CPEEDY::BRADLEY

JOB 95

GOLDSTEIN–STORAGE–MGMT

Owner UIC: **[YAH,BRADLEY]**
Account: **YAH**

Priority: 100
Submit queue: LKG21_16
Submitted: 4–AUG–1998 12:28
Printer queue: LKG21_16
Printer device: 4LPS16::
Started: 4–AUG–1998 12:50

Digital Equipment Corporation
OpenVMS Alpha V7.1

PrintServer 32 4LPS16
DECprint Supervisor V1.5

INVESTIGATION REPORT
STORAGE MANAGEMENT IN VMS

Andy Goldstein

X0.2 - 3-Aug-1998

ABSTRACT

VMS faces an imminent crisis in storage management. Effective use and management of storage systems of the scale we expect in the near future is not possible with the currently available tools. We propose an overall storage architecture and a set of products, both short and longer term, to address this problem.

REVISION HISTORY

X0.1	5-Jan-1998	Andy Goldstein	Initial draft
X0.2	3-Aug-1998	Andy Goldstein	Incorporate progress on BACKUP performance and XQP large volume support; initial draft on snap disk based backup.

DIGITAL Confidential

ACKNOWLEDGMENTS

This Investigation report draws on work by a number of people:

- Steve Zalewski: for the Galaxy concept, which brings into sharp focus the expected scale of future VMS systems.
- Russ Green, Chris Whitaker, Rob Burke, Kevin Playford, and others in EDO: for Spiralog Backup, Snap Capable Disk, the Cluster File System, and various investigations into storage management and backup.
- Dave Thiel and Richie Lary: for data on the future growth of mass storage systems and seminal ideas on the concept of snapshots as backups.

REFERENCES

- `http://dopey.vmse.edo.dec.com/edoweb/CFS/Backup.htm`
  - Fast Backup Investigation Report
  - Fast Backup White Paper
- `http://dopey.vmse.edo.dec.com/edoweb/snaps/documents.htm`
  - OpenVMS Snapshot Services Investigation Report
  - OpenVMS Snapshot Services Functional Specification
  - OpenVMS Snapshot Services Design Specification

Table of Contents

1. EXECUTIVE SUMMARY

Over the past 15 years, the volume of data storage of typical VMS installations has increased a thousand fold. Disk performance has not kept pace, meaning that backups take longer and longer. At the same time, growing uptime requirements have reduced the time available for backups to the point where conventional backup techniques are simply impossible.

At the same time, the number of disk spindles in large scale installations has increased to the point where existing mechanisms for space and fault management are inadequate.

We make the following recommendations:

- Make a set of small improvements to existing VMS components too allow them to realize their full potential in the short term:
  - Buffering improvements in VMS BACKUP to improve performance
  - Other often requested enhancements in BACKUP and ongoing performance support
  - A set of changes to the XQP to support larger volumes and to allow volumes to be dynamically expanded.
- Develop a new backup product based on Snap Capable disk
- Design an overall storage architecture for VMS to describe the complete storage management solution involving base VMS components, Digital layered products, and selected third party products.

In addition to developing solutions for VMS, we plan to apply our VMS experience and storage expertise to business opportunities in the Windows NT market by porting or co-developing appropriate products on Windows NT.

2. PROBLEM STATEMENT

VMS faces an imminent crisis in storage management. The current situation results from the trends of the past 20 years, but the advent of Galaxy systems brings the issue into sharp focus.

Typical present-day high end VMS systems are built around 2 to 4 AlphaServer 8400s with 1-2 CPUs each, yielding a total compute power of 33 to 130 SpecInt95. Typical high end storage configurations are 2 to 4 terabytes.

A first-generation Galaxy system built on Wildfire can deliver as much as 1200 SpecInt95 of processing power, and as much as 128 GB of main memory. A simple application of Amdahl's law says we can expect such a system to be accompanied by 40 TB of online storage. Later generation Wildfire systems are expected to scale to 7000 SpecInt95 and 1 TB of main memory, with a corresponding scaling of mass storage to 240 TB. Use of this scale of mass storage is more than theoretical. The combination of Wildfire and Fiberchannel will provide more than adequate bandwidth and connectivity to make storage farms of this scale realizable.

Another line of reasoning leads us to a similar conclusion: We are evaluating a 1 million TPM benchmark for a Galaxy class system. To achieve the storage throughput needed for this benchmark, we will need 10,000 disk spindles. For a disaster-tolerant production installation of equivalent capacity, the number of disk drives increases to 30,000 because of the additional drives needed for shadowing, RAID parity, and spares.

When a customer assembles a mass storage facility of the dimensions we expect in future systems and fills it with data that is considered valuable, managing the storage is not possible with today's tools.

2.1 Opportunities

We are hardly operating in a vacuum. A number of efforts currently underway promise elements of a solution to the problems described. What we need is to tie these efforts together into a coherent storage strategy for VMS. In doing so, we can identify and correct shortcomings to provide VMS customers with storage offerings that work at the scale we expect them to operate.

- Argus storage management. The Argus team has already developed an interface for VMS storage configuration management. This product goes a long way toward rationalizing the existing VMS storage management features. Other present and future storage management products should be unified under the Argus interface.
- Other storage management and aggregation products, such as the Storageworks hardware and software offerings. A number of existing products offer solutions to parts of the space and fault management problem set. A coherent selection of these products needs to be included in any comprehensive solution.

- Cluster file system. EDO is developing a new file system for deployment on both VMS and Windows NT. To the extent that backup products have knowledge of the file structure (which is considerable), they must become cognizant of the new file system.
- Snap capable disk. The snap capable disk product under development at EDO provides two very important opportunities:
  - Logical volume management - the ability to organize a storage pool of disk units into a manageable set of logical volumes.
  - Snapshots. A snapshot is an inexpensive virtual copy of a logical volume. Snapshot technology provides a unique opportunity to "break the rules of physics" and solve the backup bandwidth problem. This opportunity has been under study at EDO for some time, and papers are available.

  In addition to the software snap capable disk product under development at EDO, storage engineering is planning a future generation of storage controllers that implement equivalent capabilities.
- While a long term solution to the backup problem requires a radically new design (such as snap capable disk), there are some incremental changes that can be made to the existing VMS backup to realize the best performance possible within the existing design.
- SLS and ABS. These are existing archive, backup and storage library management products maintained by storage engineering. Any new capabilities we implement in VMS must integrate with these products. In addition, there are third party products such as Legato Networker that are worth looking into.

2.2 Storage Management Problems

A considerable number of storage management tools exist for VMS. However, they provide only partial coverage for storage management needs. Many of these tools come from other organizations in Digital or from third parties. Our understanding of these products is incomplete, and we do not provide our customers with a consistent and complete storage management message.

Storage management must address the following functions:

- Space management. Any large installation will have a significant rate of growth and change in its storage use. System managers must be able to allocate and reallocate storage for different applications with a minimum of time and energy. Having the large number of individual storage devices be visible at the logical storage level is not a manageable situation.
- Failure management. As reliable as disk drives have become, simple probability says that a large installation will have to deal with a significant failure rate. System managers must be able to deal with disk failures on a routine basis with a minimum of attention and no loss of data.
- Backup. Simply put, applying conventional backup techniques to storage of this scale violates the laws of physics. While disk sizes have grown about 2 orders of magnitude over the last 15 years, disk performance has grown by about 1 order of magnitude. The problem is actually worse than this metric implies for a couple of reasons:

- Typical VMS installations have about an order of magnitude more disk spindles than they did 15 years ago (yielding a total 1000 fold increase in storage volume).
- Customers continue to prefer file-oriented backups, since most recoveries involve individual files rather then the entire volume. Despite attempts at optimization, file-oriented backups continue to be seek-intensive. Disk seek performance has increased only ½ order of magnitude over the last 15 years.
- Because of growing uptime demands, the backup window of VMS installations is shrinking. 15 years ago many installations ran essentially 8-5, leaving 12 hours or more for backups. In many installations, the backup windows is now 1 hour, rapidly shrinking to zero as literal 24x365 becomes reality. Backups are often done concurrently with at least some file activity. This is another reason why purely physical backups are not an option for many customers.

Through the combination of these factors, many large scale VMS installations are already at a crisis stage with their backup performance. Note that the figures given above describe the present. The past trends will continue, but at an accelerated pace. For example, we expect disks with 40 GB per spindle in a year or two.

3. SOLUTION OVERVIEW

This investigation yields a broad range of results, and it is not practical to address all aspects of the solution in a single project. Rather, we make a number of recommendations. Some are tactical, short term fixes to existing VMS components. Others are more general solutions that will take longer to implement. This report is organized into separate sections for the various recommendations.

The investigation phase of some of the more far-reaching proposals will take considerable time. At the same time, the tactical proposals are well understood and should not be held up by other investigations. Consequently this report will remain a work in progress for a considerable length of time. Where investigation is pending or underway, the report describes our current thinking on the subject and recommendations for further investigation. New editions will appear as investigations make significant progress.

The following sections provide overviews of the major investigation areas.

3.1 XQP - Large Volume Support

3.1.1 Storage Bitmap Size

The current XQP implementation limits the storage bitmap to 255 blocks. This translates to about 1 million allocatable units. So far we have managed to squeak by, but the advent of 9 GB disk drives raises the allocation cluster size to 18 blocks, which crosses the threshold of pain for interactive environments. The XQP logic and internal data structures need to be changed to allow a larger bitmap size.

3.1.2 Logical Volume vs Physical Volume

There are some assumptions in the current XQP implementation that the logical volume (i.e., the volume that the storage bitmap is sized for) is the same as the physical volume containing the file structure. The size of the latter is defined by UCB$L_MAXBLOCK. These assumptions should be removed, and references to the physical volume size should be changed to a separate logical volume size. This value is already recorded in the volume's storage control block. Making this change allows shadowing of dissimilar size devices without any intervening virtual device layer or other fiction to preserve the logical / physical volume correspondence.

3.1.3 Dynamic Volume Growth

Many storage aggregation products (e.g., snap capable disk and other logical volume managers) have the interesting potential of allowing a logical storage volume to grow in size. The same thing happens when larger disks are used to replace smaller disks in a VMS shadow set. A function to

grow the logical volume size should be added to the XQP to take advantage of this characteristic. Note that logical vs physical volume separation (section 3.1.2) is a prerequisite for this feature.

3.2 BACKUP - Improved Buffer Handling

There have been innumerable complaints from VMS installations that the existing VMS BACKUP is too slow. Much of this, of course, is simply that we are up against the laws of physics in moving the volume of data dictated by the current backup architecture. However, there is also considerable evidence that BACKUP does not run at its full performance potential.

We know that in the common disk to tape save mode, BACKUP uses a large buffer pool to read file data optimized to minimize disk seek time. However, disk and tape I/O are not run concurrently. After filling the buffer pool and initiating the tape writes, BACKUP waits for all tape I/O to complete before starting the next disk read pass. Aside from the lost I/O concurrency, starting and stopping a streaming tape drive incurs a disproportionate performance penalty. The end result is that backups to a streaming tape often run at significantly less than even the available disk bandwidth.

We need two activities to remedy this problem:

1. A performance study of BACKUP to understand its present performance behavior and determine the potential performance that might be realized within the existing design.
2. Changes to the buffering and I/O algorithms to allow concurrent disk and tape I/O, at least in circumstances where this will improve performance.

3.3 Snap Capable Disk Based Backup

Snap-capable disk is a facility that provide inexpensive virtual copies of a volume. Virtual volumes are mapped into a storage pool of real disks. A snapshot is simply another virtual volume that, on creation, shares the mapping of its parent volume. As the parent continues to be updated, the mapping is changed so that the snapshot retains the original data.

Snap-capable disk provide an opportunity to "break the laws of physics" as applies to backup. There are two levels of development worth considering:

1. Backups of snapshots
2. Snapshots as backups

3.3.1 Backups of snapshots

In simple terms, taking a snapshot solves the problem of the shrinking backup window. The system manager simply takes a snapshot of the volume to be backed up, and then makes a backup of the snapshot by conventional means. After the backup is complete the snapshot can be deleted. The backup window thus expands to the entire time interval between backups. This approach minimizes the performance impact of the snapshot because the snapshot only exists while the backup is being run. It fits well with installations that have demanding uptime requirements, but have periods of reduced application load that accommodate the snapshot overhead and the bandwidth required by the backup.

The principal disadvantage of this approach is that it does not solve the bandwidth vs capacity dilemma. The same amount of data must still be copied from storage onto backup media. Furthermore, we must keep in mind that behind the virtual disks that represent the primary volume

and its snapshot, there is still a common set of real media and disk heads. Bandwidth taken by the backup is not available to the application. Thus a full backup remains a painful proposition.

3.3.2 Snapshots as backups

A more radical approach to backups is to simply view the snapshot as the backup. In its crudest form, this simply means taking periodic snapshots (e.g., at the same frequency that one would do incremental backups) and keeping them forever. To recover a file, one simply copies the file from a snapshot containing a correct copy of the file back to the parent volume. Note that such a recovery can be done by any user without system manager or operator involvement, provided the snapshots are kept mounted as public volumes.

This brute force approach suffers from several fatal practical limitations:

- As much as we would like to sell disks, most installations are unwilling to continuously add real disk storage to maintain copies of data that existed in the past.
- Snapshot Services for OpenVMS supports only 12 concurrent snapshots of a volume. While this limit could be increased, any snapshot product will have some finite limit on the number of snapshots than can be retained online.
- Online snapshots provide for individual file recovery, but do not address the system manager's worst nightmare: loss of an entire volume, or worse, loss of the entire storage subsystem. Techniques such as shadowing or RAID5 effectively address media failures, but do not address failures at a higher level or of a larger scope. Disasters are only addressed with a complete offline copy of the entire storage subsystem.

These limitations dictate that it must be possible to migrate snapshots to offline storage and back. We define a new style of backup: the physical incremental backup. A snapshot may be taken offline by copying out the data necessary to recreate the snapshot from other data that is still online, such as either the parent volume, or some other snapshot. The data is precisely the set of data in which the snapshot differs from the data retained online.

To recover files from a snapshot, the entire snapshot must be brought back online. The amount of space required to bring a snapshot back online is simply the size of the portion of the snapshot that differs from some other snapshot that is currently online.

Physical incremental saves may be consolidated offline by merging their contents. Because the semantics of a logical volume are extremely simple, consolidation of physical incrementals avoids all the obscure problems associated with consolidating file-structured save sets. Consolidating a base full save with subsequent incrementals yields the exact equivalent of a current full save. Because consolidation can be done completely offline with separate equipment, we completely eliminate the bandwidth demand of periodic full saves in the primary data store.

3.4 Other BACKUP Improvements

While snap-based backup carries a lot of promise, it is not the solution to all the world's problems. Some applications may not be able to use snap capable disk because of its performance penalty, or because the rate of change on the data is too high. In addition, customers have repeatedly requested some additional features in BACKUP that we need to provide for a comprehensive solution. The following areas need to be investigated:

3.3.3 Multiple copies of save sets

Customers in many circumstances need multiple copies of a backup, for example, one locally available for file recovery and one off-site for disaster recovery. At present, they accomplish this by making two backups, or by copying the backup offline. Either method is tedious. BACKUP could just as well write multiple copies of the save set onto multiple tapes while the data is available in memory.

3.3.4 Multiple interleaved save sets

For file-based backups of interactive and file service environments, the seek time associated with gathering up many small files makes the net disk bandwidth considerably less than that of the tape. Legato Networker takes a pragmatic approach to this problem by allowing multiple save sets to be interleaved on the same tape. The installation can aggregate the bandwidth of as many storage volumes as necessary to match the bandwidth of the tape.

This capability should be added to the VMS ABS product. Support is needed in the BACKUP tape format to allow the interleaving of multiple save sets on the same tape.

3.3.5 Save set striped over multiple tapes

Despite the discouraging trends in disk bandwidth vs capacity described earlier, recent developments in storage technology suggest we may see dramatic bandwidth increases in the near future. The bandwidth of I/O controllers and busses is increasing rapidly, and caching and RAID techniques serve to multiply the effective bandwidth of disk storage. We may yet succeed in upsetting the conventional wisdom that tapes are faster than disks.

In some circumstances snap capable disk may not be a viable solution, and a brute force approach may be in order, where we simply dump a large database as fast as the total available I/O bandwidth allows. Since aggregating multiple disks multiplies the disk bandwidth, it makes sense to apply the same technique to the tape, splitting the save set over multiple tapes.

3.3.6 Selective file restore from physical save set

A physical save has always been the fastest possible way to copy a disk to tape. It has never been widely accepted because of two problems:

1. It can't be done online while there is any file activity.
2. Recovering individual files is tedious, because the entire save set must first be restored to a scratch volume.

Snap capable disk addresses problem 1 by allowing users to make a physical backup from a stable snapshot. Problem 2 can be solved by allowing a physical save set to be addressed as a virtual disk, allowing direct file recovery from the tape. Selective file recovery from file-based save sets already requires scanning a major fraction of the tape, so this is not a significant acceptance issue. Some intelligent caching is necessary for acceptable performance, since the blocks needed to find a file through the directory structure are not in sequential order on the tape. For example, a heuristic could recognize and pre-load directory and file header data into the cache as it is encountered on the tape, rather than as it is needed.

3.3.7 Ongoing performance support

The backup process by nature stretches a system's I/O capabilities to their extreme, and so is vulnerable to all manner of bottlenecks and tuning problems. We must provide continued support for inquiries on performance problems in BACKUP, both from customers and internal testing efforts. More and clearer documentation on configuring a system for optimum backup performance would help a great deal. Finally, we could provide some tools to measure the I/O bandwidth of a particular configuration as a diagnostic tool for customers.

3.4 Storage Management Architecture

4. XQP - LARGE VOLUME SUPPORT

4.1 Storage Bitmap Size

$2^8 \cdot 2^9 \cdot 2^3 = 2^{20}$

The current XQP implementation limits the storage bitmap to 255 blocks. This translates to about 1 million allocatable units. So far we have managed to squeak by, but the advent of 9 GB disk drives raises the allocation cluster size to 18 blocks, which crosses the threshold of pain for interactive environments. The XQP logic and internal data structures need to be changed to allow a larger bitmap size.

4.1.1 Investigation Results

Aside from low level design details, this investigation of this area is complete. The following sections describe work required to address the problem. This work should ship in Raven if possible. Increasing the storage map size is an upwards but not downwards compatible change; older VMS systems cannot handle the larger bitmap. However, MOUNT checks the storage map size on existing VMS versions, so it is not strictly necessary to tie this change to ODS-5. However, packaging it in the same release as ODS-5 will reduce the potential for incompatibilities and customer confusion.

4.1.2 Design Considerations

The original file system design allowed for 1 byte of storage bitmap size and offset in the data structure fields and algorithms. Fixing this problem mainly involves using larger fields to address the bitmap. While the storage bitmap is the primary issue, the index file bitmap is also limited to 255 blocks, and should be expanded at the same time.

4.1.2.1 VCB Fields

The internal data structure changes to allow larger bitmaps were made approximately 10 years ago. The following fields are currently allocated as words in the VAX XQP VCB:

VCB$W_SBMAPSIZE	size of storage bitmap
VCB$W_SBMAPVBN	current working offset in storage map
VCB$W_IBMAPSIZE	size of index file bitmap
VCB$W_IBMAPVBN	current working offset in index file map

In the Alpha XQP, all these fields have been promoted to longwords.

Implementation in the XQP simply involves using the larger fields (the VAX code base has byte overlays that are still referenced by the bitmap code). In addition, the code must be inspected for byte local variables containing copies of these fields. The word size fields limit the VAX XQP to a

65535 block bitmap, or 268 million allocatable clusters. This means the largest disk that can be handled with a cluster factor of 1 is 137 GB. I think this is a restriction we can live with.

4.1.2.2 Allocation Lock Value Block

The SBMAPVBN and IBMAPVBN fields are passed in the allocation lock value block as hints for bitmap searches. Unfortunately the value block fields are still bytes, and the value block is full. The best way to handle this problem is to scale the field values passed in the allocation lock value block so that they fit: simply divide the xBMAPVBN value by a suitable factor such that the highest possible value scales to 255. Losing granularity is acceptable, since the value is simply a starting hint for the search.

4.1.2.3 MOUNT

MOUNT enforces the 255 block bitmap limit. The limit needs to be raised appropriately (65K on VAX and 4G on Alpha).

The volume rebuild logic in MOUNT (also included in SET VOLUME/REBUILD) allocates buffers in virtual memory that match the size of the bitmaps. There is no need to redesign this logic - we simply observe that volumes with very large bitmaps will require a large virtual address space to rebuild. In addition, the logic should be inspected for size assumptions. To the best of my knowledge, there are none.

4.1.2.4 INITIALIZE

INITIALIZE adjusts the cluster factor to keep the storage map under 255 blocks. An explicit cluster factor that would cause the bitmap to exceed 255 blocks causes an error. This logic should be changed to limit the bitmaps to 65K blocks. Enforcing the limit on both VAX and Alpha means all volumes will remain mountable on VAXes. The Alpha limit can be raised in the future as circumstances warrant.

Because volumes with storage maps larger than 255 blocks cannot be mounted on older VMS versions, the defaults for INITIALIZE must be chosen carefully. I recommend the following defaults that depend on the volume structure level:

- ODS-2: by default, choose a cluster factor that limits the storage map to 255 blocks, so that the volume can be mounted on older VMS systems. The user can still explicitly choose a cluster factor that makes the storage map larger.
- ODS-5: by default, choose a small cluster factor (probably 3, for lack of anything better) and allow the storage map to be large.

4.1.2.5 BACKUP

BACKUP likewise enforces a 255 block limit on the bitmaps that needs to be raised. BACKUP should also be inspected for other assumptions on the size of the bitmaps. To the best of my knowledge there aren't any.

4.1.2.6 ANALYZE/DISK and SALVAGE

ANALYZE/DISK and SALVAGE allocate virtual memory buffers for the bitmaps in the same manner as the MOUNT REBUILD logic. The same considerations apply. In addition, SALVAGE has logic similar to INITIALIZE that limits the size of the storage bitmap to 255 blocks. This logic should track the changes to INITIALIZE above.

4.1.3 Status

With the exception of the SALVAGE modifications, this work has been completed and will ship in Raven (VMS V7.2).

4.2 Logical Volume vs Physical Volume

There are some assumptions in the current XQP implementation that the logical volume (i.e., the volume that the storage bitmap is sized for) is the same as the physical volume containing the file structure. The size of the latter is defined by UCB$L_MAXBLOCK. These assumptions should be removed, and references to the physical volume size should be changed to a separate logical volume size. This value is already recorded in the volume's storage control block. Making this change allows shadowing of dissimilar size devices without any intervening virtual device layer or other fiction to preserve the logical / physical volume correspondence.

4.2.1 Investigation Results

The investigation of this area is complete. There is a mixed version issue with this change. The present XQP implementation may behave extremely badly (including corrupting the volume or crashing) when handed a volume whose file structure is smaller than the physical volume size. Thus we must ensure that older versions of VMS never mount such a disk. The current version of MOUNT has no checks that would detect this condition. The recommend solution is to tie this feature to ODS-5. This requires that at minimum the XQP, VERIFY, and SALVAGE work ship with RAVEN.

The following sections describe work required to address the problem.

4.2.2 Design Considerations

The ODS-2 (and ODS-5) file structure already has the logical volume size recorded in the storage control block (SCB$L_VOLSIZE). This field can be used as needed by VMS components that need to reference the volume size.

4.2.2.1 The XQP

The XQP references UCB$L_MAXBLOCK in two places:

1. The storage allocator, to limit the length of bitmap searches
2. Map virtual to logical, to prevent I/O off the end of the volume.

Case (1) can be changed to use VCB$W_SBMAPSIZE. We must make sure that unused blocks at the end of the storage bitmap (which may be present due to cluster round-up) contain zeroes. Alternatively, a correct logical EOF on the bitmap would do the job.

There are no dire consequences to not changing case (2), since the primary issue is to avoid sending a driver a bad disk address in case the file structure is corrupted.

One might consider a separate VCB field to hold the logical volume size. This does not appear to be necessary.

4.2.2.2 MOUNT

MOUNT references UCB$L_MAXBLOCK in the storage bitmap length check. This computation must be changed to use the logical volume size from the storage control block.

There is another reference to UCB$L_MAXBLOCK in MOUNT, to compute the volume blocking factor (the number of sectors per logical block). This computation does not need to be modified.

4.2.2.3 ANALYZE/DISK

Like the file system, ANALYZE/DISK assumes that the logical volume size is equal to the physical volume size. This assumption pervades its storage bitmap handling code, and careful inspection will be necessary to make sure that the assumption is correctly removed. Fortunately, there are relatively few direct references to UCB$L_MAXBLOCK; most storage map size references use an internal copy that can be redefined.

At the same time, ANALYZE/DISK also carefully enforces that the storage map is large enough to cover the physical volume, and checks that the volume geometry parameters in the storage control block match those of the physical volume. This enforcement must be relaxed for ODS-5 volumes. If the storage control block on an ODS-5 volume is to be rebuilt, VERIFY may only obtain the geometry parameters from the physical volume if the storage map covers the entire volume. Otherwise, it must maintain the existing short storage map and fabricate a reasonable set of geometry parameters.

4.2.2.4 SALVAGE

SALVAGE is capable of creating a new storage map on an arbitrarily damaged volume. It makes sense to automatically size the new storage map to the physical volume size. However, SALVAGE also has logic for reusing the existing storage map if one is found. If the existing storage map does not cover the volume, the user should be queried as to whether SALVAGE should build a new storage map. If the requests to keep the existing storage map, SALVAGE must limit the logical volume size in all other operations to the space covered by the storage map.

4.2.2.5 BACKUP

Once this feature is in place, BACKUP can be modified to allow physical copies between dissimilar size volumes. BACKUP currently has a check in place to ensure that the size of the source and target volumes is identical. This check remains essential to file organizations that cannot accommodate smaller logical volumes. Therefore, BACKUP's volume size check would have to become file structure dependent. For file structures that allow smaller logical volumes (i.e., ODS-5), BACKUP's check can be relaxed to ensuring that the size of the target volume is at least as large as the logical volume size.

4.2.2.6 Host Based Volume Shadowing

Host based volume shadowing enforces the requirement that all members of a shadow set be of identical size. (At present, it also enforces them to be of identical geometry. However, this check is obsolete with the geometry-independent home block search and could already be removed.) For volumes capable of smaller logical volumes (i.e., ODS-5), this check can be relaxed to a check ensuring that the physical size of all shadow set members be at least as large as the logical volume size.

The check is made in the ADDSHAD FDT routine of the shadow driver. More investigation is needed to work out how to make the logical volume size accessible to this code. (The driver does read and update the storage control block, but not until much later in the ADDSHAD process.)

4.2.3 Status

The XQP, MOUNT, and ANALYZE/DISK portions of this work have been completed and will ship in Raven (VMS V7.2). A bitmap validation change to BACKUP was required as well. The short bitmap logic has been implemented so that it can function for both ODS-2 and ODS-5 disks. So far there are no ODS-5 checks, because there is as yet no logic that will actually cause disks with a short bitmap to exist. Because of the prior version compatibility issues, we still recommend that short bitmaps be allowed to appear on ODS-5 volumes only.

4.3 Dynamic Volume Growth

Many storage aggregation products (e.g., snap capable disk and other logical volume managers) have the interesting potential of allowing a logical storage volume to grow in size. The same thing happens when larger disks are used to replace smaller disks in a VMS shadow set. A function to grow the logical volume size should be added to the XQP to take advantage of this characteristic. Note that logical vs physical volume separation (section 3.1.2) is a prerequisite for this feature.

5. BACKUP - IMPROVED BUFFER HANDLING

There have been innumerable complaints from VMS installations that the existing VMS BACKUP is too slow. Much of this, of course, is simply that we are up against the laws of physics in moving the volume of data dictated by the current backup architecture. However, there is also considerable evidence that BACKUP does not run at its full performance potential.

We know that in the common disk to tape save mode, BACKUP uses a large buffer pool to read file data optimized to minimize disk seek time. However, disk and tape I/O are not run concurrently. After filling the buffer pool and initiating the tape writes, BACKUP waits for all tape I/O to complete before starting the next disk read pass. Aside from the lost I/O concurrency, starting and stopping a streaming tape drive incurs a disproportionate performance penalty. The end result is that backups to a streaming tape often run at significantly less than even the available disk bandwidth.

We need two activities to remedy this problem:

1. A performance study of BACKUP to understand its present performance behavior and determine the potential performance that might be realized within the existing design.
2. Changes to the buffering and I/O algorithms to allow concurrent disk and tape I/O, at least in circumstances where this will improve performance.

5.1 Investigation Results

BACKUP uses a large buffer pool to optimize head motion when backing up small files. It opens a large number of files concurrently, reading the file attributes and extent maps. The files' data blocks are allocated to buffer space in logical file order. When the entire buffer pool has been allocated, BACKUP sorts the file extents by LBN and then issues concurrent read requests in LBN order. It then waits for the reads to complete in file order. As the reads for each buffer complete, the buffer is written asynchronously to tape. Once all the buffers have been written, BACKUP waits for all tape writes to complete, and then starts the process over.

There are a couple of concurrency issues with this design:

- First and foremost, there are no tape writes in progress while BACKUP is opening the next set of files. This can be a lengthy process, easily long enough to drain the cache on any controller.
- The disk read process is vulnerable to resource problems. If BACKUP runs out of ASTLM or DIOLM while posting the reads, it stalls until previously issued reads complete. No tape writes are issued until all reads have been issued, so resource stalls are particularly insidious.
- On non-TMSCP tapes, only two writes may be outstanding at any time. This is because only TMSCP drives support the serious exception mechanism, and BACKUP must be prepared to respond promptly to exception conditions such as EOT. On non-TMSCP tapes, BACKUP

waits for previously issued writes to complete to limit the outstanding number of writes. In the existing design, this is harmless because the only concurrent activity in the tape write phase is waiting for disk reads to complete. However, it becomes an issue when we make other changes to improve disk/tape concurrency.

5.2 Solution

We solved BACKUP's concurrency problems by the simple technique of splitting the buffer pool in half. Thus, the process of opening, sorting, and reading files operates on one half of the buffer pool while the other half is being written to tape. In addition, we addressed the limit on two tape I/Os by modifying the stall mechanism to run as an AST thread. That is, when two tape writes are in progress, no further writes are issued, but the buffer is marked ready to write. In the tape write completion AST, we issue the next write if a buffer is ready.

Splitting the buffer pool also helps the resource starvation problem considerably, because even though disk reads become serialized, we still have tape writes concurrently in progress in the other half of the buffer pool.

5.3 Results

A cursory test on a fragmented disk showed a performance improvement of 30% to 40%. The resulting throughput was within a few percent of the net bandwidth available from the disk (as measured by backing up the disk to the null device), showing that we had achieved close to the maximum possible I/O overlap. In a test of backing up a single large file from an RZ29B to a TZ89, we were able to sustain a data rate of 6.5MB/sec, which is close to the spiral bandwidth of the disk.

5.4 Status

This work has been completed and will ship in Raven (VMS V7.2).

6. SNAP CAPABLE DISK BASED BACKUP

Snap-capable disk is a facility that provide inexpensive virtual copies of a volume. Virtual volumes are mapped into a storage pool of real disks. A snapshot is simply another virtual volume that, on creation, shares the mapping of its parent volume. As the parent continues to be updated, the mapping is changed so that the snapshot retains the original data. Snapshots, as implemented by the Snapshot Services for OpenVMS, have a number of important properties:

- The segment is the unit of mapping granularity, and is on the order of 65KB in size.
- A snapshot can be taken with a simple metadata update requiring a few seconds.
- The real disk space taken by the snapshot is simply the amount by which the snapshot differs from its parent or other snapshots.
- The initial write to any segment after a snapshot has been taken is slow, because the original data must be copied to a new location and the mapping of the snapshot updated. This operation, known as a "copy out" is four to five times as slow as an ordinary write.
- Reads and subsequent writes (i.e., writes to segments that have already been copied out) are as fast as an ordinary disk.
- Snapshot Services for OpenVMS supports up to 12 concurrent snapshots of a volume.

Snap-capable disk provide an opportunity to "break the laws of physics" as applies to backup. There are two levels of development worth considering:

1. Backups of snapshots
2. Snapshots as backups

6.1 Backups of Snapshots

In simple terms, taking a snapshot solves the problem of the shrinking backup window. The system manager simply takes a snapshot of the volume to be backed up, and then makes a backup of the snapshot by conventional means. After the backup is complete the snapshot can be deleted. The backup window thus expands to the entire time interval between backups. This style of BACKUP becomes available with OpenVMS Snapshot Services V1.0.

This approach minimizes the performance impact of the snapshot because the snapshot only exists while the backup is being run. It fits well with installations that have demanding uptime requirements, but have periods of reduced application load that accommodate the snapshot overhead and the bandwidth required by the backup.

The principal disadvantage of this approach is that it does not solve the bandwidth vs capacity dilemma. The same amount of data must still be copied from storage onto backup media. Furthermore, we must keep in mind that behind the virtual disks that represent the primary volume

and its snapshot, there is still a common set of real media and disk heads. Bandwidth taken by the backup is not available to the application. Thus a full backup remains a painful proposition.

6.2 Snapshots as Backups

This section provides a conceptual overview of the snapshot based backup being proposed for VMS. Section 6.6 contains a formal analysis of the material discussed here.

A more radical approach to backups is to simply view the snapshot as the backup. In its crudest form, this simply means taking periodic snapshots (e.g., at the same frequency that one would do incremental backups) and keeping them forever. To recover a file, one simply copies the file from a snapshot containing a correct copy of the file back to the parent volume. Note that such a recovery can be done by any user without system manager or operator involvement, provided the snapshots are kept mounted as public volumes.

This brute force approach suffers from several fatal practical limitations:

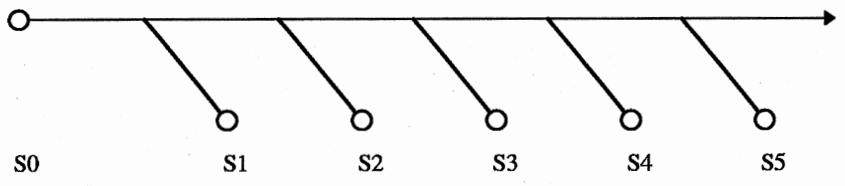
- As much as we would like to sell disks, most installations are unwilling to continuously add real disk storage to maintain copies of data that existed in the past.
- Snapshot Services for OpenVMS supports only 12 concurrent snapshots of a volume. While this limit could be increased, any snapshot product will have some finite limit on the number of snapshots than can be retained online.
- Online snapshots provide for individual file recovery, but do not address the system manager's worst nightmare: loss of an entire volume, or worse, loss of the entire storage subsystem. Techniques such as shadowing or RAID5 effectively address media failures, but do not address failures at a higher level or of a larger scope. Disasters are only addressed with a complete offline copy of the entire storage subsystem.

These limitations dictate that it must be possible to migrate snapshots to offline storage and back. We define a new style of backup:

6.2.1 Physical incremental backup

A snapshot may be taken offline by copying out the data necessary to recreate the snapshot from other data that is still online, such as either the parent volume, or some other snapshot. The data is precisely the set of data in which the snapshot differs from the data retained online. Consider the following view of a logical volume as a series of snapshots:

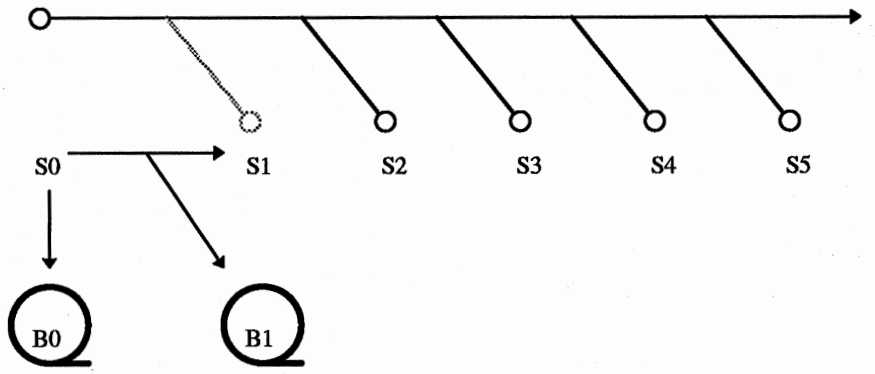
Figure 1 - Snapshot Time Series



S0 represents a snapshot of the base state of the volume; the following snapshots are taken as time goes by. A file that existed at the time, say, S3 was taken but has since been deleted or corrupted can be recovered simply by copying it from the logical volume represented by S3.

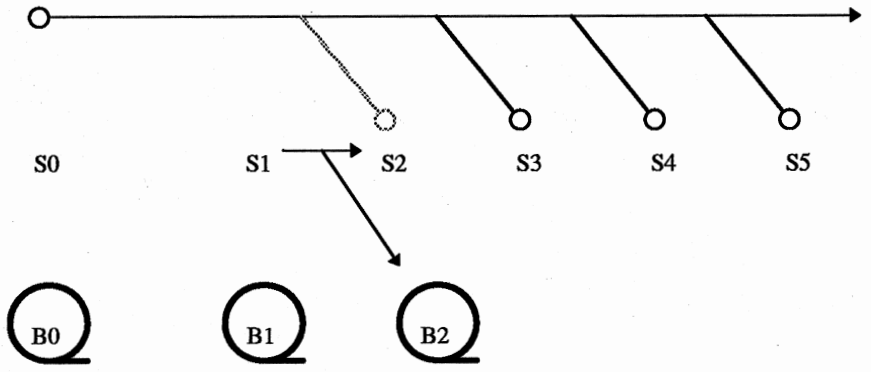
We make offline backups of this series of snapshots as follows:

Figure 2 - Base and Incremental Backup



We make a full copy of S0 to capture the initial state of the volume. Then we make the backup B1 by copying only those segments of S1 that differ from S0. We discover which segments of S1 are different by examining the snap disk metadata. Once S1 has been copied, it may be partially deleted (shown in the illustration by drawing S1 in gray). Partial deletion means that we can release blocks in the storage pool containing data unique to S1. However, we must retain the mapping metadata for S1, because it is still needed in the next step. Note it is not necessary for S1 to be deleted. It can be retained online, and perhaps deleted later.

Figure 3 - Further Incremental Backup



As the next step, we make a backup of snapshot S2 in the same way, by copying only those segments that differ from S1. At this point, as the illustration shows, S2 may be partially deleted and S1 may now be fully deleted. Incremental backups continue in this manner.

To recover the volume, should the storage subsystem fail, we simply restore the backups in order, starting with the full backup B0 and overwriting it with B1 and B2, etc.

An important characteristic of physical incremental backups is that they copy only those segments that have changed, rather than entire files. This addresses a long-standing problem with file-based incremental backups of files that are databases (not just true databases but other update in place files such as RMS indexed files, etc.) With a file-based incremental backup, changing just one record in a file causes the entire file to be backed up. With a physical incremental, only the segments containing the updated record, and possibly affected index blocks, are backed up.

6.2.2 Offline Consolidation

It is not practical to accumulate incremental backups forever. The storage volume grows indefinitely, as does the number of tapes that must be applied to accomplish a full restore. Our field experience has shows that system managers are generally unwilling to accumulate more than a week's worth of incremental backups.

Offline consolidation addresses this problem. Offline consolidation tools are available for file-structured backups (the StorageWorks Save Set Manager). However, consolidation of file-structured save sets is a tricky business because of the complex semantics of files; the end result does not always match the equivalent full backup of the volume. Logical volumes, on the other hand, are brute simple. A logical volume is simply a set of blocks; each block may be read and written, nothing else. Consequently, consolidating physical incremental save sets is a very simple matter with guaranteed results: Consolidating the base backup B0 from the preceding illustrations with subsequent incrementals B1 and B2 yields the exact equivalent of a full backup taken of S2.

Making full backups is as much an emotional issue for system managers as it is a practical one. The system manager needs to be confident that they have an easily restorable, reliable copy of their data. System managers have always done periodic full backups because there was no other way to fill this need. Offline consolidation, plus a verification function that ensures that the resulting consolidated save set is correct, fills the need and eliminates the requirement of periodic full backups. The consolidated save set B2 described above can be verified simply by comparing its contents with snapshot S2 while it is still online.

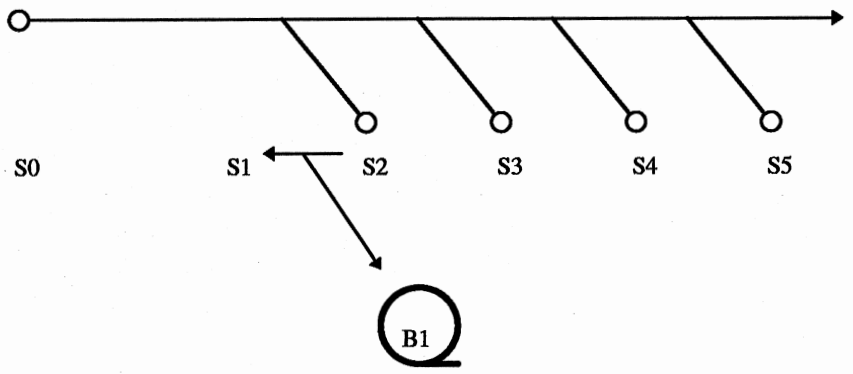
An important aspect of offline consolidation is that it consumes no bandwidth in the primary data store. It requires additional equipment in the form of tape drives and processing power. However, high end customers with stringent availability requirements are willing to make the necessary investment. Physical incremental backups with offline consolidation are the technique that "breaks the laws of physics" by eliminating the periodic full copy of the data store.

6.2.3 Incremental Restore

The regime of forward incremental backups works well for disaster recovery, but is not appealing for individual file recovery. The ideal way of recovering a file is, of course, to simply copy it from a snapshot that is still online. However, if snapshots have been rolled out and deleted as described previously, restoring a snapshot requires effectively recovering the entire volume by restoring S0 and all subsequent snapshots up to the one desired. This amounts to a full physical restore of the volume, and is not acceptable.

It is much more desirable to restore an older snapshot by effectively rolling back from a snapshot that is still online. To do this, we need what amounts to a reverse incremental backup:

Figure 4 - Reverse Incremental Backup



We accomplish a reverse incremental backup of S1 by copying out all segments of S1 that are different from its successor S2. The operation is identical to the forward incremental backup described previously, except that we make the compare in the opposite direction. Unlike a forward incremental, however, once the copy is complete S1 can be deleted completely, since the next reverse incremental will involve S2 and S3.

Having made a reverse incremental backup of S1, we can restore S1 (assuming S2 is still online) by making a clone of S2 and then overwriting it with the contents of B1. The incremental storage and time required to restore S2 are directly proportional to the amount of change between the two snapshots. More generally, any snapshot can be restored using a series of reverse incrementals by cloning the oldest (or nearest) available online snapshot and then restoring the necessary chain of backups.

6.2.4 Composite Backups

In practice, system managers need to be prepared for both disaster recovery and individual file restore. It therefore makes sense to have a single set of backups that serve both purposes. We accomplish this by simply merging the two functions. Thus, for example, the backup of S1 consists of the union of the differences between S1 and S0, and S1 and S2.

Incremental backups, both forward and reverse, can also be consolidated to reduce recovery time of old snapshots and to reduce offline storage requirements. A consolidated incremental backup turns out to be just another form of composite backup because it is the union of differences among multiple snapshots. In the general case, a composite backup contains the contents of some snapshot that differ from some set of other snapshots. From this point of view, the ordering of the snapshots is unimportant. The composite backup can be used to roll forward or back from any snapshot that appears in the differences list.

One of the complications posed by composite backups is that they contain data that is redundant, from the point of view of any particular restore. For example, when the composite B1 described in this section is used to roll S2 back to S1, some set of blocks that are different between S1 and S0 are identical between S1 and S2, and are therefore not needed for the rollback. If we write these

anyway in an incremental restore of S1, we will consume unnecessary space in the snap disk pool. This issue can be addressed in two ways:

1. maintaining a complete set of difference maps with the backups, so that we know what blocks must be written for any particular restore, or
2. performing a differential restore, by comparing the save set data to the snapshot clone being written and only writing the blocks that are different.

6.3 Product Function

Snapshot backup is implemented as a set of variants on the existing BACKUP/PHYSICAL function. Command and qualifier names are given as proposals and need to go through the usual DCL review process.

The following descriptions make reference to a snapshot unique identifier. The form and semantics of the unique identifier are discussed in section 6.5.2.1.1.

6.3.1 Full Backup

BACKUP/PHYSICAL device-name save-set-name

The full physical backup functions as it always has. However, in addition, if the device specified is a snapshot, the save set is tagged with the unique identifier of the snapshot.

There is a major optimization of full backups that we should consider. That is copying out only the allocated blocks on a volume. In a regime that uses snap-based backup on a volume for its entire lifetime, the initial full backup would be taken immediately after the volume has been initialized. The number of blocks allocated at this time is very small, and this optimization would reduce a multi-gigabyte data copy to a trivial operation. This variant can be achieved with another modifier on the physical backup:

BACKUP/PHYSICAL/ALLOCATED device-name save-set-name

The allocated blocks physical backup determines what blocks on the volume are currently in use by reading the volume's storage bitmap. The resulting save set is a sparse physical save set, containing only the allocated blocks. Like a normal physical save set, it is tagged with the unique identifier of the snapshot. Note that this function, unlike all other aspects of snap-based backup, requires knowledge of the volume's file structure.

Another approach to optimizing the initial backup of a volume is to take a snapshot before initialization, and then take the initial backup as a physical incremental with respect to the pre-initialization snapshot. This has the advantage of not requiring knowledge of the file structure. However, the initial backup is a physical incremental. Restoring it would require a mechanism to bypass the usual coherency checks of physical incremental restores.

6.3.2 Physical Incremental Backup

BACKUP/PHYSICAL device-name save-set-name /DIFFERENCE=(device-name, device-name, ...)

This command produces a new type of save set, a physical incremental save set. The save set contains the contents of all segments of the device being saved that are different from their corresponding segments in the devices in the differences list. The save set is tagged with the unique identifier of the snapshot being saved and all snapshots in the differences list.

The device being backed up and all devices listed in the differences list must be snapshot virtual disks. As in a full physical backup, the device being saved must be mounted; however the devices in the differences list need not be mounted. (This is because the differences devices may be partially deleted snapshots. Mounting these is not possible.)

6.3.3 Full Physical Restore

BACKUP/PHYSICAL save-set-name device-name

The operation functions the same way a physical restore has always worked. The named device is overwritten with the contents of the save set. If the save set is a sparse physical save, then where blocks are missing in the save set the blocks on the destination disk are left unwritten. In addition, if the device is a snap disk root volume or snapshot, its unique identifier is set to the unique identifier stored in the save set. (If the save set has no unique identifier, such as a save set written by an older version of BACKUP, then the virtual volume is given a new identifier.)

The device being restored to must be mounted /FOREIGN.

6.3.4 Incremental Physical Restore

BACKUP/PHYSICAL save-set-name device-name

The named device is overwritten with the contents of the save set where the save set contents differ from the current device contents. (This is done either by using the differences map contained in the same set or by comparing the save set data to the device before writing - see section 6.2.4.) Note that this function has the same syntax as a full physical restore. We tell the difference by observing that the save set is a physical incremental save set. The device being restored to must:

1. Be mounted /FOREIGN
2. Be a snapshot root volume or virtual device
3. Have a unique identifier that appears in the differences list in the save set.

6.3.5 Save Set Consolidation

CONSOLIDATE save-set-spec1, save-set-spec2, ... output-save-set-spec

The named save sets are consolidated, so that the output save set contains the sum total of the input save sets, overwritten left to right. That is, save-set-spec1 is overwritten by save-set-spec 2, and is subsequently overwritten by save-set-spec3, etc.

For each save set in the input list, save set **n** must contain in its differences list the unique identifier of save set **n-1**. The unique identifier of the output save set is set to the unique identifier of the last input save set in the list.

If all save sets are physical incrementals, the output save set is also a physical incremental save set, and its differences list is the union of the differences lists of all the input save sets. If any input save set is a full physical backup, the output save set is a full physical backup.

Specifying a full physical save as anything other than the first save set spec is not very useful and should probably be flagged as an error. A sparse physical save is treated the same as a full physical save, except that the output is also a sparse physical save containing the union of all blocks encountered in the input save sets.

6.3.6 Backup Operation

Performing backups consists of two sets of actions: taking a snapshot and then rolling the snapshot out to a save set. These actions need not happen at the same time.

6.3.6.1 Taking a snapshot

1. Create a snapshot of the root volume using OpenVMS Snapshot Services.
2. Mount the snapshot privately, write-enabled. Allow any file system rebuild or recovery to complete.
3. Change the volume label of the snapshot to a suitably descriptive name. (The primary requirement is that the label be unique cluster-wide.)
4. Dismount the snapshot volume.
5. Using OpenVMS Snapshot Services, write-lock the snapshot.
6. Mount the snapshot cluster-wide (write-locked).

6.3.6.2 Roll out to save set - full backup

1. Copy out the snapshot with the command

   BACKUP/PHYSICAL/ALLOCATED device-name save-set-name
2. The snapshot may now be partially deleted (data contents released).
3. Any preceding snapshots may now be fully deleted.

6.3.6.3 Roll out to save set - incremental backup

1. Identify the device names of the immediately preceding and succeeding snapshots as **prev-device** and **next-device**.
2. Copy out the snapshot with the command

   BACKUP/PHYSICAL device-name save-set-name /DIFFERENCE=(prev-device,next-device)
3. The snapshot may now be partially deleted (data contents released).
4. Any preceding snapshots may now be fully deleted.

6.3.7 Restore Operation

6.3.7.1 Full volume restore

1. Identify the collection of save sets needed to restore the volume. This should begin with a full save, followed by the complete chain of incremental saves, such that each incremental save contains in its differences list the unique identifier of the preceding save.
2. Create a new logical volume of the same size as the one being restored
3. Restore the save sets in forward order, each with the command

   BACKUP/PHYSICAL save-set-name device-name
4. Write lock the logical volume if desired.

5. Mount the volume cluster-wide.

The restore process could be optimized by restoring the save sets in reverse order and maintaining a cumulative map of segments written, so that each segment of the volume being restored is written only once. This would require a single backup command that accepted the entire list of save sets to be applied.

6.3.7.2 Incremental snapshot restore

1. Identify an online snapshot that is "near" the snapshot to be restore. Other things being equal, nearness usually correlates with proximity in time.
2. Identify the collection of save sets needed to restore the snapshot. This is a chain of incremental saves, starting with the save that includes the snapshot identified in (1) in its differences list, and ending with the save of desired snapshot. Each incremental save must contain in its differences list the unique identifier of the preceding save. The chain of save sets may proceed either forwards or backwards in time, depending on the time ordering of the identified and desired snapshots.
3. Create a snapshot of the snapshot identified in (1).
4. Restore each incremental save identified in (2) in order onto the newly created snapshot, using the command

   BACKUP/PHYSICAL save-set-name device-name
5. Write lock the new snapshot.
6. Mount the snapshot cluster-wide.

6.4 Requirements on OpenVMS Snapshot Services

Snapshot backup requires a number of features in OpenVMS Snapshot Services beyond the capabilities available in V1.0:

6.4.1 More than 12 concurrent snapshots

An installation that uses snapshots for backups as described will want to have more than 12 snapshots concurrently available. 64 feels like a good number. There is nothing inherent in the snapshot services architecture that prevents supporting a larger number. However, the vote field in the mapping metadata will have to be increased from a word to a quadword. Suitable testing to ensure that the implementation scales is also in order.

6.4.2 Difference map between two snapshots

The mapping metadata is private to snapshot services. To preserve software modularity, snapshot services must implement an API that queries the metadata to return the differences list between snapshots.

6.4.3 Last write time on snapshot

When we make an incremental restore of a snapshot, it is absolutely critical that the incremental save being restored match up with the snapshot being restored onto. That is, the snapshot being restored onto must appear in the differences list of the incremental save. While restoring an incorrectly matched incremental file save at least results in complete recognizable files, incorrectly

restoring a physical incremental results in an arbitrarily corrupted disk. In effect, we need a unique identifier on a snapshot that also tracks any updates that might be made to the snapshot. OpenVMS Snapshot Services provides a 16 byte unique identifier on each logical volume (both snapshot and root volume). To this we must add a last write time. This may be the time of dismount after being mounted write enabled.

6.4.4 Writeable unique identifier

The entire unique identifier of a snapshot (both the existing 16 byte UID and last write time) must be writeable, so that we can restore a snapshot to its exact prior state.

6.4.5 Cloning a snapshot

As section 6.2.3 describes, we perform an incremental restore by overwriting a snapshot with the differences between it and some other snapshot. In most cases, we want to retain the snapshot being overwritten as well, so what we should really overwrite is a clone of the snapshot.

The functional specification for OpenVMS Snapshot Services V1.0 implies that this capability exists, but this needs to be verified.

6.4.6 Partial snapshot deletion

As section 6.2.1 describes, once a snapshot has been copied out with an incremental backup, we can release its associated data segments. However, we must retain its segment map entries so that they can be used to determine which segments of some other snapshot are different when that other snapshot is backed up.

6.4.7 Write-locking a snapshot

As section 6.9 explains, correctly matching up the endpoints of an incremental save with a snapshot being overwritten is critical to preventing arbitrary volume corruption. To preserve the usefulness of physical incremental saves, the snapshots against which they are taken must not change subsequently. The best way to enforce this is to write-lock a snapshot at the snapshot services layer, so that it cannot subsequently be accidentally written. (Note that simply mounting a snapshot write-enabled updates its modification date and invalidates its usefulness as a base for an incremental restore.)

6.5 Design Overview

This section presents some highlights of the design. It is not a complete design specification.

6.5.1 Snap Disk Differences Map

The snap disk differences map is a data structure that appears in a couple of places. Its purpose, obviously, is to list which segments of a pair of snapshots are different. There are two obvious choices for expressing lists of disk blocks or segments: an extent map or a bitmap. The extent map has the advantage of being more compact if there is significant locality of the blocks described. To date, we have no evidence that there will be significant locality in the differences between snapshots. In fact, the metadata in the snap disk implementation uses per segment map pointers for differing segments, implying that locality is not expected.

As a result, I recommend that the differences map be expressed as a simple bitmap of segments. The size of the bitmap (in bits) is simply the size of the logical volume divided by the segment size.

6.5.2 BACKUP Save Set Format

The BACKUP save set format is sufficiently extensible that it comfortably accommodates the physical incremental backup. We define a number of new attributes and new record types.

6.5.2.1 New attributes

6.5.2.1.1 Snapshot unique identifier

The snapshot unique identifier indicates which snapshot is associated with the data contained in the save set. The format of this identifier consists of the 16 byte snap disk UID, plus the 8 byte time of last write of the snapshot.

6.5.2.1.2 Difference snapshot unique identifier

The difference snapshot identifier is the identifier of some other snapshot. It identifies a difference snapshot that was used to select the contents of this save set. Multiple difference snapshot identifiers may be present, indicating a composite physical incremental backup. The set of difference snapshot identifiers constitutes the "differences list" of a physical incremental save referred to in sections 6.2 and 6.3.

6.5.2.1.3 Snap disk segment size

Simply the size of a segment of the snap capable disk. This data is necessary to interpret the differences map.

6.5.2.1.4 Difference map

A bitmap indicating which segments of some snapshot are different from the corresponding segments of some other snapshot. It indicates which segments are present in a physical incremental save set.

Multiple difference maps may be present, indicating a composite physical incremental backup. Each difference map must be preceded by a difference snapshot identifier.

Because of record length limitations, the differences map must usually be segmented. Multiple difference map attributes that appear in successive records without intervening snapshot unique identifiers are to be concatenated into a single map.

6.5.2.2 Extensions to existing record types

6.5.2.2.1 Physvol summary record

The physvol summary record identifies a physical backup. For compatibility reasons, we continue to use this record for the full backup of a snapshot. When a physical backup is made of a snapshot, the following additional attributes are present in this record:

- Snapshot unique identifier
- Segment size

6.5.2.3 New record types

6.5.2.3.1 Physical incremental summary record

The physical incremental summary record identifies a physical incremental backup. It contains all the attributes contained in the physvol summary record, plus the following:

- Difference snapshot unique identifier
- Difference map

Multiple pairs of difference snapshot identifier and difference map may be present in a composite backup; each difference map associates with the preceding identifier.

6.5.2.3.2 Physical incremental summary extension record

Because of record length limitations, it is likely that the difference map will not fit into a single summary record. As many extension records may be present to contain the necessary data. Their contents are interpreted by simply concatenating all physical incremental summary extension records to the initial physical incremental summary record.

6.5.2.3.3 LBN difference record

This record is an alternative design to putting the full set of difference maps in the physical incremental summary record. Instead, we distribute the map data throughout the data records. The LBN difference record is identical to an LBN record, except that its record header contains a small bitmap. The bitmap indicates which difference snapshots the contents of the record are associated with.

Using the LBN difference record design, the physical incremental summary record contains only a list of difference snapshot identifiers. Treating the list of difference snapshot identifiers as an array, setting bit n in the LBN difference record header indicates that the contents differ from snapshot identifier n. This design is analogous to the design of the vote field in the snap disk map pointer.

Using this design reduces memory requirements in the physical incremental restore and in save set consolidation, because it is not necessary to keep the entire difference map in memory. However, it makes it impossible to know in advance the total space required by the restore.

6.5.2.4 New save set type

We have described a new type of save set, the physical incremental save. It consists of the following sequence of records. There are two alternative designs.

6.5.2.4.1 Differences map in summary record

- BACKUP summary record
- Physical incremental summary record, containing list of difference snapshot identifiers and difference maps
- Physical incremental summary extension records as necessary
- LBN records. All LBNs identified in the union of difference maps must be present.

6.5.2.4.2 Differences map in data records

- BACKUP summary record

- Physical incremental summary record, containing list of difference snapshot only
- Physical incremental summary extension records as necessary (not needed unless the differences list is very large)
- LBN difference records

6.5.3 BACKUP Operation

The enhanced physical backup and physical incremental backup functions are integrated into BACKUP's existing physical backup code. Much of their construction is self-evident from the preceding functional and structural descriptions. We observe that to create a physical incremental save, BACKUP uses an API to OpenVMS Snapshot Services to fetch the difference map (described in section 6.5.1). Further investigation is needed to establish whether we can get away with the brute force approach of fetching the entire map at once, or whether the API needs to support fetching the map piecemeal to save memory. We observe that for a 1TB volume with a 65KB segment size, the size of the difference map is 2MB.

6.5.4 Snapshot Difference Extraction

The existing snap disk metadata works as follows:

- A root map describes, with a short sequence of extents, the mapping of the root volume to blocks in the storage pool.
- Additional map pointers describe the exception cases where individual segments in snapshots have been remapped to other blocks in the storage pool to preserve their contents as the root volume changes. Each map pointer contains
    - The segment number being remapped
    - The pool address to which the segment is mapped
    - A vote field, consisting of a bitmap of snapshots to which this map pointer applies. (Each snapshot is assigned a unique bit in the vote field.)

Thus determining the difference map between a pair of snapshots consists of scanning all snap disk map pointers and recording the segment numbers of all map pointers that have either, but not both, vote bits set for the two snapshots in question.

6.5.5 Partial Snapshot Deletion

The physical incremental backup uses the snap disk metadata for a new purpose: determining the difference between a pair of snapshots. To optimize the use of the storage pool, it is useful to differentiate the two purposes in the metadata structure. Section 6.2.1 describes how, in a scenario of rolling successive snapshots to offline storage, the contents of a snapshot are no longer needed, but its mapping data is still necessary to determine the difference between it and the next snapshot. We release data segments unique to that snapshot but retain the mapping data with a partial deletion.

Full deletion of a snapshot consists of clearing its vote bit in all map pointers, and deleting all map pointers with a resulting zero vote field and freeing their associated pool space. The vote bit associated with the deleted snapshot may then be assigned to a new snapshot.

Partial deletion of a snapshot consists of

1. Marking the snapshot as partially deleted in the snapshot summary description
2. Freeing the pool space associated with all map pointers whose vote field lists only partially deleted snapshots, and writing a null pool address in all such pointers.

The map pointers and the vote bit associated with the partially deleted snapshot remain in use.

6.6 Opportunities with ABS

ABS (Archive Backup System) is a software product built by the Storage Systems software group. It provides a set of value added functions layered on top of VMS BACKUP:

- Automatic backup scheduling
- Remote management
- Remote save set I/O
- File catalog
- Save set tracking
- Media management

ABS is built primarily around the concept of file backup. The physical incremental backup is purely a volume backup, and so affects ABS in a couple of ways.

6.6.1 Full Volume vs File Backup

Tracking individual files in the physical and physical incremental backups may not be practical. The file names are not directly available from the backup operation the way they are from a file-oriented backup. Each backup is a full volume backup. ABS could still track individual files by extracting and storing a directory listing of the snapshot. This may prove to be excessively costly for two reasons:

1. The time required to take a directory listing of the entire volume is likely to be long compared to the time required for the physical incremental backup itself.
2. The directory listing will list every file, whether it has been modified or not. Without careful optimization, the resulting catalog would be much larger than it currently is.

We need input from the ABS engineering organization and ABS customers as to whether individual file cataloging is needed for the physical incremental backup environment.

6.6.2 Snapshot and Save Set Tracking

Sections 6.2 and 6.3 describe the mechanics and requirements of physical incremental backups. Incorporating these procedures and mechanisms would be a natural extension to ABS. Thus, ABS could provide the following functions to automate physical incremental backups:

- Automatic scheduling of snapshots
- Automatic rollout of snapshots
- Automatic deletion of rolled out snapshots based on time and space policy
- Tracking and consolidation of physical incremental backups

- Restore of snapshots, automatically identifying a suitable base for the restore and the necessary save sets
- If a file catalog is maintained, identification of a suitable snapshot from which to restore a file.

6.7 Opportunities with Windows NT

OSSG is developing products to enhance Windows NT's readiness for mission critical computing applications. To this end, we are porting selected VMS technologies to Windows NT, including snap capable disk. Bringing snap disk based backup to Windows NT is a natural consequence.

Networker is a distributed client/server backup product sold by Legato Systems. Its capabilities are comparable to those of ABS. Networker is available on many operating systems, including Windows 95, Windows NT, MacOS, and many brands of Unix. In addition, a Networker client is available for VMS. Engineering for Networker on Digital platforms is done by a storage software group in DECwest.

Because of Networker's presence as a multi-platform product and our existing business relationship and in-house expertise, it makes sense to build snap disk backup on Windows NT as a Networker component. Discussions with the DECwest group suggest that the preferred implementation method is to use existing source components of Networker to build a new Networker client that does the physical and physical incremental backup functions. Additional modifications to the Networker management components are needed to integrate the new functions.

We are not planning a Windows NT product for the immediate future because of resource constraints. However, components built for the VMS product should be designed with an eye toward portability to Windows NT.

6.8 Project Outline

6.8.1 Platform and Packaging

Snap based backup must ship as a bundled component of VMS because it is built into the existing BACKUP utility. However, its function is enabled by installing OpenVMS Snapshot Services. Since OpenVMS Snapshot Services is only available on Alpha, snap based backup is likewise an Alpha only product.

6.8.2 Software Components

The preceding functional and design descriptions make the development components of the project pretty much self evident.

6.8.2.1 BACKUP

Integration of the physical incremental save and restore functions into the existing physical save and restore components of BACKUP.

6.8.2.2 Save set consolidator

A new utility. It should be possible to reuse much of the save set handling logic from BACKUP.

6.8.2.3 OpenVMS Snapshot Services

Implementation of the requirements described in section 6.4.

6.8.2.4 Example command files

The procedures for doing backups with snapshots described in section 6.3.6 are just complex enough that it is worth building a set of sample command procedures that implement the functions.

6.8.3 Resources

At a SWAG level this looks like a three person project: one engineer each for BACKUP, the consolidator, and snap disk, plus one of these acting as project leader. Total design and development time feels like 9 months for a 3 person team. QTV resources look like ¼ engineer over 3 months.

Hardware resources are not particularly demanding, and require a VMS workstation for each engineer, plus a medium size server with a half dozen high performance disks and a high performance tape drive for performance testing.

6.9 Theoretical Analysis

The physical incremental backup and restore concepts associated with snap-based backup are sufficiently different from conventional backup techniques that they demand a certain theoretical analysis. By defining a set notation to describe the backup and restore operations, we can prove that the design actually works. Most of the following discussion consists of definitions. Once the concepts and operations of snap-based backup are rigorously defined, its operation follows almost trivially.

We begin by defining a volume **V** as a set of **N** data segments, thus

$V \equiv \{C(n), \forall\, n \in \{1 \ldots N\}\}$

where
n is a segment number
C(n) is the contents of segment n

Snap capable disks allows us to take periodic snapshots of a volume, yielding a series of volumes that are called a *family*. Typically, the snapshots in a family are a linear time series. However, this property is not exploited in this discussion. We extend the definition of a volume to include the identifier for snapshot **Si**:

$V(Si) \equiv \{C(n,Si), \forall\, n \in \{1 \ldots N\}\}$

A physical incremental backup involves copying the difference between two snapshots; thus we define a difference function to yield the set of segment numbers that differ between the two snapshots.

$\Delta(Sj,Si) \equiv \{n \mid C(n,Sj) \neq C(n,Si), \forall\, n \in \{1 \ldots N\}\}$

Thus, a physical incremental backup **B** represents the difference between two snapshots:

$B(Sj,Si) \equiv \{C(n,Sj), \forall\, n \in \Delta(Sj,Si)\}\}$

We refer to the snapshot being copied, **Sj**, as the backup's *content* and the snapshot against which we take the difference, **Si**, as the *base*. Note that a backup is semantically similar to a volume. Both are sets of data segments; they differ in the parameters that determine their constituency.

It is useful to generalize the notion of a physical incremental backup to cover the difference between one snapshot and multiple other snapshots:

$$B(Sk,\{Si,Sj,\ldots\}) \equiv B(Sk,Si) \cup B(Sk,Sj) \cup \ldots$$

or

$$B(Sk,\{Si,Sj,\ldots\}) \equiv \{C(n,Sk), \forall\, n \in \{\Delta(Sk,Si) \cup \Delta(Sk,Sj) \cup \ldots\}\}$$

As an aside, we observe that if a series of snapshots are strictly time-ordered, one can make some simplifying assumptions. A series of snapshots is strictly time-ordered if each successive snapshot consists of updates to the previous snapshot, that is,

$$\text{for } i < j < k,\ \Delta(Sj,Si) \subseteq \Delta(Sk,Si)$$

It follows that for a set of time-ordered snapshots, the composite backup over the set is equal to the simple backup over the endpoints:

$$\text{for } i < j < k,\ B(Sk,\{Si,Sj\}) = B(Sk,Si)$$

In practice, we expect snapshots to be mostly, but not completely, time-ordered. A snapshot is likely to receive minor modifications before it is backed up. For example, to mount a snapshot of a volume system-wide in VMS, its volume label must be changed. As a result, the remainder of this analysis does not assume time ordering and copes with the modest additional complexity.

To take advantage of the semantic equivalence between backups and volumes and to avoid a lot of clutter in the following discussion, we define a shorthand notation for the set of segment numbers contained in backup of volume **B**, **N(B)**.

$$N(B) \equiv \{n\} \mid B = \{C(n), n \in N(B)\}$$

A volume or backup may be overwritten by another volume or backup. We denote this with the operator $\multimap$. Thus, "A overwrites B" is written:

$$A(S_A) \multimap B(S_B) \equiv \{C(n,S_A)\ \forall\, n \in N(A)\} \cup \{C(n,S_B)\ \forall\, n \in N(B) \mid n \notin N(A)\}$$

Observe that

$$N(A(S_A) \multimap B(S_B)) = N(A) \cup N(B)$$

Finally, we define an operation called *conditional overwrite*. Conditional overwriting achieves the same results as overwriting; however, it includes an additional test for data equality. If the overwriting data is equal to the data being overwritten, we use the latter rather than the former. In terms of content, the results are the same as a simple overwrite. However, in the practical case of restoring a physical incremental backup to a snapshot, it avoids unnecessary data copies.

The operator $\not\multimap$ is used to denote conditional overwriting. Thus, "A conditionally overwrites B" is written:

$$A(S_A) \not\multimap B(S_B) \equiv \{C(n,S_A)\ \forall\, n \in N(A) \mid n \notin N(B) \vee C(n,S_A) \neq C(n,S_B)\}$$
$$\cup \{C(n,S_B)\ \forall\, n \in N(B) \mid n \notin N(A) \vee C(n,S_A) = C(n,S_B)\}$$

With these definitions we can describe the physical incremental backup and restore process in formal notation. Assume we have a family of successive snapshots of a volume, V(Si), V(Sj),

V(Sk), etc. We now take a physical incremental backup of snapshot Sj with respect to snapshot Si (that is, all segments that have changed from Si to Sj). Thus we have

$B(Sj,Si) \equiv \{C(n,Sj), \forall\, n \in \Delta(Sj,Si)\}\}$

or

$B(Sj,Si) \equiv \{C(n,Sj), \forall\, n \mid C(n,Si) \neq C(n,Sj), \forall\, n \in \{1...N\}\}$

Snapshot **Sj** is now deleted. Some time later we wish to restore **Sj** and **Si** still exists. **Sj** can be restored by overwriting **Si** with the backup of **Sj**. (Note we presumably overwrite a copy of **Si**, rather than the original.)

$V(Sj') = B(Sj,Si) \nleftrightarrow V(Si)$

$= \{C(n,Sj)\ \forall\, n \in \Delta(Sj,Si)\} \cup \{C(n,Si)\ \forall\, n \in N(V) \mid n \notin \Delta(Sj,Si)\}$

$= \{C(n,Sj)\ \forall\, n \in N(V) \mid C(n,Sj) \neq C(n,Si)\} \cup \{C(n,Si)\ \forall\, n \in N(V) \mid C(n,Sj) = C(n,Si)\}$

By substituting **C(n,Sj)** for **C(n,Si)** where they are equal, the equation collapses:

$V(Sj') = \{C(n,Sj)\ \forall\, n \in N(V) \mid C(n,Sj) \neq C(n,Si)\} \cup \{C(n,Sj)\ \forall\, n \in N(V) \mid C(n,Sj) = C(n,Si)\}$

$= \{C(n,Sj)\ \forall\, n \in N(V)\}$

$= V(Sj)$

Multiple successive backups may be consolidated, by overwriting one with the other. Thus:

$B(Sk,Sj) \nleftrightarrow B(Sj,Si) = \{C(n,Sk)\ \forall\, n \in \Delta(Sk,Sj)\} \cup \{C(n,Sj)\ \forall\, n \in \Delta(Sj,Si) \mid n \notin \Delta(Sk,Sj)\}$

$= \{C(n,Sk)\ \forall\, n \in N(V) \mid C(n,Sk) \neq C(n,Sj)\}$

$\cup \{C(n,Sj)\ \forall\, n \in N(V) \mid C(n,Sj) \neq C(n,Si) \wedge C(n,Sk) = C(n,Sj)\}$

Taking advantage of the final term **C(n,Sk) = C(n,Sj)** above, we can substitute **Sk** for **Sj**:

$= \{C(n,Sk)\ \forall\, n \in N(V) \mid C(n,Sk) \neq C(n,Sj)\}$

$\cup \{C(n,Sk)\ \forall\, n \in N(V) \mid C(n,Sk) \neq C(n,Si)\}$

$= \{C(n,Sk)\ \forall\, n \in N(V) \mid C(n,Sk) \neq C(n,Sj) \vee C(n,Sk) \neq C(n,Si)\}$

$= \{C(n,Sk), \forall\, n \in \{\Delta(Sk,Sj) \cup \Delta(Sk,Si)\}\}$

$= B(Sk,\{Sj,Sj\})$

More generally, composite backups may be consolidated as long as the overwriting backup includes as a base the content of the backup being overwritten, that is,

$B(Sm,\{Sl,Sk,Sj,Si,...\}) = B(Sm,\{Sl,Sk,...\}) \nleftrightarrow B(Sk,\{Sj,Si,...\})$ for some Sk

The proof is based on substituting, as above, **C(n,Sm)** for **C(n,Sk)** in the cases where they are equal, and is the same as the above proof of simple consolidation, but with considerably more clutter.

Next, we observe that a composite backup may be used to restore a snapshot as long as the snapshot used as a base for the restore appears in the list of bases of the backup. That is, we can restore **Sk** onto **Sj** using **B(Sk,{Sj,Si})**:

$V(Sk') = B(Sk,\{Sj,Si\}) \nleftrightarrow V(Sj)$

We simply observe from the definition of a composite backup above that

$B(Sk,\{Sj,Si\}) \supseteq B(Sk,Sj)$

Thus, some segments of **V(Sj)** where **C(n,Sk) = C(n,Sj)** will be unnecessarily written, but substituting **B(Sk,Sj)** for **B(Sk,{Sj,Si})** reduces the operation to

$V(Sk') = B(Sk,Sj) \oplus V(Sj)$

proven above. It is because of the redundant writes that we introduce the notion of the conditional overwrite, so that an incremental restore is done as

$V(Sk') = B(Sk,\{Sj,Si\}) \oplus_{\Delta} V(Sj)$

We prove this in a manner similar to the derivation for simple overwriting above. Note the simplification of the conditional overwrite expression, because $\Delta(Sj,Si) \subseteq N(V)$.

$$V(Sk') = B(Sk,\{Sj,Si\}) \oplus_{\Delta} V(Sj)$$

$$= \{C(n,Sk)\ \forall\ n \in (\Delta(Sk,Sj) \cup \Delta(Sk,Si)) \mid C(n,Sk) \neq C(n,Sj)\}$$

$$\cup \{C(n,Sj)\ \forall\ n \in N(V) \mid n \notin (\Delta(Sk,Sj) \cup \Delta(Sk,Si)) \vee C(n,Sk) = C(n,Sj)\}$$

$$= \{C(n,Sk)\ \forall\ n \in N(V) \mid (C(n,Sk) \neq C(n,Sj) \vee C(n,Sk) \neq C(n,Si)) \wedge C(n,Sk) \neq C(n,Sj)\}$$

$$\cup \{C(n,Sj)\ \forall\ n \in N(V) \mid C(n,Sk) = C(n,Sj)\}$$

Simplifying the above expression discards the redundant portion of **B(Sk,{Sj,Si})**.

$$= \{C(n,Sk)\ \forall\ n \in N(V) \mid C(n,Sk) \neq C(n,Sj)\}$$

$$\cup \{C(n,Sj)\ \forall\ n \in N(V) \mid C(n,Sk) = C(n,Sj)\}$$

By substituting C(n,Sk) for C(n,Sj) where they are equal, the equation collapses:

$$V(Sk') = \{C(n,Sk)\ \forall\ n \in N(V) \mid C(n,Sk) \neq C(n,Sj\}$$

$$\cup \{C(n,Sk)\ \forall\ n \in N(V) \mid C(n,Sk) = C(n,Sj)\}$$

$$= \{C(n,Sk)\ \forall\ n \in N(V)\}$$

$$= V(Sk)$$

The analysis of the physical incremental backup and restore assumes that the various snapshots **V(Si)** and backups **B(Si,{Sj...})** are immutable objects. Backups are traditionally treated as read-only entities, so for them it is not an issue. A snapshot, however, is simply another volume that can be mounted, read, and, written. Because a physical incremental backup is totally divorced from the file structure, any incorrectness in a restore will have unpredictable consequences from the file system's point of view. For this reason, enforcing that snapshots, once backed up, do not change, is critical.

7. STORAGE MANAGEMENT ARCHITECTURE

Later. Much later.

8. ISSUES

8.1 Snap Disk Performance

The success of snap-disk based backup is critically dependent on the acceptance of snap capable disk as a mainstream storage technology. The principal issue that affects the adoption of snap capable disk is its performance. We have primitive level performance data available for snap capable disk. "Normal" disk I/O, that is, direct reads and writes, are essentially unaffected. Each application I/O translates into a single real I/O, with an increase in CPU overhead of less than 5%.

However, the first write of any segment after a snapshot has been taken requires a "copy-out", in which the current data is copied to a different location so that the primary mapping can be retained for the root volume being written. A copy-out write takes about 4 to 5 times as long as an ordinary write because of the additional data and metadata I/O.

What is not yet well understood is the real life impact of copy-out performance. We have some data points:

- The engineering group in LKG that is porting snap disk to Windows NT used a snap disk (with a snapshot active) as a build result disk. The net performance of the build was 30% slower.
- We have acquired a number of I/O trace logs from Ken Bates in Storage Engineering. These are complete, unadulterated logs of physical I/O activity taken with a SCSI bus analyzer. They represent the disk activity of a variety of different workloads. We have done statistical analysis and simulation of a snapshot on a number of these traces. The results vary greatly.

So far, the best conclusion we can draw on the system-wide performance of snap-capable disk is the classic "it depends greatly on the workload".

8.2 Snap Disk Design Center

The snap disk engineering folks have expressed some concerns that the snap disk backup design takes snap capable disk well beyond what they intended. The backup model described in the snap disk design documents is the "backup of snapshot" approach discussed in section 3.3.1. The consequences of such an approach are that much of the time, the installation runs with no snapshots present. Backups are done in an off-peak period when the performance impact of the snapshot and the concurrent backup can be tolerated. In general, the installation uses a very small number of snapshots of any volume.

The snapshots as backups approach, on the other hand, assumes that each volume will have a considerable number of snapshots active at any time. The performance issue described in section 8.1 aside, there remain open questions over whether the snap disk implementation will scale to handle larger numbers of snapshots successfully.

8.3 ...