Overview of the Spiralog File System

James E. Johnson
William A. Laing

The OpenVMS Alpha environment requires a file system that supports its full 64-bit capabilities. The Spiralog file system was developed to increase the capabilities of Digital's Files-11 file system for OpenVMS. It incorporates ideas from a log-structured file system and an ordered write-back model. The Spiralog file system provides improvements in data availability, scaling of the amount of storage easily managed, support for very large volume sizes, support for applications that are either write-operation or file-system-operation intensive, and support for heterogeneous file system client types. The Spiralog technology, which matches or exceeds the reliability and device independence of the Files-11 system, was then integrated into the OpenVMS operating system.

Digital's Spiralog product is a log-structured, clusterwide file system with integrated, on-line backup and restore capability and support for multiple file system personalities. It incorporates a number of recent ideas from the research community, including the log-structured file system (LFS) from the Sprite file system and the ordered write back from the Echo file system.[1,2]

The Spiralog file system is fully integrated into the OpenVMS operating system, providing compatibility with the current OpenVMS file system, Files-11. It supports a coherent, clusterwide write-behind cache and provides high-performance, on-line backup and per-file and per-volume restore functions.

In this paper, we first discuss the evolution of file systems and the requirements for many of the basic designs in the Spiralog file system. Next we describe the overall architecture of the Spiralog file system, identifying its major components and outlining their designs. Then we discuss the project's results: what worked well and what did not work so well. Finally, we present some conclusions and ideas for future work.

Some of the major components, i.e., the backup and restore facility, the LFS server, and OpenVMS integration, are described in greater detail in companion papers in this issue.[3–5]

The Evolution of File Systems

File systems have existed throughout much of the history of computing. The need for libraries or services that help to manage the collection of data on long-term storage devices was recognized many years ago. The early support libraries have evolved into the file systems of today. During their evolution, they have responded to the industry's improved hardware capabilities and to users' increased expectations. Hardware has continued to decrease in price and improve in its price/performance ratio. Consequently, ever larger amounts of data are stored and manipulated by users in ever more sophisticated ways. As more and more data are stored on-line, the need to access that data 24 hours a day, 365 days a year has also escalated.

Significant improvements to file systems have been made in the following areas:

- Directory structures to ease locating data
- Device independence of data access through the file system
- Accessibility of the data to users on other systems
- Availability of the data, despite either planned or unplanned service outages
- Reliability of the stored data and the performance of the data access

Requirements of the OpenVMS File System

Since 1977, the OpenVMS operating system has offered a stable, robust file system known as Files-11. This file system is considered to be very successful in the areas of reliability and device independence. Recent customer feedback, however, indicated that the areas of data availability, scaling of the amount of storage easily managed, support for very large volume sizes, and support for heterogeneous file system client types were in need of improvement.

The Spiralog project was initiated in response to customers' needs. We designed the Spiralog file system to match or somewhat exceed the Files-11 system in its reliability and device independence. The focus of the Spiralog project was on those areas that were due for improvement, notably:

- Data availability, especially during planned operations, such as backup.

  If the storage device needs to be taken off-line to perform a backup, even at a very high backup rate of 20 megabytes per second (MB/s), almost 14 hours are needed to back up 1 terabyte. This length of service outage is clearly unacceptable. More typical backup rates of 1 to 2 MB/s can take several days, which, of course, is not acceptable.
- Greatly increased scaling in total amount of on-line storage, without greatly increasing the cost to manage that storage.

  For example, 1 terabyte of disk storage currently costs approximately $250,000, which is well within the budget of many large computing centers. However, the cost in staff and time to manage such amounts of storage can be many times that of the storage.[6] The cost of storage continues to fall, while the cost of managing it continues to rise.
- Effective scaling as more processing and storage resources become available.

  For example, OpenVMS Cluster systems allow processing power and storage capacity to be added incrementally. It is crucial that the software support-ing the file system scale as the processing power, bandwidth to storage, and storage capacity increase.
- Improved performance for applications that are either write-operation or file-system-operation intensive.

  As file system caches in main memory have increased in capacity, data reads and file system read operations have become satisfied more and more from the cache. At the same time, many applications write large amounts of data or create and manipulate large numbers of files. The use of redundant arrays of inexpensive disks (RAID) storage has increased the available bandwidth for data writes and file system writes. Most file system operations, on the other hand, are small writes and are spread across the disk at random, often negating the benefits of RAID storage.
- Improved ability to transparently access the stored data across several dissimilar client types.

  Computing environments have become increasingly heterogeneous. Different client systems, such as the Windows or the UNIX operating system, store their files on and share their files with server systems such as the OpenVMS server. It has become necessary to support the syntax and semantics of several different file system personalities on a common file server.

These needs were central to many design decisions we made for the Spiralog file system.

The members of the Spiralog project evaluated much of the ongoing work in file systems, databases, and storage architectures. RAID storage makes high bandwidth available to disk storage, but it requires large writes to be effective. Databases have exploited logs and the grouping of writes together to minimize the number of disk I/Os and disk seeks required. Databases and transaction systems have also exploited the technique of copying the tail of the log to effect backups or data replication. The Sprite project at Berkeley had brought together a log-structured file system and RAID storage to good effect.[1]

By drawing from the above ideas, particularly the insight of how a log structure could support on-line, high-performance backup, we began our development effort. We designed and built a distributed file system that made extensive use of the processor and memory near the application and used log-structured storage in the server.

Spiralog File System Design

The main execution stack of the Spiralog file system consists of three distinct layers. Figure 1 shows the overall structure. At the top, nearest the user, is the file

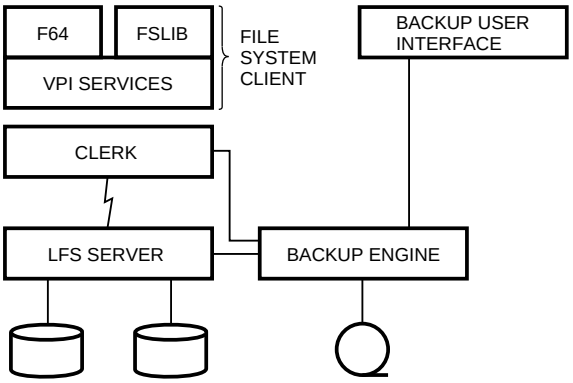

Figure 1
Spiralog Structure Overview

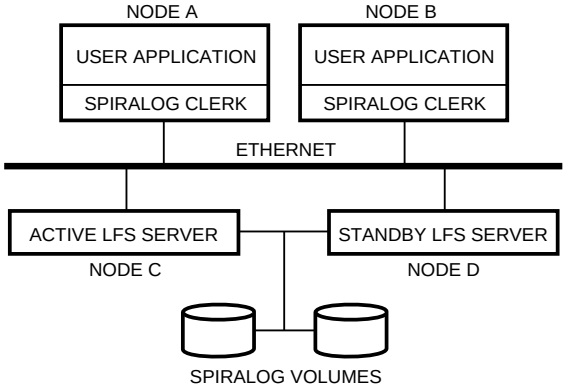

Figure 2
Spiralog Cluster Configuration

system client layer. It consists of a number of file system personalities and the underlying personality-independent services, which we call the VPI.

Two file system personalities dominate the Spiralog design. The F64 personality is an emulation of the Files-11 file system. The file system library (FSLIB) personality is an implementation of Microsoft's New Technology Advanced Server (NTAS) file services for use by the PATHWORKS for OpenVMS file server.

The next layer, present on all systems, is the clerk layer. It supports a distributed cache and ordered write back to the LFS server, giving single-system semantics in a cluster configuration.

The LFS server, the third layer, is present on all designated server systems. This component is responsible for maintaining the on-disk log structure; it includes the cleaner, and it is accessed by multiple clerks. Disks can be connected to more than one LFS server, but they are served only by one LFS server at a time. Transparent failover, from the point of view of the file system client layer, is achieved by cooperation between the clerks and the surviving LFS servers.

The backup engine is present on a system with an active LFS server. It uses the LFS server to access the on-disk data, and it interfaces to the clerk to ensure that the backup or restore operations are consistent with the clerk's cache.

Figure 2 shows a typical Spiralog cluster configuration. In this cluster, the clerks on nodes A and B are accessing the Spiralog volumes. Normally, they use the LFS server on node C to access their data. If node C should fail, the LFS server on node D would immediately provide access to the volumes. The clerks on nodes A and B would use the LFS server on node D, retrying all their outstanding operations. Neither user application would detect any failure. Once node C had recovered, it would become the standby LFS server.

File System Client Design

The file system client is responsible for the traditional file system functions. This layer provides files, directories, access arbitration, and file naming rules. It also provides the services that the user calls to access the file system.

VPI Services Layer The VPI layer provides an underlying primitive file system interface, based on the UNIX VFS switch. The VPI layer has two overall goals:

1. To support multiple file system personalities
2. To effectively scale to very large volumes of data and very large numbers of files

To meet the first goal, the VPI layer provides

- File names of 256 Unicode characters, with no reserved characters
- No restriction on directory depth
- Up to 255 sparse data streams per file, each with 64-bit addressing
- Attributes with 255 Unicode character names, containing values of up to 1,024 bytes
- Files and directories that are freely shared among file system personality modules

To meet the second goal, the VPI layer provides

- File identifiers stored as 64-bit integers
- Directories through a B-tree, rather than a simple linear structure, for log(n) file name lookup time

The VPI layer is only a base for file system personalities. Therefore it requires that such personalities are trusted components of the operating system. Moreover, it requires them to implement file access security (although there is a convention for storing access control list information) and to perform all necessary cleanup when a process or image terminates.

F64 File System Personality As previously stated, the Spiralog product includes two file system personalities, F64 and FSLIB. The F64 personality provides a service that emulates the Files-11 file system.[5] Its functions, services, available file attributes, and execution behaviors are similar to those in the Files-11 file system. Minor differences are isolated into areas that receive little use from most applications.

For instance, the Spiralog file system supports the various Files-11 queued I/O ($QIO) parameters for returning file attribute information, because they are used implicitly or explicitly by most user applications. On the other hand, the Files-11 method of reading the file header information directly through a file called INDEXF.SYS is not commonly used by applications and is not supported.

The F64 file system personality demonstrates that the VPI layer contains sufficient flexibility to support a complex file system interface. In a number of cases, however, several VPI calls are needed to implement a single, complex Files-11 operation. For instance, to do a file open operation, the F64 personality performs the tasks listed below. The items that end with (VPI) are tasks that use VPI service calls to complete.

- Access the file's parent directory (VPI)
- Read the directory's file attributes (VPI)
- Verify authorization to read the directory
- Loop, searching for the file name, by
  - Reading some directory entries (VPI)
  - Searching the directory buffer for the file name
  - Exiting the loop, if the match is found
- Access the target file (VPI)
- Read the file's attributes (VPI)
- Audit the file open attempt

FSLIB File System Personality The FSLIB file system personality is a specialized file system to support the PATHWORKS for OpenVMS file server. Its two major goals are to support the file names, attributes, and behaviors found in Microsoft's NTAS file access protocols, and to provide low run-time cost for processing NTAS file system requests.

The PATHWORKS server implements a file service for personal computer (PC) clients layered on top of the Files-11 file system services. When NTAS service behaviors or attributes do not match those of Files-11, the PATHWORKS server has to emulate them. This can lead to checking security access permissions twice, mapping file names, and emulating file attributes.

Many of these problems can be avoided if the VPI interface is used directly. For instance, because the FSLIB personality does not layer on top of a Files-11 personality, security access checks do not need to be performed twice. Furthermore, in a straightforward design, there is no need to map across different file naming or attribute rules. For reasons we describe later, in the VPI Results section, we chose not to pursue this design to its conclusion.

Clerk Design

The clerks are responsible for managing the caches, determining the order of writes out of the cache to the LFS server, and maintaining cache coherency within a cluster. The caches are write behind in a manner that preserves the order of dependent operations.

The clerk-server protocol controls the transfer of data to and from stable storage. Data can be sent as a multiblock atomic write, and operations that change multiple data items such as a file rename can be made atomically. If a server fails during a request, the clerk treats the request as if it were lost and retries the request.

The clerk-server protocol is idempotent. Idempotent operations can be applied repeatedly with no effects other than the desired one. Thus, after any number of server failures or server failovers, it is always safe to reissue an operation. Clerk-to-server write operations always leave the file system state consistent.

The clerk-clerk protocol protects the user data and file system metadata cached by the clerks. Cache coherency information, rather than data, is passed directly between clerks.

The file system caches are kept in the clerks. Multiple clerks can have copies of stabilized data, i.e., data that has been written to the server with the write acknowledged. Only one clerk can have unstabilized, volatile data. Data is exchanged between clerks by stabilizing it. When a clerk needs to write a block of data to the server from its cache, it uses a token interface that is layered on the clerk-clerk protocol.

The writes from the cache to the server are deferred as long as possible within the constraints of the cache protocol and the dependency guarantees.

Dirty data remains in the cache as long as 30 seconds. During that time, overwrites are combined within the constraints of the dependency guarantees. Furthermore, operations that are known to offset one another, such as freeing a file identifier and allocating a file identifier, are fully combined within the cache.

Eventually, some trigger causes the dirty data to be written to the server. At this point, several writes are grouped together. Write operations to adjacent, or overlapping, file locations are combined to form a smaller number of larger writes. The resulting write operations are then grouped into messages to the LFS server.

The clerks perform write behind for four reasons:

- To spread the I/O load over time
- To remove occluded data, which can result from repeated overwrites of a data block, from being transferred to the server

- To avoid writing data that is quickly deleted such as temporary files
- To combine multiple small writes into larger transfers

The clerks order dependent writes from the cache to the server; consequently, other clerks never see "impossible" states, and related writes never overtake each other. For instance, the deletion of a file cannot happen before a rename that was previously issued to the same file. Related data writes are caused by a partial overwrite, or an explicit linking of operations passed into the clerk by the VPI layer, or an implicit linking due to the clerk-clerk coherency protocol.

The ordering between writes is kept as a directed graph. As the clerks traverse these graphs, they issue the writes in order or collapse the graph when writes can be safely combined or eliminated.

LFS Server Design

The Spiralog file system uses a log-structured, on-disk format for storing data within a volume, yet presents a traditional, update-in-place file system to its users. Recently, log-structured file systems, such as Sprite, have been an area of active research.[1]

Within the LFS server, support is provided for the log-structured, on-disk format and for mapping that format to an update-in-place model. Specifically, this component is responsible for

- Mapping the incoming read and write operations from their simple address space to positions in an open-ended log
- Mapping the open-ended log onto a finite amount of disk space
- Reclaiming disk space by cleaning (garbage collecting) the obsolete (overwritten) sections of the log

Figure 3 shows the various mapping layers in the Spiralog file system, including those handled by the LFS server.

Incoming read and write operations are based on a single, large address space. Initially, the LFS server transforms the address ranges in the incoming operations into equivalent address ranges in an open-ended log. This log supports a very large, write-once address space.

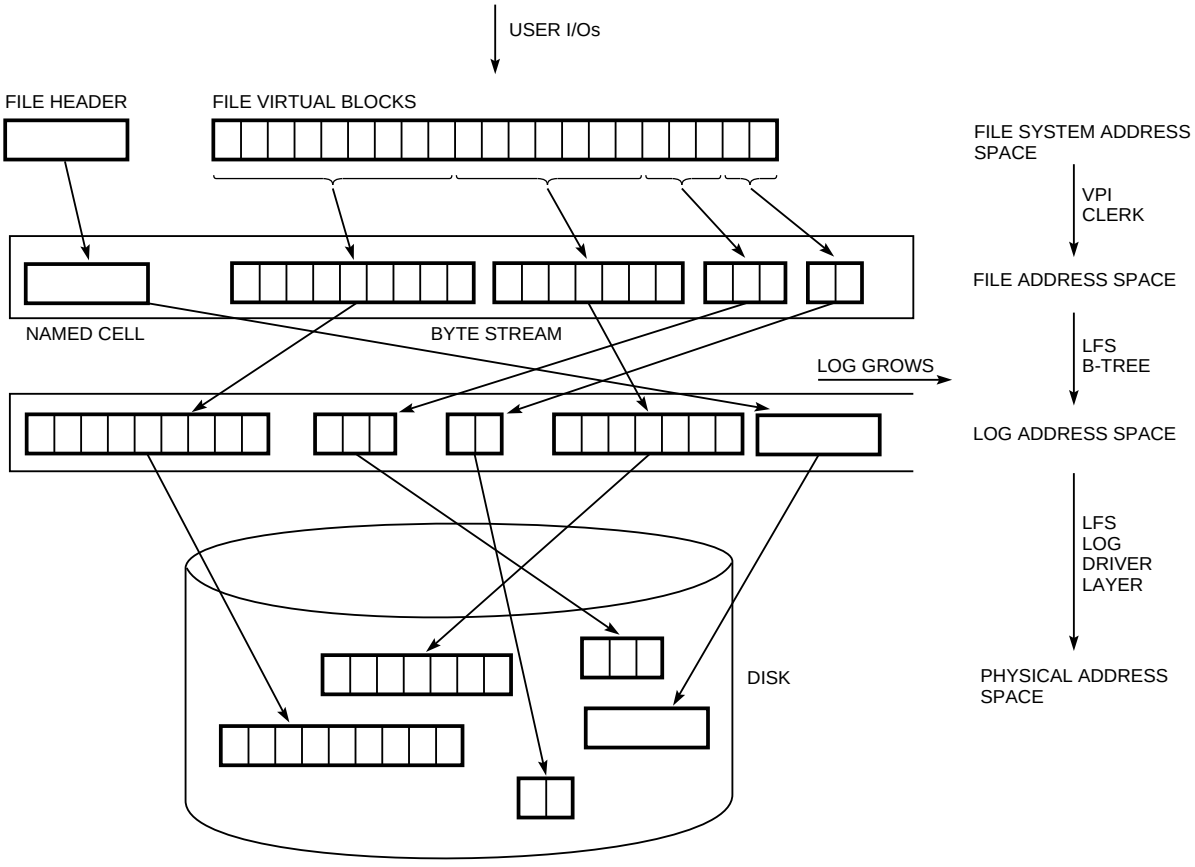

Figure 3
Spiralog Address Mapping

A read operation looks up its location in the open-ended log and proceeds. On the other hand, a write operation makes obsolete its current address range and appends its new value to the tail of the log.

In turn, locations in the open-ended log are then mapped into locations on the (finite-sized) disk. This additional mapping allows disk blocks to be reused once their original contents have become obsolete.

Physically, the log is divided into log segments, each of which is 256 kilobytes (KB) in length. The log segment is used as the transfer unit for the backup engine. It is also used by the cleaner for reclaiming obsolete log space.

More information about the LFS server can be found in this issue.[4]

On-line Backup Design

The design goals for the backup engine arose from higher storage management costs and greater data availability needs. Investigations with a number of customers revealed their requirements for a backup engine:

- Consistent save operations without stopping any applications or locking out data modifications
- Very fast save operations
- Both full and incremental save operations
- Restores of a full volume and of individual files

Our response to these needs influenced many decisions concerning the Spiralog file system design. The need for a high-performance, on-line backup led to a search for an on-disk structure that could support it. Again, we chose the log-structured design as the most suitable one.

A log-structured organization allows the backup facility to easily demarcate snapshots of the file system at any point in time, simply by marking a point in the log. Such a mark represents a version of the file system and prevents disk blocks that compose that version from being cleaned. In turn, this allows the backup to run against a low level of the file system, that of the logical log, and therefore to operate close to the spiral transfer rate of the underlying disk.

The difference between a partial, or incremental, and a full save operation is only the starting point in the log. An incremental save need not copy data back to the beginning of the log. Therefore, both incremental and full save operations transfer data at very high speed.

By implementing these features in the Spiralog file system, we fulfilled our customers' requirements for high-performance, on-line backup save operations. We also met their needs for per-file and per-volume restores and an ongoing need for simplicity and reduction in operating costs.

To provide per-file restore capabilities, the backup utility and the LFS server ensure that the appropriate file header information is stored during the save operation. The saved file system data, including file headers, log mapping information, and user data, are stored in a file known as a *saveset.* Each saveset, regardless of the number of tapes it requires, represents a single save operation.

To reduce the complexity of file restore operations, the Spiralog file system provides an off-line saveset merge feature. This allows the system manager to merge several savesets, either full or incremental, to form a new, single saveset. With this feature, system managers can have a workable backup save plan that never calls for an on-line full backup, thus further reducing the load on their production systems. Also, this feature can be used to ensure that file restore operations can be accomplished with a small, bounded set of savesets.

The Spiralog backup facility is described in detail in this issue.[3]

Project Results

The Spiralog file system contains a number of innovations in the areas of on-line backup, log-structured storage, clusterwide ordered write-behind caching, and multiple-file-system client support.

The use of log structuring as an on-disk format is very effective in supporting high-performance, on-line backup. The Spiralog file system retains the previously documented benefits of LFS, such as fast write performance that scales with the disk size and throughput that increases as large read caches are used to offset disk reads.[1]

It should also be noted that the Files-11 file system sets a high standard for data reliability and robustness. The Spiralog technology met this challenge very well: as a result of the idempotent protocol, the cluster failover design, and the recover capability of the log, we encountered few data reliability problems during development.

In any large, complex project, many technical decisions are necessary to convert research technology into a product. In this section, we discuss why certain decisions were made during the development of the Spiralog subsystems.

VPI Results

The VPI file system was generally successful in providing the underlying support necessary for different file system personalities. We found that it was possible to construct a set of primitive operations that could be used to build complex, user-level, file system operations.

By using these primitives, the Spiralog project members were able to successfully design two distinctly different personality modules. Neither was a functional superset of the other, and neither was layered on top of the other. However, there was an important second-order problem.

The FSLIB file system personality did not have a full mapping to the Files-11 file system. As a consequence, file management was rather difficult, because all the data management tools on the OpenVMS operating system assumed compliance with a Files-11, rather than a VPI, file system.

This problem led to the decision not to proceed with the original design for the FSLIB personality in version 1.0 of Spiralog. Instead, we developed an FSLIB file system personality that was fully compatible with the F64 personality, even when that compatibility forced us to accept an additional execution cost.

We also found an execution cost to the primitive VPI operations. Generally, there was little overhead for data read and write operations. However, for operations such as opening a file, searching for a file name, and deleting a file, we found too high an overhead from the number of calls into the VPI services and the resulting calls into the cache manager. We called this the "fan-out" problem: one high-level operation would turn into several VPI operations, each of which would turn into several cache manager calls. Table 1 gives the details of the fan-out problem.

We believe that it would be worthwhile to provide slightly more complex VPI services in order to combine calls that always appear in the same sequence.

Table 1
Call Fan-out by Level

Operation	F64 Calls	VPI Calls	Clerk Calls	Revised Clerk Calls
Create file	4	18	29	24
Open file	1	6	18	14
Read block	1	1	3	3
Write block	2	4	7	6
Close file	1	4	13	10

Clerk Results

The clerk met a number of our design goals. First, the use of idempotent operations allowed failover to standby LFS servers to occur with no loss of service to the file system clients, and with little additional complexity within the clerk.

Second, the ordered write behind proved to be effective at ordering dependent, metadata file system operations, thus supporting the ability to construct complex file system operations out of simpler elements.

Third, the clerk was able to manage large physical caches. It is very effective at making use of unused pages when the memory demand from the OpenVMS operating system is low, and at quickly shrinking the cache when memory demands increase. Although certain parameters can be used to limit the size of a clerk's cache, the caches are normally self-tuning.

Fourth, the clerks reduce the number of operations and messages sent to the LFS server, with a subsequent reduction to the number of messages and operations waiting to be processed. For the COPY command, the number of operations sent to the server was typically reduced by a factor of 3. By using transient files with lifetimes of fewer than 30 seconds, we saw a reduction of operations by a factor of 100 or more, as long as the temporary file fit into the clerk's cache.

In general, the code complexity and CPU path length within the clerk were greater than we had originally planned, and they will need further work. Two aspects of the services offered by the clerk compounded the cost in CPU path length. First, the clerk has a simple interface that supports reads and writes into a single, large address space only. This interface requires a number of clerk operations for a number of the VPI calls, further expanding the call fan-out issues. Second, a concurrency control model allows the clerk to unilaterally drop locks. This requires the VPI layer to revalidate its internal state with each call.

Either a change to the clerk and VPI service interfaces to support notification of lock invalidation, or a change to the concurrency control model to disallow locks that could be unilaterally invalidated, would reduce the number of calls made. We believe such changes would produce the results given in the last column of Table 1.

LFS Server Results

The LFS server provides a highly available, robust file system server. Under heavy write loads, it provides the ability to group together multiple requests and reduce the number of disk I/Os. In a cluster configuration, it supports failover to a standby server.

In normal operation, the cleaner was successful in minimizing overhead, typically adding only a few percent to the elapsed time. The cleaner operated in a lazy manner, cleaning only when there was an immediate shortage of space. The cleaner operations were further lessened by the tendency for normal file overwrites to free up recently filled log segments for reuse.

Although this produced a cleaner that operated with little overhead, it also brought about two unusual interactions with the backup facility. In the first place, the log often contains a number of obsolete areas that

are eligible for cleaning but have not yet been processed. These obsolete areas are also saved by the backup engine. Although they have no effect on the logical state of the log, they do require the backup engine to move more data to backup storage than might otherwise be necessary.

Second, the design initially prohibited the cleaner from running against a log with snapshots. Consequently, the cleaner was disabled during a save operation, which had the following effects: (1) The amount of available free space in the log was artificially depressed during a backup. (2) Once the backup was finished, the activated cleaner would discover that a great number of log segments were now eligible for cleaning. As a result, the cleaner underwent a sudden surge in cleaning activity soon after the backup had completed.

We addressed this problem by reducing the area of the log that was off-limits to the cleaner to only the part that the backup engine would read. This limited snapshot window allowed more segments to remain eligible for cleaning, thus greatly alleviating the shortage of cleanable space during the backup and eliminating the postbackup cleaning surge. For an 8-gigabyte time-sharing volume, this change typically reduced the period of high cleaner activity from 40 seconds to less than one-half of a second.

We have not yet experimented with different cleaner algorithms. More work needs to be done in this area to see if the cleaning efficiency, cost, and interactions with backup can be improved.

The current mapping transformation from the incoming operation address space to locations in the open-ended log is more expensive in CPU time than we would like. More work is needed to optimize the code path.

Finally, the LFS server is generally successful at providing the appearance of a traditional, update-in-place file system. However, as the unused space in a volume nears zero, the ability to behave with semantics that meet users' expectations in a log-structured file system proved more difficult than we had anticipated and required significant effort to correct.

The LFS server is described in much more detail in this issue.[4]

Backup Performance Results

We took a new approach to the backup design in the Spiralog system, resulting in a very fast and very low impact backup that can be used to create consistent copies of the file system while applications are actively modifying data. We achieved this degree of success without compromising such functionality as incremental backup or fast, selective restore.

The performance improvements of the Spiralog save operation are particularly noticeable with the large numbers of transient or active files that are typically found on user volumes or on mail server volumes. In the following tables, we compare the Spiralog and the file-based Files-11 backup operations on a DEC 3000 Model 500 workstation with a 260-MB volume, containing 21,682 files in 401 directories and a TZ877 tape.

Table 2 gives the results of two save operations, which are the average of five operations. Although its saveset size is somewhat larger, the Spiralog save operation completes nearly twice as fast as the Files-11 save operation.

Table 3 gives the results from restoring a single file to the target volume. In this case, the Spiralog file restore operation executes more than three times as fast as the Files-11 system.

The performance advantage of the Spiralog backup and restore facility increases further for large, multitape savesets. In these cases, the Spiralog system is able to omit tapes that are not needed for the file restore; the Files-11 system does not have this capability.

Observations and Conclusions

Overall, we believe that the significant innovation and real success of the Spiralog project was the integration of high-performance, on-line backup with the log-structured file system model. The Spiralog file system delivers an on-line backup engine that can run near device speeds, with little impact on concurrently running applications. Many file operations are significantly faster in elapsed time as a result of the reduction in I/Os due to the cache and the grouping of write operations. Although the code paths for a number of operations are longer than we had planned, their

Table 2
Performance Comparison of the Backup Save Operation

File System	Elapsed Time (Minutes:Seconds)	Saveset Size (MB)	Throughput (MB/s)
Spiralog	05:20	339	1.05
Files-11	10:14	297	0.48

Table 3
Performance Comparison of the Individual File Restore Operation

File System	Elapsed Time (Minutes:Seconds)
Spiralog	01:06
Files-11	03:35

length is mitigated by continuing improvements in processor performance.

We learned a great deal during the Spiralog project and made the following observations:

- Volume full semantics and fine-tuning the cleaner were more complex than we anticipated and will require future refinement.
- A heavily layered architecture extends the CPU path length and the fan-out of procedure calls. We focused too much attention on reducing I/Os and not enough attention on reducing the resource usage of some critical code paths.
- Although elegant, the memory abstraction for the interface to the cache was not as good a fit to file system operations as we had expected. Furthermore, a block abstraction for the data space would have been more suitable.

In summary, the project team delivered a new file system for the OpenVMS operating system. The Spiralog file system offers single-system semantics in a cluster, is compatible with the current OpenVMS file system, and supports on-line backup.

Future Work

During the Spiralog version 1.0 project, we pursued a number of new technologies and found four areas that warrant future work:

- Support is needed from storage and file-management tools for multiple, dissimilar file system personalities.
- The cleaner represents another area of ongoing innovation and complex dynamics. We believe a better understanding of these dynamics is needed, and design alternatives should be studied.
- The on-line backup engine, coupled with the log-structured file system technology, offers many areas for potential development. For instance, one area for investigation is continuous backup operation, either to a local backup device or to a remote replica.
- Finally, we do not believe the higher-than-expected code path length is intrinsic to the basic file system design. We expect to be working on this resource usage in the near future.

Acknowledgments

We would like to take this opportunity to thank the many individuals who contributed to the Spiralog project. Don Harbert and Rich Marcello, OpenVMS vice presidents, supported this work over the lifetime of the project. Dan Doherty and Jack Fallon, the OpenVMS managers in Livingston, Scotland, had day-to-day management responsibility. Cathy Foley kept the project moving toward the goal of shipping. Janis Horn and Clare Wells, the product managers who helped us understand our customers' needs, were eloquent in explaining our project and goal to others. Near the end of the project, Yehia Beyh and Paul Mosteika gave us valuable testing support, without which the product would certainly be less stable than it is today. Finally, and not least, we would like to acknowledge the members of the development team: Alasdair Baird, Stuart Bayley, Rob Burke, Ian Compton, Chris Davies, Stuart Deans, Alan Dewar, Campbell Fraser, Russ Green, Peter Hancock, Steve Hirst, Jim Hogg, Mark Howell, Mike Johnson, Robert Landau, Douglas McLaggan, Rudi Martin, Conor Morrison, Julian Palmer, Judy Parsons, Ian Pattison, Alan Paxton, Nancy Phan, Kevin Porter, Alan Potter, Russell Robles, Chris Whitaker, and Rod Widdowson.

References

1. 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.
2. T. Mann, A. Birrell, A. Hisgen, C. Jerian, and G. Swart, "A Coherent Distributed File Cache with Directory Write-behind," Digital Systems Research Center, Research Report 103 (June 1993).
3. R. Green, A. Baird, and J. Davies, "Designing a Fast, On-line Backup System for a Log-structured File System," *Digital Technical Journal,* vol. 8, no. 2 (1996, this issue): 32–45.
4. 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.
5. M. Howell and J. Palmer, "Integrating the Spiralog File System into the OpenVMS Operating System," *Digital Technical Journal,* vol. 8, no. 2 (1996, this issue): 46–56.
6. R. Wrenn, "Why the Real Cost of Storage is More Than $1/MB," presented at U.S. DECUS Symposium, St. Louis, Mo., June 3–6, 1996.

Biographies

James E. Johnson
Jim Johnson, a consulting software engineer, has been working for Digital since 1984. He was a member of the OpenVMS Engineering Group, where he contributed in several areas, including RMS, transaction processing services, the port of OpenVMS to the Alpha architecture, file systems, and system management. Jim recently joined the Internet Software Business Unit and is working on the application of X.500 directory services. Jim holds two patents on transaction commit protocol optimizations and maintains a keen interest in this area.

William A. Laing
Bill Laing, a corporate consulting engineer, is the technical director of the Internet Software Business Unit. Bill joined Digital in 1981; he worked in the United States for five years before transferring to Europe. During his career at Digital, Bill has worked on VMS systems performance analysis, VAXcluster design and development, operating systems development, and transaction processing. He was the technical director of OpenVMS engineering, the technical director for engineering in Europe, and most recently was focusing on software in the Technology and Architecture Group of the Computer Systems Division. Prior to joining Digital, Bill held research and teaching posts in operating systems at the University of Edinburgh, where he worked on the EMAS operating system. He was also part of the start-up of European Silicon Structures (ES2), an ambitious pan-European company. He holds undergraduate and postgraduate degrees in computer science from the University of Edinburgh.