Designing a Fast, On-line Backup System for a Log-structured File System

Russell J. Green
Alasdair C. Baird
J. Christopher Davies

The Spiralog file system for the OpenVMS operating system incorporates a new technical approach to backing up data. The fast, low-impact backup can be used to create consistent copies of the file system while applications are actively modifying data. The Spiralog backup uses the log-structured file system to solve the backup problem. The physical on-disk structure allows data to be saved at near-maximum device throughput with little processing of data. The backup system achieves this level of performance without compromising functionality such as incremental backup or fast, selective restore.

Most computer users want to be able to recover data lost through user error, software or media failure, or site disaster but are unwilling to devote system resources or downtime to make backup copies of the data. Furthermore, with the rapid growth in the use of data storage and the tendency to move systems toward complete utilization (i.e., 24-hour by 7-day operation), the practice of taking the system off line to back up data is no longer feasible.

The Spiralog file system, an optional component of the OpenVMS Alpha operating system, incorporates a new approach to the backup process (called simply backup), resulting in a number of substantial customer benefits. By exploiting the features of log-structured storage, the backup system combines the advantages of two different traditional approaches to performing backup: the flexibility of file-based backup and the high performance of physically oriented backup.

The design goal for the Spiralog backup system was to provide customers with a fast, application-consistent, on-line backup. In this paper, we explain the features of the Spiralog file system that helped achieve this goal and outline the design of the major backup functions, namely volume save, volume restore, file restore, and incremental management. We then present some performance results arrived at using Spiralog version 1.1. The paper concludes with a discussion of other design approaches and areas for future work.

Background

File system data may be lost for many reasons, including

- User error—A user may mistakenly delete data.
- Software failure—An application may execute incorrectly.
- Media failure—The computing equipment may malfunction because of poor design, old age, etc.
- Site disaster—Computing facilities may experience failures in, for example, the electrical supply or cooling systems. Also, environmental catastrophes such as electrical storms and floods may damage facilities.

The ability to save backup copies of all or part of a file system's information in a form that allows it to be restored is essential to most customers who use computing resources. To understand the backup capability needed in the Spiralog file system, we spoke to a number of customers—five directly and several hundred through public forums. Each ran a different type of system in a distinct environment, ranging from research and development to finance on OpenVMS and other systems. Our survey revealed the following set of customer requirements for the Spiralog backup system:

1. Backup copies of data must be consistent with respect to the applications that use the data.
2. Data must be continuously available to applications. Downtime for the purpose of backup is unacceptable. An application must copy all data of interest as it exists at an instant in time; however, the application should also be allowed to modify the data during the copying process. Performing backup in such a way as to satisfy these constraints is often called hot backup or on-line backup. Figure 1 illustrates how data inconsistency can occur during an on-line backup.
3. The backup operations, particularly the save operation, must be fast. That is, copying data from the system or restoring data to the system must be accomplished in the time available.
4. The backup system must allow an incremental backup operation, i.e., an operation that captures only the changes made to data since the last backup.

The Spiralog backup team set out to design and implement a backup system that would meet the four customer requirements. The following section discusses the features of the implementation of a log-structured file system (LFS) that allowed us to use a new approach to performing backup. Note that throughout this paper we use *disk* to describe the physical media used to store data and *volume* to describe the abstraction of the disk as presented by the Spiralog file system.

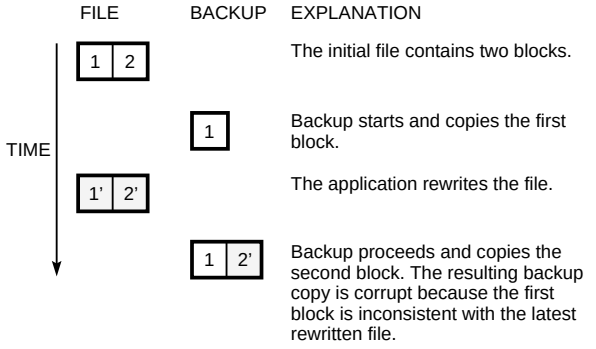


Figure 1
Example of an On-line Backup That Results in Inconsistent Data

Spiralog Features

The Spiralog file system is an implementation of a log-structured file system. An LFS is characterized by the use of disk storage as a sequential, never-ending repository of data. We generally refer to this organization of data as a log. Johnson and Laing describe in detail the design of the Spiralog implementation of an LFS and how files are maintained in this implementation.[1] Some features unique to a log-structured file system are of particular interest in the design of a backup system.[2–4] These features are

- Segments, where a segment is the fundamental unit of storage
- The no-overwrite nature of the system
- The temporal ordering of on-disk data structures
- The means by which files are constructed

This section of the paper discusses the relevance of these features; a later section explains how these features are exploited in the backup design.

Segments

In this paper, the term segment refers to a logical entity that is uniquely identified and never overwritten. This definition is distinct from the physical storage of a segment. The only physical feature of interest to backup with regard to segments is that they are efficient to read in their entirety.

Using log-structured storage in a file system allows efficient writing irrespective of the write patterns or load to the file system. All write operations are grouped in segment-sized chunks. The segment size is chosen to be sufficiently large that the time required to read or write the segment is significantly greater than the time required to access the segment, i.e., the time required for a head seek and rotational delay on a magnetic disk. All data (except the LFS homeblock and checkpoint information used to locate the end of the data log) is stored in segments, and all segments are known to the file system. From a backup point of view, this means that the entire contents of a volume can be copied by reading the segments. The segments are large enough to allow efficient reading, resulting in a near-maximum transfer rate of the device.

No Overwrite

In a log-structured file system, in which the segments are never overwritten, all data is written to new, empty segments. Each new segment is given a segment identifier (segid) allocated in a monotonically increasing

manner. At any point in time, the entire contents and state of a volume can be described in terms of a (*checkpoint position, segment list*) pair. At the physical level, a volume consists of a list of segments and a position within a segment that defines the end of the log. Rosenblum describes the concept of time travel, where an old state of the file system can be revisited by creating and maintaining a snapshot of the file system for future access.[3] Allowing time travel in this way requires maintaining an old checkpoint and disabling the reuse of disk space by the cleaner. The cleaner is a mechanism used to reclaim disk space occupied by obsolete data in a log, i.e., disk space no longer referenced in the file system. The contents of a snapshot are independent of operations undertaken on the live version of the file system. Modifying or deleting a file affects only the live version of the file system (see Figure 2). Because of the no-overwrite nature of the LFS, previously written data remains unchanged.

Other mechanisms specific to a particular backup algorithm have been developed to achieve on-line consistency.[5] The snapshot model as described above allows a more general solution with respect to multiple concurrent backups and the choice of the save algorithm.

A read-only version of the file system at an instant in time is precisely what is required for application consistency in on-line backup. This snapshot approach to attaining consistency in on-line backup has been used in other systems.[6,7] As explained in the following sections, the Spiralog file system combines the snapshot technique with features of log-structured storage to obtain both on-line backup consistency and performance benefits for backup.

Temporal Ordering

As mentioned earlier, all data, i.e., user data and file system metadata (data that describes the user data in the file system), is stored in segments and there is no overwrite of segments. All on-disk data structures that refer to physical placement of data use pointers, namely (*segid, offset*) pairs, to describe the location of the data. Each (*segid, offset*) pair specifies the segment and where within that segment the data is stored. Together, these imply the following two properties of data structures, which are key features of an LFS:

1. On-disk structure pointers, namely (*segid, offset*) pairs, are relatively time ordered. Specifically, data stored at ($s2$, $o2$) was written more recently than data stored at ($s1$, $o1$) if and only if $s2$ is greater than $s1$ or $s2$ equals $s1$ and $o2$ is greater than $o1$. Thus, new data would appear to the right in the data structure depicted in Figure 3.
2. Any data structure that uses on-disk pointers stored within the segments (the mapping data structure implementing the LFS index) must be time ordered; that is, all pointers must refer to data written prior to the pointer. Referring again to Figure 3, only data structures that point to the left are valid.

These properties of on-disk data structures are of interest when designing backup systems. Such data structures can be traversed so that segments are read in reverse time order. To understand this concept, consider the root of some on-disk data structure. This root must have been written after any of the data to which it refers (property 2). A data item that the root references must have been written before the root and so must have been stored in a segment with a segid less than or equal to that of the segment in which the root is stored (property 1). A similar inductive argument can be used to show that any on-disk data structure can be traversed using a single pass of segments in increasing segment age, i.e., decreasing segid. This is of particular interest when considering how to recover selective pieces of data (e.g., individual files) from an on-disk structure that has been stored in such a way that only sequential access is viable. The storage of the segments that compose a volume on tape as part of a backup is an example of such an on-disk data structure.

File Construction

Whitaker, Bayley, and Widdowson describe the persistent address space as exported by the Spiralog LFS.[8] Essentially, the interface presented by the log-structured server is that of a memory (various read and write operations) indexed using a file identifier and an address range. The entire contents of a file, regardless of type or size, are defined by the file identifier and all possible addresses built using that identifier.

This means of file construction is important when considering how to restore the contents of a file. All

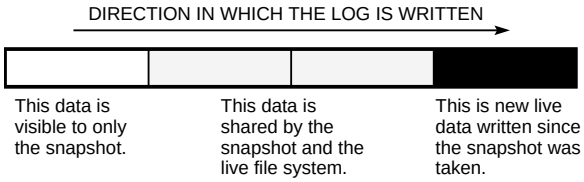


Figure 2
Data Accessible to the Snapshot and to the Live File System

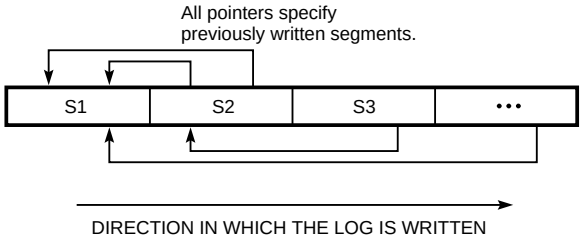


Figure 3
A Valid Data Structure in the Log

data contained in a file defined by a file identifier can be recovered, independent of how the file was created, without any knowledge of the file system structure. Consequently, together with the temporal ordering of data in an LFS, files can be recovered using an ordered linear scan of the segments of a volume, provided the on-disk data structures are traversed correctly. This mechanism allows efficient file restore from a sequence of segments. In particular, a set of files can be restored in a single pass of a saved volume stored on tape.

Existing Approaches to Backup

The design of the Spiralog backup attempts to combine the advantages of file-based backup tools such as Files-11 backup, UNIX tar, and Windows NT backup, and physical backup tools such as UNIX dd, Files-11 backup/PHYSICAL, and HSC backup (a controller-based backup for OpenVMS volumes).[9]

File-based Backup

A file-based backup system has two main advantages: (1) the system can explicitly name files to be saved, and (2) the system can restore individual files. In this paper, the file or structure that contains the output data of a backup save operation is called a saveset. Individual file restore is achieved by scanning a saveset for the file and then recreating the file using the saved contents. Incremental file-based backup usually entails keeping a record of when the last backup was made (either on a per-file basis or on a per-volume basis) and copying only those files and directories that have been created or modified since a previous backup time.

The penalty associated with these features of a file-based backup system is that of save performance. In effect, the backup system performs a considerable amount of work to lay out data in the saveset to allow simple restore. All files are segregated to a much greater extent than they are in the file system on-disk structure. The limiting factor in the performance of a file-based save operation is the rate at which data can be read from the source disk. Although there are some ways to improve performance, in the case of a volume that has a large number of files, read performance is always costly. Figure 4 illustrates the layouts of three different types of savesets.

Physical Backup

In contrast to the file-based approach to backup, a physical backup system copies the actual blocks of data on the source disk to a saveset. The backup system is able to read the disk optimally, which allows an implementation to achieve data throughput near the disk's maximum transfer rate. Physical backups typically allow neither individual file restore nor incremental backup. The overhead required to include sufficient information for these features usually erodes the performance benefits offered by the physical copy. In addition, a physical backup usually requires that the entire volume be saved regardless of how much of the volume is used to store data.

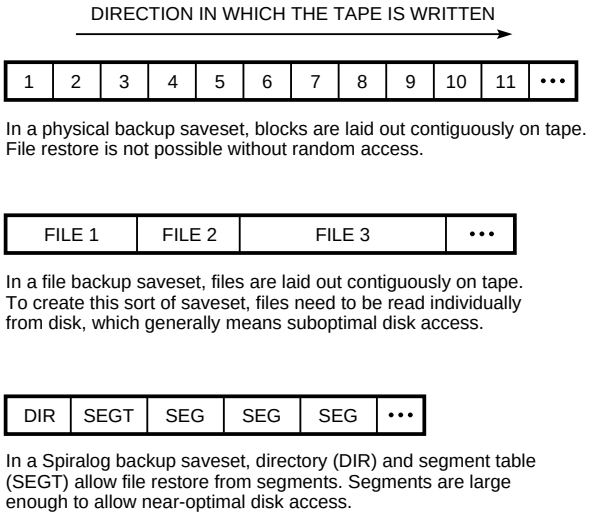


Figure 4
Layouts of Three Different Types of Saveset

How Spiralog Backup Exploits the LFS

Spiralog backup uses the snapshot mechanism to achieve on-line consistency for backup. This section describes how Spiralog attains high-performance backup with respect to the various save and restore operations.

Volume Save Operation

The save operation of Spiralog creates a snapshot and then physically copies it to a tape or disk structure called a savesnap. (This term is chosen to be different from saveset to emphasize that it holds a consistent snapshot of the data.) This physical copy operation allows high-performance data transfer with minimal processing.[10] In addition, the temporal ordering of data stored by Spiralog means that this physical copy operation can also be an incremental operation.

The savesnap is a file that contains, among other information, a list of segments exactly as they exist in the log. The structure of the savesnap allows the efficient implementation of volume restore and file restore (see Figure 5 and Figure 6).

The steps of a full save operation are as follows:

1. Create a snapshot and mount it. This mounted snapshot looks like a separate, read-only file system. Read information about the snapshot.

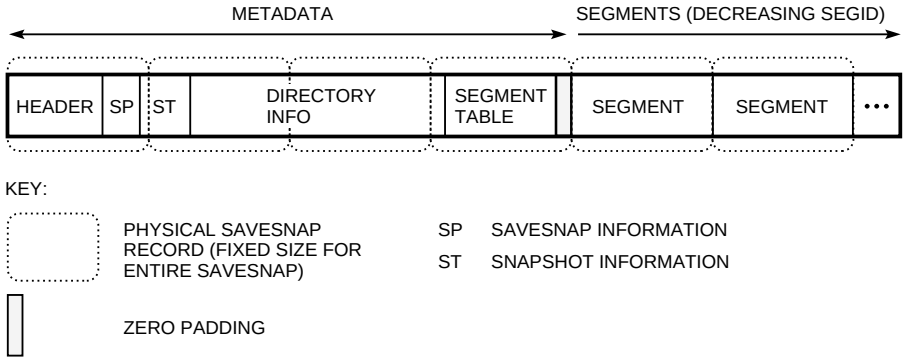


Figure 5
Savesnap Structure

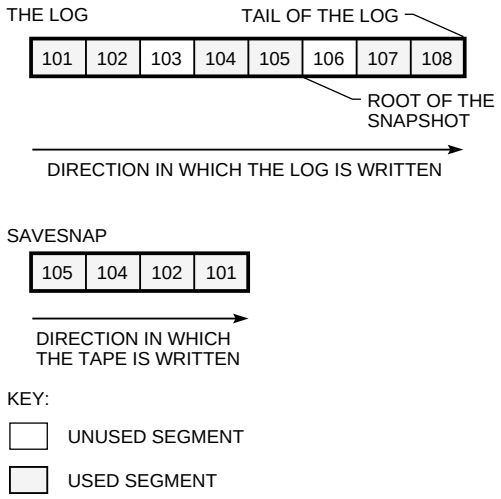


Figure 6
Correspondence between Segments on Disk and in the Savesnap

2. Write the header to the savesnap, including snapshot information such as the checkpoint position.
3. Copy the contents of the file system directories to the savesnap.
4. Write the list of segids that compose the snapshot to the savesnap as a segment table in decreasing segid order.
5. Copy these segments in decreasing segid order from the volume to the savesnap (see Figure 6).
6. Dismount and delete the snapshot, leaving only the contents of the live volume accessible. The effect of deleting the snapshot is to release all the space used to store segments that contain only snapshot data. All segments that contain data in the live volume are left intact.

The Spiralog backup system is primarily physical. The system copies the volume (snapshot) data in segments that are large enough to allow efficient disk reading, regardless of the number of files in the volume. To save a volume, the Spiralog backup system has to read all the directories in the volume and then all the segments. In comparison, a file-based backup system must read all the directories and then all the files. On volumes with large file populations, file-based backup performance suffers greatly as a result of the number of read operations required to save the volume. Our measurements showed that the directory-gathering phase of our copy operation was insignificant in relation to the data transfer during the segment copy phase.

Incremental Save Operation

The incremental save operation in Spiralog is very different from that in a file-based backup. We use the temporal ordering feature of the LFS to capture only the changes in a volume's data as part of the incremental save. The temporal ordering provides a simple way of determining the relative age of data. To be precise, data in the segment with segid $s2$ must have been written after data in the segment with segid $s1$ if and only if $s2$ is greater than $s1$.

Consider the lifetime of a volume as an endless sequence of segments. A backup copy of a volume at any time is a copy of all segments that contain data accessible in that volume. Segments in the volume's history that are not included in the backup copy are those that no longer contain any useful data or those that have been cleaned. An incremental backup contains the sequence of segments containing accessible data written since a previous backup.

This is different from an incremental save operation in a file-based backup scheme. The Spiralog incremental save operation copies only the data written since the last backup. In comparison, a file-based backup

incremental save comprises entire files that contain new or modified data. For example, consider an incremental save of a volume in which a large database file has had only one record updated in place since a full backup. Spiralog's incremental save copies the segments written since the last full backup that contain the modified record with other updated file system index data. A file-based backup copies the entire database file.

The following steps for the incremental save operation augment the six process steps previously described for the save operation. Note that steps 3a, 4a, and 5a follow steps 3, 4, and 5, respectively.

3a. Write dependent savesnap information. This is a list of the savesnaps required to complete the chain of segments that constitutes the entire snapshot contents. The savesnap information includes a unique savesnap identifier (*volume id, segment id, segment offset*). This is the checkpoint position of the snapshot and is unique across volumes.

4a. Determine the segment range to be stored in this savesnap. This range is calculated by reading the segment range of the last backup from a file stored on the source volume.

5a. Record the minimum segid stored in this savesnap with the segment table. The segment table contains the segids of all segments in the saved snapshot. The incremental savesnap contains segments identified by a subset of these segids. The segid of the last segment stored in the savesnap is recorded as the minimum segid held in the savesnap.

7. Record on the source volume the segment range stored in the savesnap.

The implementation provides an interface that allows the user to specify the maximum number of savesnaps required for a restore operation. This feature is similar to specifying the levels in the UNIX dump utility, where a level 0 save is a full backup (it requires no other savesnaps for a restore), and a level 1 save is an incremental backup since the full backup (it requires one additional savesnap for a restore, namely the full backup).

Figure 7 shows the savesnaps produced from full and incremental save operations. Note that the most recently written segment may appear in two different savesnaps that supposedly contain disjoint data. For example, segment 4, the youngest segment in Monday's savesnap, appears in the savesnaps made on both Monday and Wednesday. The youngest segment is not guaranteed to be full at the time of a snapshot creation, and therefore a later savesnap may contain data that was not in the first savesnap. Consequently, incremental savesnaps recapture the oldest segment in their segment range.

Note that with this design a slowly changing file can be spread across many incremental savesnaps. Restoring such a file accordingly may require access to many savesnaps. The file restore section shows that the design of file restore allows efficient tape traversal for these files.

Volume Restore Operation

The Spiralog backup volume restore operation takes a set of savesnaps and copies the segments that make up a snapshot onto a disk. Together, this set of segments and the location of the snapshot checkpoint define a volume. The steps involved in a volume restore from a full savesnap are

1. Open the savesnap, and read the snapshot checkpoint position from the savesnap header.
2. Initialize the target disk to be a Spiralog volume.
3. Copy all segments from the savesnap to the target disk. Note that the segments written to the target disk do not depend in any way on the target disk geometry. This means that the target disk may be completely different from the source

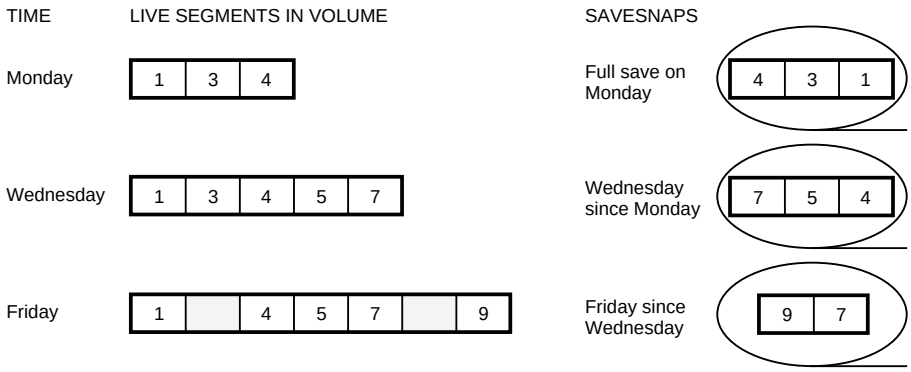


Figure 7
Snapshot Contents in Incremental Savesnaps

disk from which the savesnap was made, providing the target container is large enough to hold the restored segments.

4. Backup declares the volume restore as complete (no more segments will be written to the volume). Backup tells the file system how to mount the volume by supplying the snapshot checkpoint location.

A Spiralog restore operation treats an incremental savesnap and all the preceding savesnaps upon which it depends as a single savesnap. For savesnaps other than the most recent savesnap (the base savesnap), the snapshot information and directory information are ignored. The sole purpose of these savesnaps is to provide segments to the base savesnap.

To restore a volume from a set of incremental savesnaps, the Spiralog backup system performs steps 1 and 2 using the base savesnap. In step 3, the restore copies all the segments in the snapshot defined by the base savesnap to the target disk. (Note that there is a one-to-one correspondence between snapshots and savesnaps.) The savesnaps are processed in reverse chronological order. The contents of the segment table in the base savesnap define the list of segments in the snapshot to be restored. Although the volume restore operation copies all the segments in the base savesnap, not all segments in the savesnaps processed may be required. Savesnaps are included in the restore process if they contain some segments that are needed. Such savesnaps may also contain segments that were cleaned before the base savesnap was created.

The structure of the savesnap allows the efficient location and copying of specific segments. The segment table in the savesnap describes exactly which segments are stored in the savesnap. Since the segments are of a fixed size, it is easy to calculate the position within the savesnap where a particular segment is stored, provided the segment table is available and the position of the first segment is known. This will always be the case by the time the segment table has been read because the segments immediately follow this table.

Most savesnaps are stored on tape. This storage medium lends itself to the indexing just described. In particular, modern tape drives such as the Digital Linear Tape (DLT) series provide fast, relative tape positioning that allows tape-based savesnaps to be selectively read more quickly than with a sequential scan.[11] Similarly, on random-access media such as disks, a particular segment can be read without strict sequential scanning of data.

The volume restore operation is therefore a physical operation. The segments can be read and written efficiently (even in the case of incremental savesnaps from sequential media), resulting in a high-performance recovery from volume failure or site disaster.

File Restore Operation

The purpose of a file restore operation is to provide a fast and efficient way to retrieve a small number of files from a savesnap without performing a full volume restore. Typically, file restore is used to recover files that have been inadvertently deleted. To achieve high-performance file restore, we imposed the following requirements on the design:

- A file restore session must process as few savesnaps as possible; it should skip savesnaps that do not contain data needed by the session.
- When processing a savesnap, the file restore must scan the savesnap linearly, in a single pass.

The process of restoring files can be broken down into three steps: (1) discover the file identifiers for all the files to be restored; (2) use the file identifiers to locate the file data in the saved segments, and then read that data; and (3) place the newly recovered data back into the current Spiralog file system.

Discovering the File Identifiers The user supplies the names of the files to be restored. The mapping between the file names and the file identifiers associated with these names is stored in the segments, but this information cannot be discovered simply by inspecting the contents of the saved segments. A corollary of the temporal ordering of the segments within a savesnap is that hierarchical information, such as nested directories, tends to be presented in precisely the wrong order for scanning in a single pass. To overcome this problem, the save operation writes the complete directory tree to the savesnap before copying any segments to the savesnap. This tree maps file names to identifiers for every file and directory in the savesnap. The file restore session constructs a partial tree of the names of the files to be restored. The partial tree is then matched, in a single pass, against the complete tree stored in the savesnap. This process produces the required file identifiers.

Locating and Reading the File Data After discovering the file identifiers, the file restore session reads the list of segments present in the savesnap; this list comes after the directory tree and before any saved segments. The file restore then switches focus to discover precisely which segments contain the file data that correspond to the file identifiers.

The first segment read from the savesnap contains the tail of the log. The log provides a mapping between file identifiers and locations of data within segments. The tail of the log contains the root of the map.

We developed a simple interface for the file restore to use to navigate the map. Essentially, this interface permits the retrieval of all mapping information

relevant to a particular file identifier that is held within a given segment. The mapping information returned through this interface describes either mapping information held elsewhere or real file data. One characteristic of the log is that anything to which such mapping information points must occur earlier in the log, that is, in a subsequent saved segment. Recall property 2 of the LFS on-disk data structures. Consequently, the file restore session will progress through the savesnaps in the desired linear fashion provided that requests are presented to the interface in the correct order. The correct order is determined by the allocation of segids. Since segids increase monotonically over time, it is necessary only to ensure that requests are presented in a decreasing segid order.

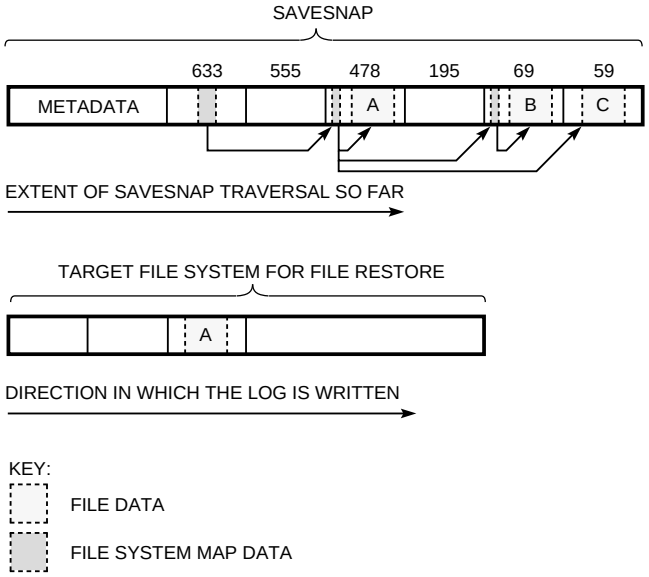
The file restore interface operates on an object called a context. The context is a tuple that contains a location in the log, namely (*segid, offset*), and a type field. When supplied with a file identifier and a context, the core function of the interface inspects the segment determined by the context and returns the set of contexts that enumerate all available mapping information for the file identifier held at the location given by the initial context.

The type of context returned indicates one of the following situations:

- The location contains real file data.
- The location given by the context holds more mapping information. In this case, the core function can be applied repeatedly to determine the precise location of the file's data.

A work list of contexts in decreasing segid order drives the file restore process. The procedure for retrieving the data for a single file identifier is as follows. At the outset of the file restore operation, the work list holds a single context that identifies the root of the map (the tail of the log). As items are taken from the head of the list, the file restore must perform one of two actions. If the context is a pointer to real file data, then the file restore reads the data at that location. If the context holds the location of mapping information, then the core function must be applied to enumerate all possible further mapping information held there. The file restore operation places all returned contexts in the work list in the correct order prior to picking the next work item. This simple procedure, which is illustrated in Figure 8, continues until the work list is empty and all the file's data has been read.

To cope with more than one file, the file restore operation extends this procedure by converting the work list so that it associates a particular file identifier



The shaded areas represent the file data to be restored and the file system metadata that needs to be accessed to retrieve that data. The restore session has thus far processed segment 478. Part A of the file has been recovered into the target file system. Parts B and C are still to come. After processing segment 478, the file restore visits the next known parts of the log, segments 69 and 59. Items that describe metadata in segment 69 and data in segment 59 will be on the work list. The next segment that the file restore will read is segment 69, so the session can skip the intervening segment (segment 195).

Figure 8
File Restore Session in Progress

with each context. File restore initializes the work list to hold a pointer to the root of the map (the tail of the log) for each file identifier to be restored. The effect is to interleave requests to read more than one file while maintaining the correct segid ordering.

A further subtlety occurs when the context at the head of the work list is found to refer to a segment outside the current savesnap. The ordering imposed on the work list implies that all subsequent items of work must also be outside the current savesnap. This follows from the temporal ordering properties of LFS on-disk structures and the way in which incremental savesnaps are defined. When this situation occurs, the work list is saved. When the next savesnap is ready for processing, the file restore session can be restarted using the saved work list as the starting point.

During this step, the file restore writes the pieces of files to the target volume as they are read from the savesnap. Since the file restore process allocates file identifiers on a per-volume basis, restore must allocate new file identifiers in the target volume to accept the data being read from the source savesnap.

The new file identifiers are hidden from users during the file restore until the file restore process has finished since the files are not complete and may be missing vital parts such as access permissions. Rather than allow access to these partial files, the file restore hides the new file identifiers until all the data is present, at which time the final stage of the file restore can take place.

Making the Recovered Files Available to the User In the third step of the process, the file restore operation makes the newly recovered files accessible. At the beginning of the step, the files exist only as bits of data associated with new file identifiers—the files do not yet have names. The names that are now bound to these file identifiers come from the partial directory tree that was originally used to match against the directory tree in the savesnap. This final step restores the original names and contents to all the files that were originally requested. The files retain the new file identifiers that were allocated during the file restore process.

Management of Incremental Saves

One design goal for the Spiralog backup was to reduce the cost of storage management. The design includes the means of performing an incremental volume save that copies only data written since the previous backup. To implement a backup strategy that never requires more than one full backup but allows restores using a finite number of savesnaps, we designed and implemented the savesnap merge function.

Savesnap merge operates similarly to volume restore, but instead of copying segments to a disk as in a volume restore, savesnap merge copies segments to a new savesnap. As shown in Figure 9, the effect of merging a base savesnap and all the incremental savesnaps upon which it depends is to produce a full savesnap. This savesnap is precisely the one that would have been created had the base savesnap been specified as a full savesnap instead of an incremental savesnap. Spiralog merge copies the savesnap information and the directory information stored in the base savesnap to the merged savesnap before it copies the segment table and the segments.

Savesnap merge provides a practical way of managing very large data volumes. The merge operation can be used to limit the number of savesnaps required to restore a snapshot, even if full backups are never taken. Merge is independent of the source volume and can be undertaken on a different system to allow further system management flexibility.

Summary of Spiralog Backup Features

A summary of the features and performance provided by the Spiralog backup system appears in Table 3 at the end of the Results section. For comparison, the table also contains corresponding information for the file-based and physical approaches to backup.

Results

We measured volume save and individual file restore performance on both the Spiralog backup system and the backup system for Files-11, the original OpenVMS file system. The hardware configuration consisted of a DEC 3000 Model 500 and a single RZ25 source disk each for Spiralog and Files-11 volumes, respectively. The target device for the backup was a TZ877 tape. The system was running under the OpenVMS version 7.0 operating system and Spiralog version 1.1. The volumes were populated with file distributions that reflected typical user accounts in our development environment. Each volume contained 260 megabytes (MB) of user data, which included a total of 21,682 files in 401 directories.

Volume Save Performance

For both the Spiralog backup and the Files-11 backup, we saved the source volume to a freshly initialized tape on an otherwise idle system. We measured the elapsed time of the save operation and recorded the size of the output savesnap or saveset. We averaged the results over five iterations of the benchmark. Table 1 presents these measurements and the resulting throughput.

The throughput represents the average rate in megabytes per second (MB/s) of writing to tape over the duration of a save operation. In the case of Spiralog, tape throughput varies greatly with the

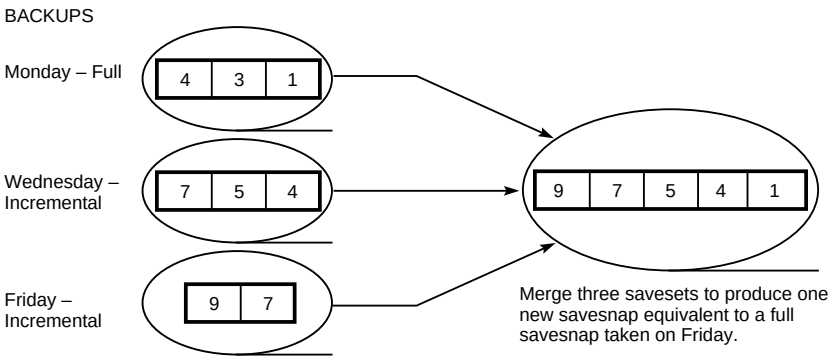


Figure 9
Merging Savesnaps

Table 1
Performance Comparison of the Spiralog and Files-11 Backup Save Operations

Backup System	Elapsed Time (Minutes:seconds)	Savesnap or Saveset Size (Megabytes)	Throughput (Megabytes/second)
Spiralog save	05:20	339	1.05
Files-11 backup	10:14	297	0.48

phases of the save operation. During the directory scan phase (typically up to 20 percent of the total elapsed save time), the only tape output is a compact representation of the volume directory graph. In comparison, the segment writing phase is usually bound by the tape throughput rate. In this configuration, the tape is the throughput bottleneck; its maximum raw data throughput is 1.25 MB/s (uncompressed).[11]

Overall, the Spiralog volume save operation is nearly twice as fast as the Files-11 backup volume save operation in this type of computing environment. Note that the Spiralog savesnap is larger than the corresponding Files-11 saveset. The Spiralog savesnap is less efficient at holding user data than the packed per-file representation of the Files-11 saveset. In many cases, though, the higher performance of the Spiralog save operation more than outweighs this inefficiency, particularly when it is taken into account that the Spiralog save operation can be performed on-line.

File Restore Performance

To determine file restore performance, we measured how long it took to restore a single file from the savesets created in the save benchmark tests. The hardware and software configurations were identical to those used for the save measurements. We deleted a single 3-kilobyte (KB) file from the source volume and then restored the file. We repeated this operation nine times, each time measuring the time it took to restore the file. Table 2 shows the results.

Table 2
Performance Comparison of the Spiralog and Files-11 Individual File Restore Operations

Backup System	Elapsed Time (Minutes:seconds)
Spiralog file restore	01:06
Files-11 backup	03:35

The Spiralog backup system achieves such good performance for file restore by using its knowledge of the way the segments are laid out on tape. The file restore process needs to read only those segments required to restore the file; the restore skips the intervening segments using tape skip commands. In the example presented in Figure 8, the restore can skip segments 555 and 195. In contrast, a file-based backup such as Files-11 usually does not have accurate indexing information to minimize tape I/O. Spiralog's tape-skipping benefit is particularly noticeable when restoring small numbers of files from very large savesnaps; however, as shown in Table 2, even with small savesets, individual file restore using Spiralog backup is three times as fast as using Files-11.

Table 3 presents a comparison of the save performance and features of the Spiralog, file-based, and physical backup systems.

Table 3
Comparison of Spiralog, File-based, and Physical Backup Systems

	Spiralog Backup System	File-based Backup System	Physical Backup System
Save performance (the number of I/Os required to save the the source volume)	The number of I/Os is O(number of segments that contain live data) plus O(number of directories)	The number of I/Os is O(number of files) I/Os to read the file data plus O(number of directories) I /Os	The number of I/Os is O(size of the disk)
File restore	Yes	Yes	No
Volume restore	Yes, fast	Yes	Yes, fast but limited to disks of the same size
Incremental save	Yes, physical	Yes, entire files that have changed	No

Note that this table uses "big oh" notation to bound a value. O(*n*), which is pronounced "order of *n*," means that the value represented is no greater than C*n* for some constant C, regardless of the value of *n*. Informally, this means that O(*n*) can be thought of as some constant multiple of *n*.

Other Approaches and Future Work

This section outlines some other design options we considered for the Spiralog backup system. Our approach offers further possibilities in a number of areas. We describe some of the opportunities available.

Backup and the Cleaner

The benefits of the write performance gains in an LFS are attained at the cost of having to clean segments.[8] An opportunity appears to exist in combining the cleaner and backup functions to reduce the amount of work done by either or both of these components; however, the aims of backup and the cleaner are quite different. Backup needs to read all segments written since a specific time (in the case of a full backup, since the birth of the volume). The cleaner needs to defragment the free space on the volume. This is done most efficiently by relocating data held in certain segments. These segments are those that are sufficiently empty to be worth scavenging for free space. The data in these segments should also be stable in the sense that the data is unlikely to be deleted or outdated immediately after relocation.

The only real benefit that can be exacted by looking at these functions together is to clean some segments while performing backup. For example, once a segment has been read to copy to a savesnap, it can be cleaned. This approach is probably not a good one because it reduces system performance in the following ways: additional processing required in cleaning removes CPU and memory resources available to applications, and the cleaner generates write operations that reduce the backup read rate.

There are two other areas in which backup and the cleaner mechanism interact that warrant further investigation.

1. The save operation copies segments in their entirety. That is, the operation copies both "stale" (old) data and live data to a savesnap. The cost of extra storage media for this extraneous data is traded off against the performance penalty in trying to copy only live data. It appears that the file system should run the cleaner vigorously prior to a backup to minimize the stale data copied.
2. Incremental savesnaps contain cleaned data. This means that an incremental savesnap contains a copy of data that already exists in one of the savesnaps on which it depends. This is an apparent waste of effort and storage space.

It is best to undertake a full backup after a thorough cleaning of the volume. A single strategy for incremental backups is less easy to define. On one hand, the size of an incremental backup is increased if much cleaning is performed before the backup. On the other hand, restore operations from a large incremental backup (particularly selective file restores) are likely to be more efficient. The larger the incremental backup, the more data it contains. Consequently, the chance of restoring a single file from just the base savesnap increases with the size of the incremental backup. Studying the interactions between the backup and the cleaner may offer some insight into how to improve either or both of these components.

A continuous backup system can take copies of segments from disk using policies similar to the cleaner. This is explored in Kohl's paper. [12]

Separating the Backup Save Operation into a Snapshot and a Copy

The design of the save operation involves the creation of a snapshot followed by the fast copy of the snapshot to some separate storage. The Spiralog version 1.1 implementation of the save operation combines these steps. A snapshot can exist only during a backup save operation.

System administrators and applications have significantly more flexibility if the split in these two functions of backup is visible. The ability to create snapshots that can be mounted to look like read-only versions of a file system may eliminate the need for the large number of backups performed today. Indeed, some file systems offer this feature.[6,7] The additional advantage that Spiralog offers is to allow the very efficient copying of individual snapshots to off-line media.

Improving the Consistency and Availability of On-line Backup

There are a number of ways to improve application consistency and availability using the Spiralog backup design. In addition, some of these features further reduce storage management costs.

Intervolume Snapshot Creation Spiralog allows a practical way of creating and managing large volumes, but there will be times when applications require data consistency for backup across volumes. A coordinated snapshot across volumes would provide this.

Application Involvement The Spiralog version 1.1 implementation does not address application involvement in the creation of a snapshot. A snapshot's contents are precisely the volume's contents that are on disk at the time of snapshot creation. This means that applications accessing the volume have to commit independently to the file system data they require to be part of the snapshot.

There is an emerging trend to design system-level interfaces that allow better application interaction with the file system. For example, the Windows NT operating system provides the oplock and NtNotifyChangeDirectory interfaces to advise an interested application of changes to files and directories. Similarly, an interface could allow applications to register an interest with the file system for notification of an impending snapshot creation. The application would then be able to commit the data it needs as part of a backup and continue, thus improving application consistency and availability and reducing work for system administrators.

Minimizing Disk Reads

The Spiralog file restore retrieves the data that constitutes a number of files in a single pass of segments read in a specific order. This design was important to allow the efficient restore of files from sequential media.

More generally, this way of traversing the file system allows specific, known parts of a set of files to be obtained by reading the segments that contain part of this data only once. This technique is also interesting for random-access media storage of volumes because it describes an algorithm for minimizing the number of disk reads to get this data. Possible applications of this technique are numerous and are particularly interesting in the context of data management of very large volumes.

For example, suppose an application is required to monitor an attribute (e.g., the time of last access) of all files on a massive volume. Suppose also that the volume is too big to allow the application to trawl the file system daily for this information; this process takes too long. If the application maintains a database of the information, it needs only to gather the changes that have happened to this data on a daily basis. Therefore, the application could obtain this information by traversing only those segments written since the last time it updated its database and locating the relevant data within those segments. Our mechanism for restoring files provides exactly this capability. An investigation of how applications might best use this technique could lead to the design of an interface that the file system could use for fast scanning of data.

Conclusions

File systems use backup to protect against data loss. A significant portion of the cost associated with managing storage is directly related to the backup function.[13–17] Log-structured data storage provides some features that reduce the costs associated with backup.

The Spiralog log-structured file system version 1.1 for the OpenVMS Alpha operating system includes a new, high-performance, on-line backup system. The approach that Spiralog takes to obtain data consistency for on-line backup is similar to the snapshot approach used in Network Appliance Corporation's FAServer, the Digital UNIX Advanced File System, and other systems.[6,7] The feature unique to the Spiralog backup system is its use of the physical attributes of log-structured storage to obtain high-performance saving and restoring of data to and from tape. In particular, the gain in save performance is the result of a restore strategy that can efficiently retrieve data from a sequence of segments stored on tape as they are on disk. This design leads to a minimum of processing and discrete I/O operations. The restore operation uses improvements in tape hardware to reduce processing and I/O bandwidth consumption; the operation uses tape record skipping within savesnaps for fast

data indexing. The Spiralog backup implementation provides an on-line backup save operation with significantly improved performance over existing offerings. Performance of individual file restore is also improved.

Acknowledgments

We would like to thank the following people whose efforts were vital in bringing the Spiralog backup system to fruition: Nancy Phan, who helped us develop the product and worked relentlessly to get it right; Judy Parsons, who helped us clarify, describe, and document our work; Clare Wells, who helped us focus on the real customer problems; Alan Paxton, who was involved in the early design ideas and later specification of some of the implementation; and, finally, Cathy Foley, our engineering manager, who supported us throughout the project.

References

1. J. Johnson and W. Laing, "Overview of the Spiralog File System," *Digital Technical Journal,* vol. 8, no. 2 (1996, this issue): 5–14.
2. M. Rosenblum and J. Ousterhout, "The Design and Implementation of a Log-Structured File System," *ACM Transactions on Computer Systems,* vol. 10, no. 1 (February 1992): 26–52.
3. M. Rosenblum, "The Design and Implementation of a Log-Structured File System," Report No. UCB/CSD 92/696 (Berkeley, Calif.: University of California, Berkeley, 1992).
4. M. Seltzer, K. Bostock, M. McKusick, and C. Staelin, "An Implementation of a Log-Structured File System for UNIX," *Proceedings of the USENIX Winter 1993 Technical Conference,* San Diego, Calif. (January 1993).
5. K. Walls, "File Backup System for Producing a Backup Copy of a File Which May Be Updated during Backup," U.S. Patent No. 5,163,148.
6. D. Hitz, J. Lau, and M. Malcolm, "File System Design for an NFS File Server Appliance," *Proceedings of the USENIX Winter 1994 Technical Conference,* San Francisco, Calif. (January 1994).
7. S. Chutani, O. Anderson, M. Kazar, and B. Leverett, "The Episode File System," *Proceedings of the USENIX Winter 1992 Technical Conference,* San Francisco, Calif. (January 1992).
8. C. Whitaker, J. Bayley, and R. Widdowson, "Design of the Server for the Spiralog File System," *Digital Technical Journal,* vol. 8, no. 2 (1996, this issue): 15–31.
9. *OpenVMS System Management Utilities Reference Manual: A–L,* Order No. AA-PV5PC-TK (Maynard, Mass.: Digital Equipment Corporation, 1995).
10. L. Drizis, "A Method for Fast Tape Backups and Restores," *Software—Practice and Experience,* vol. 23, no. 7 (July 1993): 813–815.
11. "Digital Linear Tape Meets Critical Need for Data Backup," Quantum Technical Information Paper, http://www.quantum.com/products/whitepapers/dlttips.html (Milpitas, Calif.: Quantum Corporation, 1996).
12. J. Kohl, C. Staelin, and M. Stonebraker, "HighLight: Using a Log-structured File System for Tertiary Storage Management," *Proceedings of the USENIX Winter 1993 Technical Conference* (Winter 1993).
13. R. Mason, "The Storage Management Market Part 1: Preliminary 1994 Market Sizing," IDC No. 9538 (Framingham, Mass.: International Data Corporation, December 1994).
14. I. Stenmark, "Implementation Guidelines for Client/Server Backup" (Stamford, Conn.: Gartner Group, March 14, 1994).
15. I. Stenmark, "Market Size: Network and Systems Management Software" (Stamford, Conn.: Gartner Group, June 30, 1995).
16. I. Stenmark, "Client/Server Backup—Leaders and Challengers" (Stamford, Conn.: Gartner Group, May 9, 1994).
17. R. Wrenn, "Why the Real Cost of Storage is More Than $1/MB," presented at the U.S. DECUS Symposium, St. Louis, Mo., June 3–6, 1996.

Biographies

Russell J. Green
Russell Green is a principal software engineer in Digital's OpenVMS Engineering group in Livingston, Scotland. He was responsible for the design and delivery of the backup component of the Spiralog file system for the OpenVMS operating system. Currently, Russ is the technical leader of Spiralog follow-on work. Prior to joining Digital in 1991, he was a staff member in the computer science department at the University of Edinburgh. Russ received a B.Sc. (Honours, 1st class, 1983) in engineering from the University of Cape Town and an M.Sc. (1986) in engineering from the University of Edinburgh. He holds two patents and has filed a patent application for his Spiralog backup system work.

Alasdair C. Baird
Alasdair Baird joined Digital in 1988 to work for the ULTRIX Engineering group in Reading, U.K. He is a senior software engineer and has been a member of Digital's OpenVMS Engineering group since 1991. He worked on the design of the Spiralog file system and then contributed to the Spiralog backup system, particularly the file restore component. Currently, he is involved in Spiralog development work. Alasdair received a B.Sc. (Honours, 1988) in computer science from the University of Edinburgh.

J. Christopher Davies
Software engineer Chris Davies has worked for Digital Equipment Corporation in Livingston, Scotland, since September 1991. As a member of the Spiralog team, he initially designed and implemented the Spiralog on-line backup system. In subsequent work, he improved the performance of the file system. Chris is currently working on further Spiralog development. Prior to joining Digital, Chris was employed by NRG Surveys as a software engineer while earning his degree. He holds a B.Sc. (Honours, 1991) in artificial intelligence and computer science from the University of Edinburgh. He is coauthor of a filed patent application for the Spiralog backup system.