Backup Software Applications for Filesystems

650046686

November 22, 2017

Abstract

This paper surveys backup techniques for protecting filesystems. First, we discuss the two primary backup schemes: the device-based and file-based approaches. The device-based strategy writes the complete filesystem to the backup storage, and the file-based strategy allows for writing individual disk blocks of the source filesystem to the backup storage media. The Write-Anywhere-File-Layout will provide a suitable test-bed for the evaluation of these to schemes, as it implements both schemes with the same attention to detail. Next, we discuss the two primary algorithms for backup - full and incremental - and the techniques for on-line backup, which is an approach that allows end-users to access a filesystem during a backup cycle. These features include detection, system locking and capturing copy-on-write snapshots of the complete filesystem. We discuss important features of backup systems in the context of the AMANDA Network Backup Manager and, and present necessary future advances in order to resolve impending challenges of existing backup systems.

I certify that all material in this dissertation which is not my own work has been identified.

1 Introduction

In computing, a filesystem is a method for controlling how data is stored within a given storage medium to provide intuitive access for an end-user. As a central component of an operating system, filesystems have received a great deal of attention from both the industrial and academic communities. As filesystems expand, it becomes more and more difficult to ensure that all data is safely stored. As a result, backup systems have seen a similar increase in attention received from the industrial community. The cause for concern stems from the realisation that access times and data rates of disk drives will increase at reduced rates - that is, at approximately 20% per annum[3]. On the other hand, as the capacity of new disk drives continues to increase with a higher rate of over 50% per annum[3], networked computing environments will expand to contain terabytes of disk storage. The disaccord in progression of these two fundamental components of data transfer indicates that the time to read contents of a disk drive and write them to a backup device will take increasingly longer. Over time, this trend suggests that traditional backup schemes will be too slow, so protecting rapidly expanding data repositories from software errors that may corrupt a filesystem, disk or other hardware failures, natural disasters, or loss of data by user error continues to be a growing challenge. The aim of my project is to build a backup software application capable of incremental backups of an entire filesystem. The final application will be distributed with the Linux operating system and will employ some of the backup schemes and techniques discussed throughout this paper.

To adhere to its rudimentary purpose, backup software is designed to enable restoration of lost data in any case of the aforementioned incidence. However, as will be discussed throughout this paper, the most advanced of todays backup applications often go above and beyond the most basic features to satisfy onerous software specific requirements and to stay relevant in todays cut-throat tech industry. Data compression, file restore, and remote backup are only a few of the powerful features offered by these products[3][27]. The use of backup systems remains important not only for data recovery, but also for improving system resource utilisation and the overall data maintenance process[19], especially in an environment where backups are performed regularly[10][18]; this encourages a continuous search for improved backup products. This paper surveys current backup techniques as a prelude to establishing desirable features of future backup systems. Initially, we compare the performance and evaluate the design flaws of the file-based and device-based backup schemes in the context of Network Appliances Write Anywhere File Layout, which provides an intriguing test-bed for comparing and contrasting the two utilities. Section 3 evaluates existing backup algorithms. Section 4 discusses the features commercial backup schemes in the context of the Amanda Backup Manager, and identifies the needs of future backup systems. Finally, Section 5 concludes.

2 Design for backup systems

In evaluating backup strategies, a range of end-user expectations must be met. A system administrator can hope to never have a reason to look at backup data, and in fact, in normal operation this is the case for majority of collected data[18]. Therefore, it is favourable to maximise the speed at which data can be backed up, and to minimise the CPU and disk resources used in performing backup[18]. Unfortunate situations abound regarding system administrators trying to restore filesystems after a disaster occurs, only to find out that the backup is not readable; content saved on a storage medium is often kept for a long period of time, so robustness is a critical requirement in the design of a backup system[18]. Hence, both to provide filesystem history and improved resilience to disasters, it is vital that data storage format be archival in nature[18]. That is, the contents of the storage medium should be recoverable even if the operating system, hardware, or backup software has been altered since the backup was performed.

On the restore side of a backup system, there are two common types of recovery: disaster recovery and user error recovery[3][18]. Disaster recovery manifests itself as a request to recover an entire filesystem that is lost due to software, hardware, or device failure[18]. Since the disaster recovery solution requires a full restore of data onto a new device, it can also be utilised for reasons other than recovery, such as migration of data from one device to another (e.g. the dd command for a Unix system)[22]. User error recovery comes into play when individual files of a filesystem have been overwritten or deleted, commonly by user error[18]. In the following section, we evaluate two primary backup schemes that enable the recovery features discussed above: the file-based and device-based backup schemes.

2.1 File-based vs. device-based filesystem backup

A file is made of logical blocks, which are commonly of fixed size of approximately 8 kilobytes[3][7]. Each respective logical block of a file is stored on a contiguous disk block, but distinct logical file blocks do not necessarily have to be stored contiguously on a physical disk block[3][5]. Operating systems that belong to the UNIX family map logical block addresses to their respective physical addresses on disk by using an index node structure (also known as inode)[5][22], which store pointers to physical disk blocks[22]. Although the terminology used in other operating systems may be a bit different to that in the UNIX world, regarding an integer that identifies a file, some Windows API and NTFS expose a similar concept of unique file IDs[26].

Backup software can leverage either files or physical disk blocks, which are known as the file-based and device-based schemes, respectively, and are the two primary solutions to the backup and restore components of a backup system[18]. A file-based (or logical) system interprets filesystem metadata and determines which files and directories need to be copied to the backup device (details on use of metadata are explored in Section 2.2)[23]. Since the file-based strategy understands filesystem structure, the system is able to discover the files that need to be duplicated and write them to a backup device in canonical representation, which is highly desired for end-users who may not know much about the filesystem structure[18]. A file-based system reads the physical blocks of each file by traversing the pointers stored in each inode[18]. Thereafter, the backup software can store each file contiguously, which enables rapid single file recovery[3]. Unfortunately, writing each file contiguously to a backup medium means that more disk seek operations are required, which results in a decreased disk throughput and increased disk overhead[3][18]. These extra costly seek operations result in slower backup[3][18]. Another limitation of file-based backup scheme is that even minor a changes to a file demands that the entire file be backed up again[23].

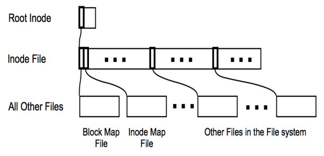
By contrast, device-based (or physical) backup systems copy an entire filesystem onto the backup medium with minimal or no interpretation of the target filesystem structure[23]. Device-based backup has initially been developed to migrate data from one device to another[3], but has since evolved and been used as a fully functional backup strategy by Digital[18]. One of the advantages of duplicating an entire physical medium is that even filesystem attributes that may not be representable in the standard archival format are copied[3]; examples of such attributes are filesystem configurations, hidden files, or snapshots, which are further discussed in section 2.2.1[18]. Neglecting the file structure when copying disk blocks onto a backup medium allows for fewer necessary seek operations, which improves overall backup software performance[3]. However, since a device-based approach may not store files contiguously on backup medium, it has a slower restore than the file-based approach[23]. To enable file recovery, device-based backups must keep information on how files and directories are stored on disks to link blocks on the backup medium with their respective files[18]. As a result, device-based programs often vary in implementation for different filesystems and are, therefore, seldom portable[3][18]. On the other hand, the file-based software utilities like tar contain contiguous files, which means they are more portable because the concept of files is considerably universal[3]. Another disadvantage of the device-based approach is that it may cause data inconsistencies[3][18]. Prior to writing to disk, it is possible that an operating system kernel will buffer write data. Typically, device-based backup schemes neglect this file cache data when traversing disk blocks and back up older versions of files, which will undeniably introduce data inconsistencies[3]. By contrast, a file-based backup scheme will first check the file cache and back up the most up to date versions of the files[3].

Recent years have seen a discussion about the merits of file-based versus device-based backup schemes. However, there are very few systems in which the two schemes have been implemented with comparable attention to detail[18]. Network Appliances Write Anywhere File Layout (WAFL) filesystem implements both the file-based and device-based schemes; WAFL utilises snapshots that enable both the file-based and device-based dump to backup a consistent image of the filesystem (snapshots are read-only images of a filesystem at single point in time, and are further explored in Section 2.2.1)[18]. Section 2.2 discusses the WAFL file layout, which is important in our comprehensive evaluation of the file-based and the device-based backup strategies. Sections 2.3 and 2.4 compare the performance and design flaws of the file-based approach as embodied in WAFLs modified BSD dump and WAFLs physical dump.

2.2 WAFLs image and BSD-style dump and restore

A WAFL file layout can be best interpreted as a tree of blocks, as shown in Figure 1[9][18]. At the root of this tree we can find the root inode, which is a unique inode that describes the inode file[9]. The inode file stores the inodes that, in turn, describes all other files in the filesystem[9][22]. The leaves of the tree represent the data blocks of all the files in the filesystem[9].

Figure 1: The WAFL Filesystem[9]

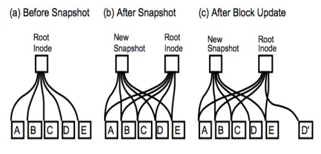


WAFL's device-based backup approach is known as image restore (or image dump)[18] It operates by leveraging WAFLs snapshot implementation to rapidly identify those disk blocks that store the data that needs to be dumped[18]. Furthermore, the bookkeeping required to enable copy-on-write (further discussed in Section 2.2.2) also supports incremental image restores[6] - that is, solely those disk blocks that have been modified from the time of last image dump are included in the next incremental dump[6][10]. The image dump utility achieves improved restore performance by neglecting a lot of WAFLs filesystem constructs when backing up and restoring data[18]; hence, it is can classified as a strictly device-based implementation. On the other hand, WAFLs file-based backup approach, the BSD-style dump and restore utility, has been designed to exploit the features of the WAFL file layout, rather than bypassing them[18]. The BSD-style dump and restore takes advantage of WAFL to access data that will help identify the contents that need to copied[18].

On the device-based backup side, WAFL's image dump technology enables more intriguing mirroring and replication opportunities[9]. On the file-based backup side, companies can utilise the restore utility to achieve a makeshift Hierarchical Storage Management (HSM) system[18]; high performance RAID systems can duplicate data on cheaper backup file servers, which eventually backup data to a storage medium[18]. Since the image and BSD-style dumps can be implemented with comparable degree of completeness, WAFL provides an intriguing test-bed for comparing and contrasting the performance of the device-based and file-based schemes, as will be demonstrated in Sections 2.3 and 2.4. However, first we discuss snapshots and the copy-on-write resource management technique, which are features of WAFL that are highly relevant in our discussion of backup strategies.

2.2.1 The snapshot facility

Figure 2: WAFL's snapshot facility[9]



Snapshots are frozen, read-only copies that provide a consistent view and access to earlier versions of a filesystem, and are a distinguishing feature of WAFL's write anywhere approach[14][20]. WAFL performs a snapshot by making a duplicate copy of the root data structure, and updating the block allocation information (see Figure 2)[9]; this operation takes just a few seconds and with very little disk I/O[20] Snapshot performance is vital because WAFL performs a snapshot every few seconds to eliminate the need for time-consuming recovery and filesystem consistency checks after unclean system shutdowns, such as power loss or system failure[3][18]. 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 require significant additional disk space until a file in the active filesystem is modified or deleted[18]; the WAFL approach leverages the copy-on-write technique (detailed in Section 2.2.2) to minimise the disk space that snapshots consume by avoiding duplication of disk blocks that are the same in a snapshot as in the active filesystem[20].

Snapshots in general may be utilised as an on-line backup utility that allows users to restore their own files[9]. Each snapshot can be made automatically at prescheduled times defined by an end-user, but can also be created or deleted manually[14]. A recovery from user error or other disastrous incidence can be facilitated if frequent incremental backups are made[10][14]. For example, systems such as Andrew and Time Machine can make either full or incremental backups from one of many snapshots[3] - Time Machine can capture an array of hourly or weekly local snapshots, which a time schedule adjustable in the graphical user interface[4]

2.2.2 Copy-on-write

Copy-on-write is a resource-management method used to efficiently decide a copy operation - that is, if a resource must be duplicated but has not been modified, it is not required that a new resource is created as the duplicate can be deferred to the original write[6]. Modifications must still be recorded, hence the technique: the data represented by a single snapshot can be shared among the latest snapshot and the first copy of a filesystem, which reduces the resource consumption of unchanged copies at the price of adding minimal overhead to resource-modifying operations[20].

WAFL enables the copy-on-write technique implemented by snapshots to minimise the disk space used by each snapshot of a filesystem, such that only blocks that have been changed in subsequent snapshots to the original copy are cloned[6][20].

2.3 Design issues for file-based systems in context of WAFLs modified BSD dump

WAFLs file-based backup utility has been around for almost twenty years[18]. In that time, the BSD dump has been ported to platforms like Solaris and Linux[18]. One of the prominent advantages of the BSD dump format has been its exceptional ability to cross-restore data from one environment to another[18]. The BSD dump format operates on inodes, which is the distinguishing factor from archiver utilities such as *cpio* or *tar*[16] [18][24]. It the backup, every directory must precede its files, and both files and directories are stored in ascending inode order[18][22]. Directories are stored in a common format of the file name followed by the inode number[22]. Every backed up file and directory is pre-fixed with 1KB of header meta-data, which includes: file type, a map of the 1KB holes in the file, permissions, group, size, and the owner[18]. The backup itself includes two bitmaps that describe the system at the time of backup[18], which allows for a format that facilitates what is a relatively simple dump utility[9]. The first map identifies the inodes being used in the dumped subtree during the time of the dump, and helps in determining which files have been deleted between incremental dumps[18]. The second map identifies the inodes written to the backup storage device, and help verify the degree of correctness of a restore[18].

The main merits of the file-based backup utility can be accredited to its ability to use the filesystem structure to its advantage[23]. In a file-based back approach, an end-user can chose to backup only a subset of data in a filesystem, which saves significant backup time and backup device space[23]. In that sense, a file-based backup is able to exploit to use of filters, such as omitting given files from backup. Once again, this saves device space and reduces backup time. Furthermore, file-based backups allows for relatively easy user error recoveries; if the system administrator deletes a file by accident, a file-based restore operation can locate the given file on disk and restore that single file[3]. A file-based approach is tremendously resilient to even small corruption of the disk, or errors caused by the backup utility of the software - that is, since each individual file is self-contained, minimal disk corruption will most of the time corrupt only that file. An error on the backup side of the software may cause a number of files being excluded from backup, but a majority of the filesystem copy will still be successfully stored on the disk[18] .

Since the data of each distinct file is grouped together, a file-based backup stream will have to tendency to assume minimal or no knowledge of the underlying filesystem[18]. Therefore, it is no surprise that the primary disadvantage of a file-based backup and restore utility can be accredit to the use of the source filesystem[18]. For instance, the dump utilities must rely on the source filesystem for basic operation, which accounts for additional overhead[18]. It may happen that a filesystems caching policies and read-ahead are not optimised for backup. Consequently, the backup softwares use of a certain filesystem may have considerable impact on user experience. This means that performance is often a significant issue. Similar problems concern restore operations. For example, data and metadata must be written for each stored file[18]. However, many filesystems are not optimised for the data access patterns of a restore operation[18]. In such cases, we observe a performance tradeoff for functionality. Although the improved functionality often attempts to alleviate the performance degradation, it is often an unsuccessful attempt[18].

2.4 Design issues for WAFLs device-based dump strategy

A device-based backup is the complete migration of data from one device to another[23]; with regard to filesystem backup, the source media are disks and the destination media may include solid state storage media such as flash memory, optical storage media such as CDs, or remote backup services[18]. It is a simple augmentation to the traditional device-based backup scheme to be able to understand filesystem meta-data enough to identify which disk blocks are in use and should therefore be backed up. All filesystems must include a method for identifying what blocks are free[18], which a backup procedure can use to back up only those blocks that are in use; of course, this requires that the block address of every block stored on a backup storage medium is recorded so that the restore operation can recover data to the correct address[18][23].

The primary advantage of using the device-based backup schemes stems from its simplicity[3]; all data from the source device is migrated to the destination device, so the backup procedure can neglect the format of the data[18]. Another advantage is speed; the device-based backup scheme can place the accesses to the source device in the order it deems as most efficient[18]. There are several limitations to device-based backups, however.

First, because data is not interpreted when stored, it is not portable[3]. Only if a filesystem layout of the source disk has not been modified since the last backup can the backup data be used to recreate a filesystem[18]. In fact, it may even be required to recover a filesystem to a disk of same configuration and size as the original depending on the underlying filesystem organisation[18].

Second, to recover a small subset of the filesystem, such as an individual file that was deleted due to user error, is highly impractical[23]. The complete filesystem must then be recovered in order to identify the individual disk blocks that constitute the requested file[18].

Third, the disk blocks stored on disk will most likely be internally inconsistent if a filesystem is modified while performing a backup[18]. It could be said that the indelicate nature of the device-based backup approach is a cause of its own problems; since minimal to no interpretation of filesystem information occurs during a backup procedure, both backing up subsets of a filesystem and incremental backups are unavailable features of a device-based approach. A garden hose provides an appropriate analogy for raw device-based backup. Although data flows from the source device to the target storage medium at high speed and with operational simplicity, the control of this flow is limited, since all a person can do is to hold the hose shut - any finer grained control of the data flow is very difficult. WAFL's image dump facility addresses some of the issues with device-cased dump identified above, which will be further explored in Section 2.4.1.

2.4.1 WAFLs image dump

WAFLs image dump resolves some of the problems of the device-based approach by enabling incremental snapshots of a filesystem[9]. First, it should reiterated that since snapshots capture instantaneous clones of a filesystem, copying all blocks of a snapshot can produce a consistent image of the filesystem at hand[17]. Since each snapshot contains a bitmap that stored information on which blocks have been included in a snapshot, it is possible to easily establish which blocks have not been included in the last back up by comparing the two sets of blocks in the two snapshots[9].

We consider a full backup made based on a snapshot A. Snapshot B is created from which we desire to perform an incremental image dump. Figure 3 demonstrates the state of blocks as defined by the bits found in the snapshot bit planes. The objective is to perform an incremental dump for every block that is used in bit plane B but is not used in bit plane A, but not for any other blocks. We can also interpret the bitmaps to define sets of blocks, in which case we would perform incremental dump for blocks in the set B A.

Figure 3: Block states[18]

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

An image dump should not be too reliant on the internal filesystem, otherwise the advantage of running at device speed is lost[18]. Hence, a device-based dump exploits the filesystem solely to gain access to the block map information, but bypasses the filesystem constructs and performs reads and writes directly by using the internal software RAID subsystem, which enhances performance[18].

Finally, the device-based dump provides several features that are not available with the file-based dump. While the file-based dump stores only the active filesystem, a device-based device can backup all frozen snapshots of a live filesystem, which allows for a restoration of system that is identical to the one that was dumped[3]. Unfortunately, the two limitations discussed above - single file recovery and portability - continue to be inherent limitations of the device-based backup scheme that are not resolved by WAFL. We'll address the single file recovery problem in Section 4.2, which identifies the needs of future systems.

3 Design issues for backup algorithms

Most backup algorithms experience a tradeoff between the performance of two utilities: the backup operation and the restore operation. At one extreme, an algorithm may be optimised for rapid backup operation by exclusively cloning files that have been modified from the time of last backup, or simply performing periodic incremental backups[27]. A pure incremental approach performs scheduled writes to a backup storage medium for only those files that have been changed since the last backup. The drawback of this fast and simple backup approach is the degradation of performance on the restore side of the backup software[3]. For instance, given a case where files within one directory are changed between backups, a pure incremental algorithm will write these files far apart from each other on the sequential backup media[27]. Thereafter, to recover a directory, restoring sizeable amounts of sequential media may be required[27].

At the other extreme, an algorithm may be optimised for a fast restore utility. The optimisation of a restore operation requires carefully writing data to the backup storage medium - that is, linking directories that are logically associated and backing up all files of the same directory to a single backup storage medium[27]. This type of algorithm can perform scheduled full backups of a filesystem, in which an entire filesystem is written to the backup storage medium[3]. The objective of such an algorithm is to minimise to restore-time of data by reducing the amount of sequential media accessed by a restore operation. Unfortunately, the tradeoff for an algorithm optimised for restore operations is that it necessitates more time and significantly more media for the backup side, which varies depending on the bandwidth restrictions of the sequential tape media, such as the speed at which a disk drive can write data[27]. The following section compares and contrasts the two primary algorithms that represent the approaches detailed above, in order to further our discussion on the current design challenges of backup systems.

3.1 Full vs. Incremental Backup

The easiest method for protecting a filesystem from file corruption and disk failures is to clone all contents of a given system to a backup storage medium, which is known as full backup. One of the inherent advantages of a full backup it can be used to recover an entire filesystem onto a new disk, but also to restore individual files[27]. However, full backups have two significant limitations: writing a copy of a filesystem requires a lot of space on a backup storage medium, and reading and writing a complete filesystem is not very fast[3]. Faster backups can be realised with an incremental backup algorithm, which solely clones files that have been modified from the previous backup. Since, in normal operation, little percentage of files in a filesystem are modified on a daily basis, an incremental algorithm can be exploited to minimise the size of backup[3]. An incremental algorithm is typically designed to make frequent incremental backups supported by infrequent full backups of an entire filesystem[13][8]. Recovering a lost file or filesystem is not as fast with an incremental-based backup system; restoration may necessitate interpreting an array of backup files, starting with the latest full backup of a filesystem and identifying the modification were made in multiple incremental backups[8].

As mentioned above, it is not the case that backup software must chose between the incremental and full backup algorithms, but can combine them to satisfy system requirements and optimise backup at a lower cost[8]. In the past, this has been seen for database systems. When a full backup is performed frequently, all copies of a database can be stored but at high cost[8]. Therefore, a backup technique was developed to ensure that data is safely stored at reduced backup time: an incremental backup, which only copies newly modified files since the last full backup, is executed at fixed intervals between the operation of full backups, which are performed at longer, prescheduled intervals[1][13]. This policy allows for performing the full backup, which has large overhead, fewer times - either at dispersed, prescheduled times or when the number of modified files in filesystem exceeds a predefined threshold value - while performing the incremental backup with small overhead at shorter intervals[8].

Seeking the optimal interval for a full backup will help minimise the expected cost of the database backup. In the context of database system, it seems straightforward that an incremental backup with small overhead should be employed to lessen the overhead of backups, and indeed, this is the case for most massive database systems[8]. However, the overhead of an incremental backup increases proportionally to the number of modified files in the system[8]. Therefore, it is highly beneficial to devise a suitable interval for performing full backups of a system. Although the overall performance of backup programs receives a great deal of attention from the industry community, end-user experience is also highly relevant, especially for commercial backup software. Some backup software demands that a filesystem stays quiescent while a backup is performed, which can harm user-experience and can be bypassed using an on-line backup approach[3][24]; such systems enable continuous access to the filesystem even during backup, which will be discussed in Section 3.2.

3.2 On-line backup

Although on-line backup systems provide improved accessibility to files, they also introduce consistency problems[24]. Consistency checks occur in cases such as when an end-user moves a directory to a different location during backup[3]. In this case, it is possible that the backup program will not find the directory in the filesystem hierarchy and therefore not clone it to the backup storage medium or will find a copy of the directory in many different locations depending on the stage of traversal of the backup program. Other limitations of the on-line backup strategy are file changes, deletions, and transformations during a backup cycle. An example of file transformation is compression, in which the information of a file is compressed, assigned to a different file name, and the new file may even be moved to a different directory - the initial file is then deleted[3]. In the case of file compression, if the backup program has already either traversed the section of the filesystem hierarchy (e.g., tar) or finished the scan phase (e.g., dump), neither the new file nor the original (since it has been deleted) will be included in the backup[3]. In most cases, if a file is removed from the filesystem hierarchy prior to being read by the backup program, it will not be backed up to the backup storage medium[3][24]. In the case of file modification during a backup cycle, the consequences are often unpredictable, which may result in an undesirable combination of new and old data[24].

There are three solutions to the problems with on-line backup: locking, detection, and copy-on-write, which has been discussed in Section 2.2.2[3]. First, system can inhibit file changes or directory movement during a backup cycle by locking files and directories; this is done by disabling the write, link, move and truncate commands[3][22]. However, it is vital to minimise the locking time period, since it can result in reduced filesystem availability[3]. Second, a backup system can compare the new and old states of a filesystem to identify any file and directory changes and movements; if a file that has been modified or moved to a different location was not included in the last backup, the backup software will include it in the next backup cycle[3].

4 Backup systems

As a prelude to this section, we will first discuss three important features of a backup system: concurrent backups, compression, and file restores. A desirable, if not necessary feature of modern backup systems is the ability to perform backups of many filesystem in parallel onto multiple disk drives[3][13]. This is especially relevant for backup of a big collection of networked machines. Most of the systems discussed or mentioned in this paper enable concurrent backups. The Amanda Backup Manager, for example, utilises a temporary holding disk to store backups of multiple filesystems, which are eventually are written to a storage disk at its maximum transfer rate[3][13].

To minimise the required network bandwidth and disk storage usage, backup systems will often provide the system administrators with the option of file files prior to backup[3][11]. Often, the system is able to compress the data at file or client server by exploiting compression software or hardware[3]. In some cases, data can be compressed even before the files are stored written the backup storage medium[3].

Throughout the paper, we have discussed several factors that might influence the speed at which individual files or complete filesystems are restored. Inevitably, file restores will operate slower in incremental backup systems because they first perform a full backup, and apply any modification from the incremental backups in a calculated manner[3][10]. Similarly, device-based backup schemes will show reduced restore time because the individual blocks of a file will not always be written contiguously to the backup storage medium[23]. Furthermore, recovering complete filesystems may cause deletion of some files because the latest backup will be shown in the recovered filesystem[3]. We can expect improved performance of single file restore in systems that store online snapshots of older versions of a filesystem[17][18], as discussed in Section 2.2.1. The following section will describe the AMANDA Network Backup to provide appropriate context for discussing the backup features detailed so far.

4.1 AMANDA Network Backup Manager

The Amanda Network Backup Manager was founded at the University of Maryland at College Park to enable network backup of multiple UNIX machines in parallel[13] Amanda takes advantage of UNIX backup programs such as tar and dump[3]. To achieve best possible performance, Amanda was designed to perform many backups in parallel to a temporary staging disk, which later writes data to a disk drive at high bandwidth[1][13].

Amandas backup cycles are determined by a central backup server host, which signals when to perform backups (most commonly in off-peak hours)[3], and also indicates the backup level to be performed for each distinct filesystem for a given cycle[25]. The Amanda backup manager provides optional file compression and full backup history for each filesystem[25].

Amanda takes error-checking measures prior to and after backups are performed[25] Prior to backups, AMANDA verifies that the correct disk drives are used and that the staging disk has enough temporary storage[3][25]. After backups, the system produces a report that details any encountered problems; these may include backup program issues such software failure or permission errors, disk errors, or reporting any client hosts that were down during the backup[3]. Although AMANDA can be considered as a reliable backup manager, there remain many limitations to existing backup schemes with regard to the increasingly demanding product requirements. These difficulties should be resolved by future work, as will be discussed in the following section.

4.2 The Future

In the past, backup algorithms and methods were invented that exploit filesystem constructs to resolve backup and restore challenges. The rsync algorithm, for example, is able to efficiently identify the contents of a source file that are equivalent to the contents of a destination file, and only transfers the data that does not match[2]. However, rsync requires that each file on the source filesystem is read to determine which contents have been modified[2][12]. Some backup software applications, such as Time Machine, resolved this efficiency problem by tracking changes made to a filesystem rather than checking each individual file for changes[4]; this can be done by utilising the inode notify subsystem of kernel[15][22]. It is no surprise that as filesystem grow in size, new backup strategies are created to protect them. Currently, the most promising strategy for handling large-scale filesystem backup is the incremental-only backup. This is a device-based scheme that does not require backing up entire files due to small file modifications, utilities copy-on-write implemented by snapshots for on-line backup, provides compression of files prior to backup, and makes use of disk bandwidth efficiently. While filesystems continue to expand exponentially, the capability of a device-based backup scheme to scale with greater disk bandwidth is intriguing, even though it is a backup and restore strategy that has been neglected in the past. With regard to WAFL, which takes advantage of snapshot's capability to efficiently pinpoint recently modified blocks, incremental image backups can also expect attention as a future backup scheme.

5 Conclusion

This paper has evaluated the issues and performance of both the file-based and device-based strategies in the context of the Write-Anywhere-File-Layout. We have demonstrated that, although challenging, it is possible for a backup system to find a balance in the performance of the backup and restore utilities. A file-based backup strategy enables recovery of single files and allows for a portable archival format. On the other hand, the device-based backup and restore strategy allows high throughput and supports multiple data replication strategies. Although both strategies have a lot to offer, the features of the device-based strategy that allow it to efficiently utilise the high bandwidths attainable when transferring data between disks means that it has the potential to be the workhorse technology used to securely backup our increasingly growing data repositories.

We classified the full and incremental backup algorithms, and shown that we can exploit the advantages of both schemes to improve backup performance for large systems by creating infrequent full backups with more frequent incremental backups. We also examined the challenges associated with enabling on-line backups, in which case a system can disable writes during a backup cycle or make read-only snapshots of the complete file system that can then be stored to the backup storage medium. Most systems that utilise snapshots also make use of the copy-on-write resources management technique, which saves significant disk space by only copying modified data.

This paper discussed many fundamental backup features, such of concurrent backups, file restores, and data compression, in the context of the popular archiving tool AMANDA Network Backup Manager. As file systems grown size and continue to require more complicated backup maintenance, we can expect to observe an array of new backup features and technologies making appearances in the industry.

6 Project Specification

My project is a backup software application distributed with the Linux operating system. The software will make incremental backups of a filesystem, which will allow restoration at a later date of either individual files or an entire filesystem. The end-user will be able to perform a restore operation from the user interface. Each backup cycle will capture the latest state of the source filesystem. As the snapshots of the filesystem age, the more recent snapshots become prioritised over the older ones on the destination volume to save storage space. The software may be used with multiple external volumes. The default settings are such that 30 minute backups of a filesystem will be stored for the past 24 hours, daily backups will be stored for the past month, and weekly backups will be kept for snapshots older than a month. However, all backups older than a day will be kept at a daily frequency until half of the space on target storage device is used. The oldest weekly backups are deleted when the destination volume runs out of space. The backend of the application runs as a daemon, and is responsible for the backup utility of the software the backend is a system service, which will read the its configuration from a configuration file to set the necessary parameters and initial settings of the software. The frontend allows the user to modify the configuration file, to an extent, using the graphical user interface limits will be placed to maintain appropriate software performance. The frontend is also responsible for providing access to the restore operation. To provide further comprehensive specifications, the sections below detail an overview of all specifications of the two of the primary application components: the backend and the frontend.

Backend

Runs as a system-level daemon (rather than user-level process), to allow processes to run even if a user is logged off the machine; it is a system service that reads its configuration from a configuration file that can be modified from the user interface.

The software works with locally connected storage devices. A folder on designated volume (disk image on a local drive*) is created, onto which the software clones the directory tree of every locally connected disk drive, with the exception of files and directories that the end-user has chosen to omit (front end option) and the backup volume itself. *A disk image can be thought of as a single file that stores the structure and contents of a disk volume.

Periodically, every X* user-defined minutes (upper and lower limits are 5 and 1,440 minutes, respectively) after the creation of the original folder, the software makes another subordinate folder and backs up only those files that have been modified since the previous backup those files that files that have not been modified are not copied again, as the software will make hard links to defer to the existing files to save space. *Upper and lower limits of X are 5 minutes and 24 hours, respectively.

A user will be able to search through the directory hierarchy of the backed up filesystem as if searching through a primary disk. An end-user should be able to manually search through the backup volume without the user interface; this is made possible by the use of hard links, which display each backup full disk copy.

The software will be available to the Linux operating system it is limited to this operating system because the use of the inotify (inode notify) subsystem of the Linux kernel is central to its functionality, which is a module that is available only to the Linux operating system of the Unix family.

The software operates on modification date of a file, which accounts for most intuitive design. This simply means that the system is concerned only with the last modified time metadata of a file, rather than executing MD5 checksums for a file, which can be inefficient for larger files e.g. large movie files. Therefore, the system operates on the assumptions that a file with new modification date but no changes to its contents is a rare occasion, as forcing such an occurrence does not pose any benefits to a user.

The system has event listeners for modification of a file on the users filesystem, which is achieved by utilising the inotify subsystem* of the Linux kernel that signals an event when a file is modified. This allows for being able to track directories that have been modified on the hard drive since the last backup. The software then needs to scan only those directories for modified files to copy in the next backup, since the unmodified files will be hard-linked, rather than having to check every file on the filesystem for its modification date every time a backup cycle occurs. *Inotify operates to extend filesystems to notice changes to the filesystem, and report those changes to applications.

Frontend

The GUI will be launched if a registered storage device is connected to the machine. In any other situation, a user can manually launch the user interface.

The front end part of the software enables browsing of snapshots if a filesystem and the retrieval of directories and files within a particular directory via an animated user interface. This feature of the front end allows for a user to be able to recover a single file*, but also to be able to entirely recover contents from the storage device to a Linux machine. This is so that a hard drive can be recovered from the backup in case that the users original disk loses all data (this can be achieved using the dd low level byte-for-byte copy utility).

When restoring a subset* or all contents from the backup device, the target disk needs to have enough storage space to accommodate this request. Otherwise, the front end the alert the user. * A user can recover a single file from the backup either by replacing the current file in the directory by the backed up file with the original modification date (in which case it is the users responsibility to be sure of this option, and chose Replace in the alert box), or to add the backed file to the directory of the current copy (Continue choice of the alert box). Alternatively, the user can Cancel the request.

A user will be able to register and de-register backup volumes in the user interface.

A user can register multiple volumes simultaneously, so an automatic back up of two (or more) separate destinations can occur in rotation. (Note: the software achieves this by rotating among the desired volumes each backup cycle in a manner that each backup is independent from the others.)

Works Cited

[1] AMANDA http://www.amanda.org/. 2017.

[2] Andrew Tridgell and Paul Mackerras *The rsync algorithm.* The Australian National University, 1996.

[3] Ann Chervenak, Vivekanand Vellanki, Zachary Kurmas. *Protecting File Systems: A Survey of Backup Techniques.* Joint NASA and IEEE Mass Storage Conference, 1998.

[4] Apple. *AirPort Time Capsule.* https://www.apple.com/uk/airport-time-capsule/.

[5] Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Andrea C. *File System Implementation.* Arpaci-Dusseau Books, 2014.

[6] Francisco Javier Thayer Fbrega, Francisco Javier, and Joshua D. Guttman *Copy on Write.* 1995.

[7] C. Mee and Eric Daniel. *Magnetic Storage Handbook, 2nd Edition.* McGraw-Hill Professional, 1996.

[8] Cunhua Qian, Syouji Nakamura, and Toshio Nakagawa *Optimal Backup Policies for a Database System with Incremental Backup, Volume 85, Issue 4.* Electronics and Communications in Japan (Part III: Fundamental Electronic Science), 2002.

[9] Dave Hitz, James Lau, and Michael Malcolm *File system design for an NFS file server appliance.* Network Appliance Corporation, 1994.

[10] David Cane and David Hirschman *Handbook of Network and System Administration.* Connected Corporation, 1998.

[11] David Salomon *Data Compression: The Complete Reference.* Springer Science & Business Media, 2013.

[12] Guan-qun Sy, Hong-cai Tao *Design and implementation of remote data backup and recovery system based on Linux.* School of Information Science and Technology, Southwest Jiaotong University, 2012.

[13] James da Silva and Olafur Guthmundsson *The Amanda Network Backup Manager.* Seventh System Administration Conference (LISA 93). Monterey, California, November, 1993.

[14] Jin-Sun Suk and Jae-Chun No *File System Snapshot, Volume 47, Issue 4, pp.88-95.* Journal of the Institute of Electronics Engineers of Korea CI, The Institute of Electronics Engineers of Korea, 2010.

[15] Michael Kerrisk *Filesystem notification, part 2: A deeper investigation of inotify.* LWN.net, 2014.

[16] Mostafa Khalil *Storage Design and Implementation in vSphere 6: A Technology Deep Dive, 2nd Edition.* vmware Press, 2017.

[17] Neeta Garimella *Understanding and exploiting snapshot technology for data protection, Part 1: Snapshot technology overview.* IBM, 2006.

[18] Norman C. Hutchinson, Stephen Manley, Mike Federwisch, Guy Harris, Dave Hitz, Steven Kleiman, and Sean O'Malley *Logical vs. Physical File System Backup.* Proceedings of the 3rd Symposium on Operating Systems Design and Implementation. University of British Columbia, Network Appliance, Inc. New Orleans, Louisiana, February, 1999.

[19] Paolo Di Francesco *Design and implementation of a MLFQ scheduler for the Bacula backup software.* Universit degli Studi dell'Aquila, Italy, 2012.

[20] Peter Macko, Margo Seltzer, and Keith A. Smith *Tracking back references in a write-anywhere file system.* USENIX Association FAST 10: 8th USENIX Conference on File and Storage Technologies, 2010.

[21] Pratap P. Nayadkar and Prof B.L Parne *Logical vs. Physical File System Backup.* (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (6) , 2014, 8236-8238. Dept. of Computer Technology, Yeshwantrao Chavan College of Engineering, Nagpur (MS), India, 2003.

[22] Robert Love *Linux Kernel Development.* Third Edition, Pearson Education, Inc., 2010.

[23] Russell J. Green, Alasdair C. Baird and J. Christopher Davies *Designing a Fast, On-line Backup System for a Log-structured File System.* Digital Technical Journal Vol. 8 No. 2, 1996.

[24] Steve Shumway *Issues in On-line Backup.* SunSoft, Inc., 1991.

[25] W. Curtis Preston *Backup & Recovery.* O'Reilly, 2007.

[26] Windows Dev Center *BY_HANDLE_FILE_INFORMATION structure.* https://msdn.microsoft.com/en-us/library/windows/desktop/aa363788(v=vs.85).aspx, Windows.

[27] Zachary Kurmas and Ann L. Chervenak *Evaluating Backup Algorithms.* College of Computing, Georgia Tech USC Information Sciences Institute, 2000.