Technologies, October 1993.

[3] Sarandrea, B., "*Storage System Architectures and Their Characteristics*", Third NASA Goddard Conference on Mass STorage Systems and Technologies, October 1993.

[4] Produced by EMASS, Inc. See *http://www.emass.com* for information on the library.

[5] Shaver, D., Schnoebelen, E., Bier, G., "*An implementation of Large Files for BSD Unix*", USENIX Winter Conference Proceedings, January 1992.

[6] Association for Interactive Media and Marketing, *Metadata for Interchange of Files on Sequential Storage Media Between File Storage Management Systems (FSMS)*.

[7] For more information on this effort, please see the following URL: http://ses2.ses-inc.com/~joelw/