The following paper was originally published in the
Proceedings of the 3rd Symposium on Operating Systems Design and Implementation
New Orleans, Louisiana, February, 1999

Logical vs. Physical File System Backup

Norman C. Hutchinson
University of British Columbia
Stephen Manley, Mike Federwisch, Guy Harris, Dave Hitz, Steven Kleiman, Sean O'Malley
Network Appliance, Inc.

For more information about USENIX Association contact:

1. Phone: 1.510.528.8649
2. FAX: 1.510.548.5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org/

Logical vs. Physical File System Backup

Norman C. Hutchinson
Computer Science Department
University of British Columbia
Vancouver, B.C., Canada V6T 1Z4
hutchinson@cs.ubc.ca, http://www.cs.ubc.ca/spider/norm

Stephen Manley, Mike Federwisch, Guy Harris
Dave Hitz, Steven Kleiman, Sean O'Malley
Network Appliance, Inc.
2770 San Tomas Expressway
Santa Clara, CA 95051
{stephen,mikef,guy,hitz,srk,sean}@netapp.com, http://www.netapp.com

Abstract

As file systems grow in size, ensuring that data is safely stored becomes more and more difficult. Historically, file system backup strategies have focused on logical backup where files are written in their entirety to the backup media. An alternative is physical backup where the disk blocks that make up the file system are written to the backup media. This paper compares logical and physical backup strategies in large file systems. We discuss the advantages and disadvantages of the two approaches, and conclude by showing that while both can achieve good performance, physical backup and restore can achieve much higher throughput while consuming less CPU. In addition, physical backup and restore is much more capable of scaling its performance as more devices are added to a system.

1 Introduction

As a central player in every operating system, file systems have received a tremendous amount of attention from both the academic and industrial communities. However, one important aspect of file systems has been conspicuous by its absence in the literature - file system backup and restoration. As file systems grow in size, ensuring that data is safely stored becomes more and more difficult. The research literature on this topic is extremely limited; Chervenak et al. present a good survey of the current literature [CVK98]. On the other hand, industry is very interested in both the correctness and performance of file system backup [Sun97, CCC98].

In evaluating a backup/restore strategy, a number of end-user desires must be balanced. On the backup side, it is the hope of every system administrator to never need any of the backup data that is collected, and in fact, in normal operation the vast majority of the data that is backed up is never looked at. This means that maximizing the speed with which data can be backed up, and minimizing the resources (disk and CPU) that are used in performing the backup are very important. This also means that the robustness of backup is critical. Horror stories abound concerning system administrators attempting to restore file systems after a disaster occurs, only to discover that all the backup tapes made in the last year are not readable.

Because backup data is kept for a long time, both to provide file system history and increased resilience to disasters, it is important that the format used to store the data be archival in nature. That is, the data should still be recoverable even if the hardware, operating system, or backup/restore software of the system has been changed since the backup was created.

On the restore side, there are at least two primary sorts of restores that are performed. We call these disaster recovery and stupidity recovery. Disaster

recovery comes into play when whole file systems are lost because of hardware, media, or software failure. A disaster recovery solution involves a complete restore of data onto new, or newly initialized media. The solution to the disaster recovery requirement also allows migration of data from one set of media to another. Stupidity recovery manifests itself as requests to recover a small set of files that have been "accidentally" deleted or overwritten, usually by user error.

There are two primary approaches to the backup/restore problem: logical and physical. A logical (or file-based) strategy interprets the file system meta data, discovers which files need to be duplicated and writes them to the backup media, usually in a canonical representation which can be understood without knowing very much if anything about the file system structure. A dump command has been implemented in every version of Unix since Version 6 from AT&T. The current standard was developed as part of the original BSD Unix effort. Other logical backup approaches include using tar or cpio. Each of these tools define their own format for the data, but in both cases the format is architecture neutral and well documented. A number of companies such as Legato, Veritas, and IBM have extended this idea by defining their own proprietary formats (typically based on tar or cpio) which can be used to stream file system data to a backup server.

A physical (or block-based) strategy duplicates the physical medium on which the files are stored (disks or disk arrays) onto the backup medium without interpretation (or with a minimum of interpretation). Physical backup has primarily been used to copy data from one medium to another (the Unix "dd" command), but has also been developed into a fully functional backup/restore strategy by Digital [GBD96]. The Plan 9 file system uses a physical block level copy-on-write scheme to implement its file system "epoch" scheme, which is similar in some respects to incremental file system backup [Qui91]. One of the advantages of a physical dump is that all file and file system attributes are duplicated, even those that may not be representable in the standard archival format. Examples of such attributes include CIFS access control lists, snapshots, hidden files, file system configuration or tuning information and file system statistics.

Recent years have witnessed a quiet debate between the merits of file-based versus block based backup schemes. Unfortunately, the comparisons have generated little interest for two reasons. First, since the two schemes are fundamentally different, it is difficult to find common ground on which to base reasonable analyses. Second, it is rare to find systems in which the two schemes have both been implemented with comparable degrees of completeness and attention to detail.

Network Appliance's WAFL (Write Anywhere File Layout) filesystem [HLM94] implements both the logical and physical backup strategies. By using snapshots (consistent, read-only images of the file system at an instant of time) both logical and physical dump can backup a consistent picture of the file system. WAFL's physical backup strategy is called image dump/restore. It takes advantage of the snapshot implementation to quickly find those disk blocks that contain data that needs to be dumped. Furthermore, the bookkeeping necessary to support copy-on-write enables incremental image dumps — only those disk blocks that have changed since the last image dump are included in an incremental dump. While taking advantage of the benefits of WAFL, image dump bypasses many of the file system constructs when reading and restoring data in order to improve performance. Thus, it is a true block-based implementation.

WAFL's BSD-style dump and restore utility has also been modified to take advantages of the features of the WAFL file system. First and foremost, unlike most other BSD-style dumps, the Network Appliance dump is built into the kernel. Since Network Appliance's filers are specialized to the task of serving files, there is no user level. Instead the file system has been designed to include dump and restore. This not only avoids context switches and data copies, but further allows dump and restore to utilize their own file system access policies and algorithms as well as giving them access to internal data structures. Unlike the image dump utility which bypasses the filesystem however, BSD-style dump and restore uses WAFL to access data.

WAFL therefore provides an intriguing test-bed for comparing and contrasting file-based and block-based backup strategies. First, rarely is a file system designed with backup as one of the primary goals. Second, system designers do not usually optimize both block-based and file-based backup. Finally, the nature of WAFL enables functionality that supersedes fundamental backup and restore. On the physical backup side, WAFL's image dump technol-

ogy allows more interesting replication and mirroring possibilities. On the logical backup side, some companies are using dump/restore to implement a kind of makeshift Hierarchical Storage Management (HSM) system where high performance RAID systems nightly replicate data on lower cost backup file servers, which eventually backup data to tape.

Section 2 of this paper describes those features of WAFL that are important in our discussion of backup strategies. Section 3 of this paper describes logical dump as embodied in WAFL's modified BSD dump. Section 4 describes WAFL's physical dump strategy (image dump). Section 5 compares the performance of the two utilities. Section 6 describes the future possibilities inherent in each scheme, and Section 7 concludes.

2 WAFL - Write Anywhere File Layout

In many respects, WAFL's disk format is similar to that of other UNIX file systems such as the Berkeley Fast File System [MJLF84] and TransArc's Episode [CAK+92] file system. For example:

- WAFL is block based, using 4 KB blocks with no fragments.
- WAFL uses inodes to describe its files.
- Directories are specially formatted files.

Like Episode, WAFL uses files to store meta-data. WAFL's two most important meta-data files are the inode file (which contains all inodes) and the free block bitmap file. Keeping meta-data in files allows meta-data blocks to be written anywhere on disk. WAFL has complete flexibility in its write allocation policies because no blocks are permanently assigned to fixed disk locations as they are in the Berkeley Fast File System (FFS). The only exception to the write anywhere policy is that one inode (in WAFL's case the inode describing the inode file) must be written in a fixed location in order to enable the system to find everything else. Naturally, this inode is written redundantly.

2.1 The Snapshot Facility

Snapshots are a primary benefit of WAFL's write anywhere approach. A snapshot is an on-line, read-only copy of the entire file system. The file server is able to copy the entire file system in just a few seconds. The copy uses no additional disk space until files are changed or deleted due to the use of copy-on-write. Only as blocks in the active file system are modified and written to new locations on disk does the snapshot begin to consume extra space.

Snapshots can be used as an on-line backup capability allowing users to recover their own files. Snapshots can be taken manually, and are also taken on a schedule selected by the file system administrator; a common schedule is hourly snapshots taken every 4 hours throughout the day and kept for 24 hours plus daily snapshots taken every night at midnight and kept for 2 days. With such a frequent snapshot schedule, snapshots provide much more protection from accidental deletion than is provided by daily incremental backups. WAFL allows up to 20 snapshots to be kept at a time. How long snapshots can be kept depends on how quickly the file system changes, but ranges from a few hours to a few weeks. Snapshots also simplify backup to tape. Since snapshots are read-only copies of the entire file system, they allow self-consistent backup from an active system. Instead of taking the system off-line, the system administrator can make a backup to tape of a recently created snapshot.

A WAFL file system can be thought of as a tree of blocks rooted by a data structure that describes the inode file. The inode file in turn contains the inodes that describe the rest of the files in the system including meta-data files such as the free block bitmap. WAFL creates a new snapshot by making a duplicate copy of the root data structure, and updating the block allocation information. Since the root data structure is only 128 bytes, and since only the block allocation information needs to be copied on disk, a new snapshot does not consume significant additional disk space until a user deletes or modifies data in the active file system. WAFL creates a snapshot in just a few seconds.

Due to the presence of snapshots, the question of whether a block is free or in use is more complicated than in file systems without snapshots. Typically, when a file is deleted all of the blocks holding data for that file may be marked as free. In WAFL,

however, if the file is in a snapshot, then the blocks must not be reused until all of the snapshots holding the file are deleted. Accordingly, WAFL's free block data structure contains 32 bits per block in the file system instead of the 1 bit per block that is needed without snapshots. The live file system as well as each snapshot is allocated a bit plane in the free block data structure; a block is free only when it is not marked as belonging to either the live file system or any snapshot.

2.2 Consistency Points and NVRAM

At least once every 10 seconds WAFL generates an internal snapshot called a consistency point so that the disks contain a completely self-consistent version of the file system. When the filer boots, WAFL always uses the most recent consistency point on disk, which means that even after power loss or system failure there is no need for time consuming file system checks. The filer boots in just a minute or two, most of which is spent spinning up disk drives and checking system memory.

The filer uses non-volatile RAM (NVRAM) to avoid losing any NFS requests that might have occurred after the most recent consistency point. During a normal system shutdown, the filer turns off NFS service, flushes all cached operations to disk and turns off the NVRAM. When the filer restarts after a system failure or power loss, it replays any NFS requests in the NVRAM that have not reached disk.

Using NVRAM to store a log of uncommitted requests is very different from using NVRAM as a disk cache, as some UNIX products do [LS89]. When NVRAM is used at the disk layer, it may contain data that is critical to file system consistency. If the NVRAM fails, the file system may become inconsistent in ways that fsck cannot correct. WAFL uses NVRAM only to store recent NFS operations. If the filer's NVRAM fails, the WAFL file system is still completely self consistent; the only damage is that a few seconds worth of NFS operations may be lost.

3 Logical Backup

The BSD dump utility has been available in various forms for nearly twenty years. In that time, the format has remained fairly constant, and the utility has been ported to platforms ranging from Solaris to Linux. One of the benefits of the format has been the ability to cross-restore BSD dump tapes from one system to another. Even if the utility did not already exist on the platform, such a utility could be written or ported since the format is publicly known.

The dump format is inode based, which is the fundamental difference between dump and tar or cpio. The dump format requires that all directories precede all files in the backup. Both directories and files are written in ascending inode order, assuming that inode #2 is the root of dump, not necessarily of the original file system. Directories are written in a simple, known format of the file name followed by the inode number. Each file and directory is prefixed with 1KB of header meta-data. This data includes: file type, size, permissions, group, owner, and a map of the 1KB holes in the file. The tape itself is prefixed with two bitmaps describing the system being dumped. The first map indicates which inodes were in use in the dumped subtree at the time of the dump. The second map indicates which inodes have been written to the backup media. The first map helps show which files have been deleted between incremental dumps. The second map verifies the correctness of the restore.

The format leads to a fairly simple dump utility. The dump runs as a four phase operation. A user specifies the incremental level of a dump of some subset of the file system. Since dump is file based, the incremental dump backs up a file if it has changed since the previously recorded backup - the incremental's base. A standard dump incremental scheme begins at level 0 and extends to level 9.

Once the dump level and path have been selected, the dump enters Phase I. This phase is basically a tree walk, marking the map with the used and to-be-dumped file inodes. Phase II then marks the directories between the root of the dump and the files selected in Phase I. These directories are needed for restore to map between file names and inode numbers. The final two phases of dump write out the directories and files in inode order, respectively.

Restore reads the directories from tape into one large file. It uses this to create a desiccated file system. That is to say, it tracks the offsets of the beginning of each directory in this file. So, when a user asks for a file, it can execute its own namei (that part of the kernel that maps file names to inodes), without ever laying this directory structure on the file system. This saves quite a bit of space. The reasoning behind this procedure revolves around the ability of restore to reclaim a subset of the data that is on tape. Thus, to write the full directory structure on the system is a bad idea, especially if it is low on disk space or inodes. Restore calculates which files need to be extracted. It then begins to lay the directories and files on the system. After the directories and files have been written to disk, the system begins to restore the directories' permissions and times. Note that it couldn't do this when creating the directories, since creating the files might have failed due to permission problems and definitely would have affected the times.

The primary benefits of a logical backup and restore utility can be traced to the use of the underlying file system. A user can back up a subset of a data in a file system, which can save time and backup media space. Furthermore, logical backup schemes often take advantage of filters - excluding certain files from being backed up. Again, one saves media space and time. Logical backups enable fairly easy stupidity recoveries. If a user accidentally deletes a file, a logical restore can locate the file on tape, and restore only that file. A logical backup is extremely resilient to minor corruption of the tape, or mistakes on the part of the backup utility. Since each file is self-contained, a minor tape corruption will usually affect only that single file. A mistake by the backup utility may result in some number of files being omitted, but much of the system should still be accessible on the tape. Since each file's data is grouped together, a logical backup stream tends to presuppose little or no knowledge of the source file system.

Not surprisingly, the weaknesses of a logical backup and restore utility can be traced to the use of the underlying file system. For basic operations, the utilities must use the file system, adding additional overhead. The file system's read-ahead and caching policies may not be optimized for backup. Concurrently, the backup's use of the file system may have serious impact on other users. Performance is often a very serious issue. The same concerns apply to a restore. For each file, metadata and data need to be written. Most file systems are not optimized for the data access patterns of a restore. Often, then we see the performance being traded for functionality. In some cases, the new functionality attempts to allay the performance problems. It is more often than not, unsuccessful.

Numerous formats and programs exist within the space of logical backup and recover utilities. The BSD dump utility, an inode based program that produces a commonly known output stream garners the advantages and weaknesses inherent in its design. The benefit of any well-known format is that the data on a tape can usually be easily restored on a different platform than that on which it was dumped. If a "dump" utility does not exist on the platform of choice, one can always port a different platform's dump. While certain attributes may not map across the different file systems, the data itself should be sound. Dump's advantages over other well-known formats, such as tar and cpio come from the ease of running incremental backups. Since dump maps inodes, each dump stores a picture of the file system as it looked at the time of the dump. Thus, an incremental dump not only saves those files that have changed, but also easily notes files that have been moved or deleted.

The weaknesses of dump arise from the inherent nature of being a well-known, inode based format. To create an output stream in a well-known format requires a potentially expensive conversion of file system metadata into the standard format. Furthermore, features of the file system that are not accounted for in the well-known format must either be lost or hidden in the stream. As people continually extend a well-known format to incorporate proprietary structures, such extensions can eventually conflict. These conflicts can harm or eliminate cross-platform advantages. This weakness plagues all well-known formats. Dump is further weakened by not being a completely self-identifying format. While this weakness does not exhibit itself during a backup, the fact that BSD restore needs to create a map of file-system on tape significantly slows down data recovery. Furthermore, the complexity of the operation increases the probability of error, and the consumption of system resources. Finally, since dump reads files in inode order, rather than by directory, standard file system read-ahead policy may not help, and could even hinder dump performance. Unnecessary data will continually be prefetched and cached. Dump enables cross-platform restores and accurate incremental backups. However, for these

advantages it adds a great deal of complexity in restoring systems and, as always, a well-known format can rapidly turn into a variety of proprietary formats that no longer work together.

Unlike the versions of BSD dump that run on Solaris, BSD OS, or Linux, the Network Appliance dump is part of the system's kernel. Since the architecture of the system does not allow for a user-level, the system was designed with dump as an integral part of the microkernel. The primary benefit to this approach is performance. For example, other dump utilities cross the user-kernel boundary to read data from files, and then to write the data to the tapefile. Network Appliance has implemented a no-copy solution, in which data read from the file system is passed directly to the tape driver. Unix versions of dump are negatively affected by the file system's read-ahead policy. Network Appliance's dump generates its own read-ahead policy. Similar fast paths have been set up for restoring data. The Network Appliance version of restore enjoys significant performance improvements that are only possible because it is integrated into the kernel and runs as root. Most notable among these is that while the standard restore accesses files and directories by name, which requires it to be able to construct pathnames that the file system can parse, the Network Appliance version directly creates the file handle from the inode number which is stored in the dump stream. In addition, since it runs as root, it can set the permissions on directories correctly when they are created and does not need the final pass through the directories to set permissions that user-level restore programs need.

Functionally, Network Appliance's dump and restore also add benefit to the standard program. By using snapshots, dump can present a completely consistent view of the file system. This not only helps users, but it obviates many consistency checks used by BSD restore on other systems. This enhances performance and simplifies restores. As discussed before, companies tend to extend the format in ways to back up attributes not included in the basic model. For Network Appliance, these attributes include DOS names, DOS bits, DOS file times, and NT ACLs created on our multi-protocol file system. None of these extensions break the standard format.

Still, there are some difficulties that arise due to being integrated in the kernel. First, since there is no "user level" only a person with root access to the filer can run restore. This is a disadvantage compared to the standard dump, restore, and tar utilities on Unix. The filer also does not support the interactive restore option due to limitations that arise from integrating restore into the kernel. On the whole, however, the benefits make the Network Appliance dump and restore a more popular option with users than mounting the file system onto a Unix machine, and using a Unix version of dump and restore.

4 Physical Backup

In its simplest form, physical backup is the movement of all data from one raw device to another; in the context of file system backup the source devices are disks and the destination devices may include disk, CD-Rom, floppy, Zip drives, and of course, tape.

It is a straightforward extension to the simple physical backup described in the preceding paragraph to interpret the file system meta-data sufficiently to determine what disk blocks are in use and only back those up. All file systems must have some way of determining which blocks are free, and the backup procedure can interpret that information to only back up the blocks that are in use. Naturally, this requires that the block address of each block written to the backup medium be recorded so that restore can put the data back where it belongs.

The primary benefits of a physical backup scheme are simplicity and speed. It is simple because every bit from the source device is copied to the destination; the format of the data is irrelevant to the backup procedure. It is fast because it is able to order the accesses to the media in whatever way is most efficient. There are a number of limitations to physical backup, however. First, since the data is not interpreted when it is written, it is extremely non-portable; the backup data can only be used to recreate a file system if the layout of the file system on disk has not changed since the backup was taken. Depending on the file system organization, it may even be necessary to restore the file system to disks that are the same size and configuration as the originals. Second, restoring a subset of the file system (for example, a single file which was accidently deleted) is not very practical. The entire file system must be recreated before the individual disk blocks that make up the file being requested

can be identified. Third, the file system must not be changing when the backup is performed, otherwise the collection of disk blocks that are written to disk will likely not be internally consistent. Finally, the coarse grained nature behind this method leads to its own difficulties. Because file system information is not interpreted by the backup procedure, neither incremental backups nor backing up less than entire devices is possible. A raw device backup is analogous to a fire hose. Data flows from the device simply and rapidly — but it is all the you can do to hold the hose. Finer grained control is generally impossible.

These negative aspects of physical backup have until now caused it to have very limited application. Physical backup is used in a tool from BEI Corporation that addresses the problem of restoring NT systems after catastrophic failure of the system disk [Edw97]. Two large experiments at getting Terabyte per hour backup performance mention the use of raw (or device) backup and contain performance measures that validate the intuition that it obtains extremely good performance [CCC98, Sun97].

4.1 WAFL Image Dump

WAFL's snapshot facility addresses several of the problems with physical dump identified above. First, because a snapshot is a read-only instantaneous image of the file system, copying all of the blocks in a snapshot results in a consistent image of the file system being backed up. There is no need to take the live file system off line.

Second, snapshots allow the creation of incremental physical dumps. Since each snapshot has a bitmap indicating which blocks are in the snapshot, we can easily determine which blocks are new and need to be dumped by considering the sets of blocks included in the two snapshots. Suppose that our full backup was performed based on a snapshot called A, and we have just created a new snapshot called B from which we wish to make an incremental image dump. Table 1 indicates the state of blocks as indicated by the bits in the snapshot bit planes. We must include in the incremental dump every block that is marked as used in the bit plane corresponding to B but is not used the bit plane corresponding to A, and no other blocks. Alternatively, we can consider the bitmaps to define sets of blocks, we wish to dump the blocks in the set:

$$B - A$$

Higher level incrementals are handled in the same manner. A level 2 incremental whose snapshot is C and which is based on the full and level 1 incremental described above needs to include all blocks in:

$$C - B$$

since everything in A that is also in C is guaranteed to be in B as well. These sets are trivial to compute by looking at the bit planes associated with all of the relevant snapshots.

Of course, a block based dump does not want to be too closely linked to the internal file system, or you lose the advantage of running at device speed. Therefore, image dump uses the file system only to access the block map information, but bypasses the file system and writes and read directly through the internal software RAID subsystem. This also enables the image dump and restore to bypass the NVRAM on the file system, further enhancing performance.

Finally, the block based dump allows for some features that the file-based dump cannot provide. Unlike the logical dump, which preserves just the live file system, the block based device can backup all snapshots of the system. Therefore, the system you restore looks just like the system you dumped, snapshots and all. Unfortunately, the other two limitations detailed above, portability and single file restore, seem to be fundamental limitations of physical backup and are not addressed by WAFL. We'll return to the single file restore issue in Section 6.

5 Performance

This section reports the results of our experiments to quantify the achievable performance of the physical and logical backup and recovery strategies. We begin by reporting the performance of the most straightforward backup and recovery plans, and then proceed to measure the speedup achievable by utilizing multiple tape devices.

All of our measurements were performed on eliot, a Network Appliance F630 file server. The filer con-

Bit plane A	Bit plane B	Block state
0	0	not in either snapshot
0	1	newly written - include in incremental
1	0	deleted, no need to include
1	1	needed, but not changed since full dump

Table 1: Block states for incremental image dump

sists of a 500 MHz Digital Alpha 21164A processor with 512 MBytes of RAM and 32 MBytes of NVRAM. 56 9 GByte disks are attached via a Fibre Channel adapter. 4 DLT-7000 tape drives with Breece-Hill tape stackers are attached through dedicated Differential Fast Wide SCSI adapters. At the time the experiments were performed, the filer was otherwise idle.

The 481 GBytes of disk storage are organized into 2 volumes: home which contains 188 GBytes of data and rlse which contains 129 GBytes of data. The home volume consists of 31 disks organized into 3 raid groups and the rlse volume consists of 22 disks organized into 2 raid groups. Both the home and rlse file systems are copies (made using image dump/restore) of real file systems from Network Appliance's engineering department.

5.1 Basic Backup / Restore

Table 2 reports our measurements of backing up and restoring a large, mature[1] data set to a single DLT-7000 tape drive. In this configuration both logical and physical dump obtain similar performance, with physical dump getting about 20% higher throughput. This is because the tape device that we are using (a DLT-7000) is the bottleneck in the backup process. Note however the significant difference in the restore performance. This can be primarily attributed to image restore's ability to bypass the file system and write directly to the disk, while logical restore goes through the file system and NVRAM[2].

We can look at the resource utilization of the filer while backup and restore are proceeding to learn something about how logical and physical backup are likely to scale as we remove the bottleneck device. Table 3 indicate the time spent in various stages of backup and restore as well as the CPU utilization during each stage. It is worth noting the variation in CPU utilization between the two techniques. Logical dump consumes 5 times the CPU resources of its physical counterpart. Logical restore consumes more than 3 times the CPU that physical restore does.

One way to reduce the time taken by the backup procedure is to backup multiple file system volumes concurrently to separate tape drives. The resource requirements of both logical dump and physical dump are low enough that concurrent backups of the home and rlse volumes did not interfere with each other at all; each executed in exactly the same amount of time as they had when executing in isolation.

5.2 Backup and Restore to Multiple Tapes

Our next set of experiments are designed to measure how effectively each dump strategy can use parallel backup tape devices. For the logical strategy, we cannot use multiple tape devices in parallel for a single dump due to the strictly linear format that dump uses. Therefore we have separated the home volume into 4 equal sized independent pieces (we used quota trees, a Network Appliance construct for managing space utilization) and dumped them in parallel. For physical dump, we dumped the home volume to multiple tape devices in parallel. Tables 4 and 5 report our results.

Overall, logical dump managed to dump the 188 GByte home volume to 4 tape drives in 2.7 hours, achieving 69.6 GBytes/hour, or 17.4 GBytes/hour/tape. Physical backup to 4 drives completed in 1.7 hours, achieving 110 GBytes/hour, or 27.6 GBytes/hour/tape.

[1] A mature data set is typically slower to backup than a newly created one because of fragmentation: the blocks of a newly created file are less likely to be contiguously allocated in a mature file system where the free space is scattered throughout the disks.

[2] There is no inherent need for logical restore to go through NVRAM as it is simple to restart a restore which is interrupted by a crash. Modifying WAFL's logical restore to avoid NVRAM is in the works.

Operation	Elapsed time (hours)	MBytes/second	GBytes/hour
Logical Backup	7.4	7.2	25.0
Logical Restore	8.1	6.3	22.8
Physical Backup	6.2	8.6	30.3
Physical Restore	5.9	8.9	32.0

Table 2: Basic Backup and Restore Performance

Stage	Time spent	CPU Utilization
Logical Dump		
Creating snapshot	30 seconds	50%
Mapping files and directories	20 minutes	30%
Dumping directories	20 minutes	20%
Dumping files	6.75 hours	25%
Deleting snapshot	35 seconds	50%
Logical Restore		
Creating files	2 hours	30%
Filling in data	6 hours	40%
Physical Dump		
Creating snapshot	30 seconds	50%
Dumping blocks	6.2 hours	5%
Deleting snapshot	35 seconds	50%
Physical Restore		
Restoring blocks	5.9 hours	11%

Table 3: Dump and Restore Details

Operation	Elapsed time	CPU Utilization	Disk MB/s	Tape MB/s
Logical Backup				
Mapping	15 minutes	50%	5.0	0.0
Directories	15 minutes	40%	15.0	12.0
Files	4 hours	50%	12 - 20	12 - 20
Logical Restore				
Creating files	1.25 hours	53%	6.0	0.0
Filling in data	3.5 hours	75%	16.0	16.0
Physical Backup				
Dumping blocks	3.25 hours	12%	17	17
Physical Restore				
Restoring blocks	3.1 hours	21%	17.4	17.4

Table 4: Parallel Backup and Restore Performance on 2 tape drives

Operation	Elapsed time	CPU Utilization	Disk MB/s	Tape MB/s
Logical Backup				
Mapping	5 minutes	90%	9.5	0.0
Directories	7 minutes	90%	27.0	12.0
Files	2.5 hours	90%	21	21
Logical Restore				
Creating files	0.75 hours	53%	9.0	0.0
Filling in data	3.25 hours	100%	18.0	18.0
Physical Backup				
Dumping blocks	1.7 hours	30%	31	31
Physical Restore				
Restoring blocks	1.63 hours	41%	32	32

Table 5: Parallel Backup and Restore Performance on 4 tape drives

5.3 Summary

As expected, the simplicity of physical backup and restore means that they can achieve much higher throughput than logical backup and restore which are constantly interpreting and creating file system meta data. The performance of physical dump/restore scales very well; when physical restore hits a bottleneck it is simple to add addition tape drives to alleviate the bottleneck. Eventually the disk bandwidth will become a bottleneck, but since data is being read and written essentially sequentially, physical dump/restore allows the disks to achieve their optimal throughput. Logical dump/restore scales much more poorly. Looking at the performance of 4 parallel logical dumps to 4 tape drives (Figure 5) we see that during the writing files stage the CPU utilization is only 90% and the tape utilization is under 70% (as compared to that achieved by physical dump). The bottleneck in this case must be the disks. The essentially random order of the reads necessary to access files in their entirety achieves highly sub-optimal disk performance.

6 The Future

Physical backup and restore has been ignored for a long time. With file systems continuing to increase in size exponentially, the ability of a physical backup strategy to scale with increasing disk bandwidth makes it an interesting strategy. In the WAFL context, where snapshots provide the ability to discover recently changes blocks very efficiently, incremental image backups become a possibility. We are working on an implementation of incremental backup and will soon be able to answer questions about its performance. The image dump/restore technology also has potential application to remote mirroring and replication of volumes.

7 Conclusion

This paper has examined the limitations and performance of both the logical and physical backup and restore strategies. Logical backup has the advantage of a portable archival format and the ability to restore single files. Physical backup and restore has the advantages of simplicity, high throughput, and the ability to support a number of data replication strategies. Both have much to contribute to a complete file system backup strategy, but that the ability of physical backup/restore to effectively use the high bandwidths achievable when streaming data to and from disk argue that it should be the workhorse technology used to duplicate our constantly growing data sets and protect them from loss.

References

[CAK+92] Sailesh Chutani, Owen T. Anderson, Michael L. Kazar, Bruce W. Leverett, W. Anthony Mason, and Robert N. Sidebotham. The Episode file system. In *Proceedings of the Usenix Winter 1992 Technical Conference*, pages 43–

60, Berkeley, CA, USA, January 1992. Usenix Association.

[CCC98] Sandeep Cariapa, Robert Clard, and Bill Cox. Origin2000 one-terabyte per hour backup white paper. SGI White Paper, 1998. http://www.sgi.com/Technology/teraback/teraback.html.

[CVK98] Ann L. Chervenak, Vivekenand Vellanki, and Zachary Kurmas. Protecting file systems: A survey of backup techniques. In *Proceedings of the Joint NASA and IEEE Mass Storage Conference*, March 1998.

[Edw97] Morgan Edwards. Image backup and NT boot floppy disaster recovery. *Computer Technology Review*, pages 54–56, Summer 1997.

[GBD96] R. J. Green, A. C. Baird, and J. C. Davies. Designing a fast, on-line backup system for a log-structured file system. *Digital Technical Journal of Digital Equipment Corporation*, 8(2):32–45, October 1996.

[HLM94] Dave Hitz, James Lau, and Michael Malcolm. File system design for a file server appliance. In *Proceedings of the 1994 Winter USENIX Technical Conference*, pages 235–245, San Francisco, CA, January 1994. Usenix.

[LS89] Bob Lyon and Russel Sandberg. Breaking through the nfs performance barrier. *Sun Technical Journal*, 2(4):21–27, Autumn 1989.

[MJLF84] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry. A fast file system for UNIX. *ACM Transactions on Computer Systems*, 2(3):181–197, August 1984.

[Qui91] Sean Quinlan. A cached worm file system. *Software—Practice and Experience*, 21(12):1289–1299, December 1991.

[Sun97] High-speed database backup on Sun systems. Sun Technical Brief, 1997.