Point-in-Time Copy: Yesterday, Today and Tomorrow

Alain Azagury, Michael E. Factor and Julian Satran
IBM Research Lab in Haifa
MATAM, Haifa 31905, Israel
{azagury, factor, satran}@il.ibm.com
Phone: +972 4 829-6211, Fax: +972 4 829-6116
William Micka
IBM Storage Systems Group
9000 S RITA ROAD, Tucson, AZ, USA
micka@us.ibm.com
Phone: +1 520 799-4132

1. Introduction

Making copies of large sets of data is a common activity. These copies can provide a consistent image for a backup, a checkpoint for restoring the state of an application, a source for data mining, real data to test a new version of an application, and so on. One characteristic all of these uses have in common is that it is important that the copy appear to occur atomically, *i.e.,* any updates to the data source being copied either occur before or after the copy. In this work, we examine the history, the state-of-the art, and possible future of mechanisms for copying large quantities of data via storage subsystem facilities for providing *point-in-time* (PiT) copies.

The Storage Networking Industry Association (SNIA) defines a *point-in-time copy* as:

> A fully usable copy of a defined collection of data that contains an image of the data as it appeared at a single point-in-time. The copy is considered to have logically occurred at that point-in-time, but implementations may perform part or all of the copy at other times (e.g., via database log replay or rollback) as long as the result is a consistent copy of the data as it appeared at that point-in-time. Implementations may restrict point-in-time copies to be read-only or may permit subsequent writes to the copy. Three important classes of point-in-time copies are split mirror, changed block, and concurrent. Pointer remapping and copy on write are implementation techniques often used for the latter two classes. *cf.* snapshot [1]

As hinted at by the above definition a range of point-in-time copy facilities exist. Some of these facilities operate at the logical level of the file system [2][3] and some operate at the physical level of the disk storage subsystem [2][4][5][6]. We focus on copy facilities provided by disk storage subsystems.

Before the invention of point-in-time copy facilities, to create a consistent copy of the data, the application had to be stopped while the data was physically copied. For large data sets, this could easily involve a stoppage of several hours; this overhead meant that there were practical limits on making copies. Today's point-in-time copy facilities allow a copy to be created with almost no impact on the application; in other words, other than perhaps a very brief period of seconds or minutes while the copy is established, the application can continue running.

This paper describes the functionality of a point-in-time copy facility and describes both the benefits and drawbacks of providing this facility on the storage subsystem. While there are other benefits, the biggest benefit of providing this facility on the storage subsystem is performance; we do not needlessly add load to the storage network or host as part of making the copy. The biggest drawback is that the storage subsystem in today's world is only aware of data at the level of logical units and blocks;[i] this makes it hard to meaningfully perform copies at a granularity of less than an entire logical unit.

After defining the concept of point-in-time copies, we briefly survey several existing approaches including EMC's TimeFinder[4], IBM [7] and StorageTek's [8] virtual array solutions, and several file system based approaches. Although the focus of this paper is on point-in-time copy solutions for block controllers, we also describe file system snapshots, in particular Network Appliance's snapshot feature [9]. We then describe the FlashCopy facility of IBM's Enterprise Storage Subsystem (ESS) [6] which was developed in our labs; we present performance results showing that this facility allows copying arbitrary amounts of data in almost zero time.

We then describe one possible future for point-in-time copy facilities. We see two main future thrusts for point-in-time copy facilities. This first is improved performance; while today's facilities can make a copy in almost zero time, even this is sometimes too much time. The second is a melting of the division between the organized logical view of data implemented by a file system and the physical view as seen by today's disk subsystems [10]. In particular we believe that the arrival of object based storage *e.g.,* [11], will provide a critical enabler for allowing a disk subsystem to provide a physical point-in-time copy of logically meaningful data.

The rest of this paper is organized as follows. The next section provides a background on point-in-time copy facilities, describing the different approaches to implementing these facilities and the tradeoffs between point-in-time copies at the file system level and at the storage subsystem level. Section 3 describes several existing facilities for point-in-time copy and Section 4 describes the FlashCopy facility of IBM's ESS, showing how this facility allows copying almost arbitrary amounts of data in nearly zero time. In Section 5, we describe one possible course of development for point-in-time copy solutions prior to concluding.

2. Background

As we stated in the introduction, a point-in-time copy may be made for many reasons. While backup is probably the most common reason, checkpointing, data mining, testing

and other reasons also exist. In all cases, prior to making the copy, applications accessing the data must purge any caches they have; many middleware applications, such as databases, provide mechanisms to ensure that the underlying storage subsystem or file system has a consistent copy of the data without stopping the application. In addition, for copy facilities provided by the storage subsystem, the file system must ensure that it has written a consistent image of the data to the storage subsystem.

Obviously if the data to be copied involves multiple entities, *e.g.,* multiple logical units or multiple file systems, this quiescing of the application must occur atomically for all of the entities. Only after all of the copies have been made, is the application again allowed to modify the underlying data. This means that application access to the data is limited for the duration of time it takes to execute the copy.

Prior to the development of point-in-time copy solutions, the only way to make a copy of a data set was to allocate space and physically copy the to data. To ensure consistency, the application was not allowed to access the data while the copy was being executed. Since the time required to execute a physical copy is a function of the size of the data, this could easily lead to an application being unavailable for an extended period of time. This time overhead, as well as the need to fully allocate the space required for the target, limited the use of copies; one would not copy an entire volume of data every hour for purposes of checkpointing an applications state.

In none of today's popular facilities, however, does the point-in-time copy command execute a physical copy of the data. Instead the data is either copied prior to the execution of the command or some form of copy-on-write like facility is used. Not only does this reduce system overhead, but also it enables copies to be used in the range of applications listed above.

As stated above, there are different classes of implementations of point-in-time copy. In a *split mirror* a mirror of the data is constructed prior to the point-in-time copy. After a complete mirror of the data to be copied exists, the point-in-time copy is made by "splitting" the mirror at the instance in time of the copy. The biggest benefit of a split mirror solution is that the point-in-time copy executes very quickly; there is no work required in order to create tables or mark data as copy on write. On the other hand, split mirror suffers from a significant drawback in terms of advanced planning. One cannot create a split mirror at any time one wants; rather, it is necessary to plan ahead and create the mirror in advance of splitting. Since the mirror requires a complete physical copy of the data, the set up for creating a split mirror must begin significantly prior to the actual point-in-time copy. A second drawback of a split mirror solution since it is based on physical mirror copy is that it inherently requires that the space allocated for the target of the copy be equal to the space used by the source. Finally, the overall storage system performance is affected by the requirement to continuously mirror the changes until the administrator decides to split the mirror.

One variant of a split mirror solution allows the mirror to be resynchronized with the source. When this is done, only the records of the source, which have changed since the mirror was split, are copied to the source. This allows a true mirror to be created much

faster than if the entire data set needed to be physically copied to the mirror. This variant does, however, require work to create data structures to track which records in the source have been modified.

A second class of implementations is *changed block*. A changed block implementation shares the physical copy of the data between the source and the target until the data is written; this sharing can be at the level of a sector, a track, or conceivably some other granularity (we refer to this unit as a *record* below). To allow the data to be shared some form of table is used to determine where the actual copy of the data exists. When the source and target are accessed this table is used to determine from where the data is to be retrieved.

This table can be the directory that exists in virtual arrays such as log structured arrays [12] or it can be some other mechanism that is used only for purposes of supporting a point-in-time copy, such as a copy-on-write bitmap that tells whether or not a given record has been copied. A changed block approach is easy to implement on completely virtual systems, or other mechanisms, which use indirection for all accesses; however, it is also possible to implement a changed block approach in more conventional systems.

When the data is written a changed block implementation will either manipulate pointers in a directory or copy the written data. In either case, after the update the source and the target no longer share a physical copy of the given record.

A changed block implementation requires setting up the table to keep track of what records have been copied when the point-in-time copy is made; this obviously takes time that is linear in the size of the data to be copied. However, since these tables can be no more than a copy-on-write bitmap, this can be done very efficiently. One big benefit of changed block implementations over split mirror implementations is that no advanced set up is required prior to executing a point-in-time copy. Another feature of a changed block implementation is that the amount of space required is a function only of the amount of data modified.

A *concurrent* point-in-time copy is similar to a changed block implementation with one significant difference. A concurrent implementation always physically copies the data. Like a changed block solution, however, when the point-in-time copy is executed, no data is physically copied. Instead, the concurrent solution sets up a table to keep track of which data has been physically copied. It then physically copies the data in the background, using the table to synchronously copy any records that are about to be modified.

One other axis on which point-in-time copy solutions can be differentiated is whether or not the target of the copy is a first class citizen, *i.e.,* can the target be freely accessed or are there limitations on the way it is used, *e.g.,* no updates, only sequential reads, etc.

As discussed in the introduction point-in-time copies can be made either at the file system level or at the storage subsystem level. The biggest benefit of performing the copy at the storage subsystem level is that it can reduce the load on the server and on the

storage network (assuming one is being used). When the copy is made at the level of the file system, all of the computation of the copy must be made on the file server; in addition, whenever physical copies are required, the data must be transferred up through the storage subsystem, over the storage network to the server and then back down the same path. If the copy is made by the storage subsystem, we can totally avoid the overhead on the storage network and on the host.

3. Point-in-Time Copy Today

Research on storage point-in-time copy techniques is extremely scarce. Since one of the major uses of point-in-time copy is as a building block for efficient backup, the literature on backup techniques covers partially this topic [13]. In this section, we review some of the major point-in-time solutions available in the market. In addition, while we focus on disk storage subsystems, we describe two point-in-time copy techniques at the level of the file system.

3.1 Split Mirror Solutions

EMC's TimeFinder [2][4] and Hitachi's ShadowImage [5] are two examples of *split mirror* implementations. We describe TimeFinder's major characteristics. TimeFinder allows creating mirror images of standard devices. These mirrored images, called Business Continuance Volumes (BCVs), may be later *split* and accessed independently. BCV images are created in the background and several copies of a standard device may be created. BCVs can go through the following stages:

- *Establish* – a new BCV device is established and, initially, contains no data.
- *Isynch* – the point-in-time where the BCV pair is synchronized with the standard device.
- *Split* – makes the BCV volume available to the host. The content of the BCV volume is a point-in-time copy of the standard device at the time the split command was issued.
- *Re-establish* – The volume is re-assigned as a mirror of the standard device. The BCV volume is refreshed with any updates made to the standard device, and any updates to the BCV after the split are discarded.
- *Restore* – Copies the contents of the BCV back to the standard device.
- *Incremental restore* – Discards all the changes made to the standard device since the split occurred and copies updates made to the BCV device to the standard device.

The latest version of TimeFinder [14] introduces *changed block* capabilities: a new *instant split* operation allows BCVs to become immediately available to the hosts. This is achieved by copying tracks before they are modified in the standard device.

3.2 Log Structured Changed Block Solutions

IBM's RAMAC Virtual Array (RVA) [15][7] and StorageTek's Shared Virtual Array [8] are major examples of changed block solutions that leverage the log structure data structures for their point-in-time copy implementation. IBM's RVA represents a volume using a set of tables that eventually point to the set of tracks that comprise the volume. RVA also maintains a reference count for each track containing physical data. A snapshot operation from a source to a target volume requires (1) decreasing the reference count of the target tracks, (2) copying the "track" table from the source to the target and (3) increasing the reference count of the source volume tracks. RVA's snapshot is both efficient in time – requiring only to copy the track table of the source and updating the track reference counts – and efficient in space – since no copy of the user data is required.

3.3 File System Solutions

Many UNIX-like file systems have leveraged their *inode*, pointer-based data structures to implement efficient snapshot capabilities. The Andrew File System [3] implements a Clone operation that creates a frozen copy-on-write snapshot. Snapshots are read-only and are traditionally used for backup purposes, to allow backing up a consistent point-in-time snapshot, with minimal disruption of the activity on the production file system. In addition, snapshots can be used for easy restore of deleted or corrupted files.

Network Appliance's filer [2] also implements a copy-on-write-based snapshot facility [9] that creates on-line, read-only copies of the entire file system. It currently allows administrator to create up to twenty snapshots of a file system. In order to support snapshots, the free block data structure is extended to mark to which snapshots the block belongs. A block might be returned to the "free pool" only after each bit, for each snapshot is zero. Snapshot are created under the "`~snapshot`" directory. Users may retrieve files from previous snapshots, and restore them using standard file system "copy" commands.

Network Appliance has integrated its snapshot features with a SnapMirror/SnapRestore capability. SnapMirror allows automated, consistent replication of file systems to remote sites. It creates periodically a snapshot of the file system and then transfers the modified blocks to the remote site. After a baseline transfer is complete, Snapmirror leverages the snapshot bitmaps to identify which blocks need to be transferred to the remote site. SnapRestore allows restoring a mirrored snapshot to the primary.

File system snapshots are very efficient operations, since they only require keeping copies of modified or deleted files. However, since, not only the data, but also the metadata is read-only, one cannot modify metadata attributes of files in snapshots. For example, revoking access to a file from a user does not prevent him from accessing (earlier versions of) the file in previous snapshots. In addition, when a copy is required, the data must be transferred from the storage subsystem to the file system and back to the storage subsystem.

4. ESS's FlashCopy Today

FlashCopy is an ESS Copy Services function, developed in our labs, which is a *concurrent* class *point-in-time* copy operation. It utilizes *copy-on-write* bitmap techniques to maintain knowledge of which blocks of data have been a modified after the *point-in-time* copy was created. Real storage equal in size to the source data is required on the target volume. When a block of data on the source volume is modified, the previous version of that data is copied to the target volume before the new modification overwrites it. An optional background copy task may be initiated to perform the physical copy of the entire source volume to the target volume.

FlashCopy, unlike a split mirror technique, provides instant availability for read and write data on both the source and target volumes as soon as the invocation of the operation is complete. It utilizes the ESS cache and fast write functionality to mask any performance affects related to the *point-in-time* copy which may be activated for a given volume. FlashCopy operates on volumes for zSeries hosts and for volumes attached to open systems hosts. When invoked from a zSeries, the host program can specify that only a portion of the volume be copied. This is called a *sparse volume*. If portions of the volume are not allocated or are catalogs or volume table of contents, these can be excluded from the copy to the target and managed by the host software. An open systems volume is copied in its entirety.

The most important performance metric related to the creation of the *point-in-time* copy is the elapsed time required for the invocation of the copy on one pair or multiple pairs of volumes. During invocation, the application must maintain a consistent image of the data across all volumes used for the application. The amount of time required can be considered an application impact and must be minimized by the design of the copy function.

When a FlashCopy is initiated, the source and target are entered into a relationship using a bitmap table which reflects the location of the point-in-time data - either on the source volume or on the target volume. While the relationship table is being created within the ESS control unit, the two volumes are made unavailable to all customer access. The time for this operation can vary considerably with the method of FlashCopy initiation. The zSeries program DFSMSdss [15] performs various steps prior to the relationship creation period which elongates the initiation. DFSMSdss must read the Volume Table of Contents (VTOC), perform RACF security verifications, and then reserve the volumes involved for data integrity purposes. Given this task overhead, the FlashCopy initiation can take approximately 6 seconds for a 3 gigabyte volume. By contrast, the TSO FlashCopy function and the ESS Specialist Command Line Invocation does not include reading or verification steps and can take less than 2 seconds for the same relationship. By performing the invocation for many volumes in parallel, the time to complete the set of relationships is much better than the summation of individual invocations.

# of Flash Copies	dss small VTOC	dss large VTOC	TSO invoked
1	6 sec	8 sec	1.2 sec
256	48 sec	66 sec	18 sec

As can be seen from the table, the invocation time is a function of the number of volumes in the total data collection and the amount of information on the volumes as reflected in the VTOC.

Another important performance measurement is the effect on application response time and the number of I/O operations that can be executed per second while a FlashCopy relationship exists for a volume pair or number of volume pairs. Measurements were made using 256 FlashCopy pairs while running a cache standard workload which show less than 3% reduction in the I/O rate when the workload volumes are in a relationship with the *no background copy* option selected. With the *background copy* option selected, the rate reduction is about 7%.

The change to the workload response time is negligible when the *no background copy* is specified. There is negligible response time increase when the *background copy* is specified for 32 volumes or less in one control unit. With 256 volumes in *background copy* mode, the response time rises doubles until the number of background copy tasks is reduced by completing the copy for a pair of volumes.

5. The Future of Point-in-Time Copy

The world of data copies has improved significantly since the invention of the first facilities that allowed a logical copy without requiring a physical copy. However, there is still room for improvement. In the small, the improvements include improving the performance of today's solutions to reduce even further the impact on the application for creating a copy. In addition, it should be possible to provide greater flexibility in the facilities provided by storage subsystems, allowing a greater degree of knowledge of the logical structure of the data to flow down to the physical layer.

In the large, the greatest improvement may come from new storage architectures such as object based storage. With an object based storage and the appropriate file system architecture, it should be possible to completely bridge the gap between the logical structure as seen by the file system and the physical structure provided by the storage subsystem.

5.1 Improving Today's Point-in-time Copy

As fast as the execution of a point-in-time copy may be, until it is instantaneous, it will never be fast enough. This is because as described in Section 2, while the command for the point-in-time copy is executing, it may be necessary to limit application access to the data being copied.

There are several aspects to improving the performance of today's point-in-time copy solutions. First, it is important to speed up the time required to ensure that the component performing the copy has a copy of the data that is consistent with the application's view of the data. This includes ensuring that all data that is in cache has been written through to the appropriate level of the system or at the very least knowing what data needs to be retrieved from a cache.

Second, the data structures used to manage the copy need to be set up quickly. To some degree this is a problem that is inherently linear in the size of the data to be "copied"; for instance, a table recording which data has been copied must be a size that is the same order of magnitude as the size of the data. However, even here, by intelligently preparing the data structures it may be possible to hide some or most of the overhead from the application.

In addition to improving performance, we believe that point-in-time copy solutions will evolve to have more flexibility in terms of the allowing knowledge of a file system's logical structure to flow down to the storage subsystem. To a degree this exists today for mainframe data with the support for making point-in-time copies of individual data sets [6][16]. However, more work is required to provide this same facility for partitions in a way that is not tied to a particular logical volume manager or file system.

5.2 Point-in-time Copy and Object Based Storage

Object Based Storage (*e.g.*, [11]) provides the client (storage consumer) with a storage abstraction closer to the client's view of the data than the conventional device view. In conventional storage devices the client perceives a device as a collection of storage blocks (usually fixed length). Organizing this primitive storage into entities significant to applications and managing all storage resources is delegated to client software (operating system), sometimes in conjunction with a third party (a file server). Only through client and/or file server software is the client able to attach significance to data. This classical structure has two main disadvantages:

- it is hard to scale to large volumes of data and large throughput since data servers quickly become bottlenecks

- data management at storage level has no relation to content

Widely discussed in academia and now starting to happen in industrial laboratories, a new form of storage access - Object Based Storage - changes the way storage is accessed and managed.

Object Based Storage (OBS) relegates space management to the storage subsystem. Instead of perceiving a volume as an amorphous collection of equally sized storage blocks the storage client perceives now a volume as a collection of variable length (possibly sparsely populated) objects and the mapping of those objects to device-blocks is delegated to the storage controller.

Client access to data is based on an object-handle (capability) established by a management component in the network. Access to data is protected through the capability and unmediated.

In addition to enable building highly scalable storage subsystems (as the access does not have to go through a data server) Object Based Storage make access units (objects) "visible" and manageable at storage subsystem level. The previously discussed copy functions can now be performed based on policies pertinent to specific objects or classes of objects.

In addition since the storage subsystem has complete control over device block location information and internal object structure, management functions, such as point-in-time copy or incremental point-in-time copy, can be made with minimal space (and time) overhead and encompass any set of objects (not necessarily a volume or a large portion of a volume). And although the examples that follow involve files it can easily be observed that they might as well refer to database tables or any other type of storage object.

5.2.1 Point-in-time copy for a set of files

Point-in-time copy for a set of files in a file-system built using Object Based Storage involves the following steps on a client/administrative system:

1. Lock the set of files
2. Copy the directory entries for the set of files
3. Request a point-in-time copy for the set of objects containing the files data from the storage subsystem to be performed asynchronously
4. Release the locks
5. Wait for the point-in-time copy command to end

The storage subsystem will do the following:

1. Mark all the involved objects (their control structures) as copy-on-write
2. Return to the host an indication of "successful request"
3. Perform the request while accepting read/write operations from the host

It is easy to observe that given enough free space to accommodate host write operations during the point-in-time copy generation, any number of point-in-time operations can be performed simultaneously.

To perform such a point-in-time copy for a set of files using today's mechanisms, would require that we give the control unit detailed knowledge of the way the file system lays

out files. Since on disk layout differs between file systems, separate implementations would be required for each file system supported.

6. Conclusions

We have described the current state of the art of point-in-time copy operations, focusing on the FlashCopy facility of IBM's ESS developed in our labs. Using FlashCopy as an example, we have shown how today's point-in-time copy facilities can perform a semantic copy of large quantities of data in essentially zero time.

While performance of today's copy is orders of magnitude superior to the time required to make a physical copy, there is still some room to improve performance. More significantly we see that the future melding of block based and file based storage, promised by facilities such as object based storage, will provide an opportunity for storage subsystems to provide point-in-time copy for entities that are meaningful to the end user, *e.g.,* files, and not just entire or large portions of logical units.

Acknowledgements

FlashCopy would not exist today were it not for the diligent work of a large development team led by Yoram Novick along with the support of the entire ESS development team.

References

[1] *A Dictionary of Storage Networking Terminology*, http://www.snia.org/English/Resources/Dictionary_FS.html

[2] Hutchinson, N., Manley, S., Federwisch, M., Harris, G., Hitz, D., Kleiman, S., O'Malley, S., "Logical vs. Physical File System Backup" *Third Symposium on Operating Systems Design and Implementation.*1999.

[3] Howard, J., Kazar, M., Menees, S., Nichols, D., Satyanarayanan, M., Sidebotham, R., West, M., "Scale and Performance in a Distributed File System", *ACM Transactions on Computer Systems*, Vol. 6, No. 1, February 1988, Pages 51-81.

[4] EMC TimeFinder Product Description Guide, 1998, EMC Corporation, http://www.emc.com/products/product_pdfs/pdg/timefinder_pdg.pdf

[5] Hitachi ShadowImage, June 2001, http://www.hds.com/pdf/shadowimageR6.pdf

[6] Mellish, B., Blazek, V., Beyer, A., and Wolatka, R. *Implementing ESS Copy Services on UNIX and Windows NT/2000*. Feb. 2001, IBM.

[7] McAuley, D., Pate, A., Black, I., Bueffel, V., Chana, B., Docherty, G., Leplaideur, D., Nel, W., *IBM RAMAC Virtual Array,* July 1997, IBM

[8] "StorageTek™ SnapShot Software"
http://www.storagetek.com/products/software/snapshot/

[9] Brown, K., Katcher, J., Walters, R., Watson, A., *SnapMirror and SnapRestore: Advances in Snapshot Technology,*
http://www.netapp.com/tech_library/3043.html, Network Apppliance, Inc.

[10] Gibson, G.A., R. Van Meter, "Network Attached Storage Architecture," *Communications of the ACM,* Vol. 43, No 11, Nov., 2000

[11] "Object Based Storage Devices: A Command Set Proposal,"
http://www.nsic.org/nasd/final.pdf Nov. 1999

[12] Menon, Jai. "A performance comparison of RAID-5 and log-structured arrays" Proceedings of the Fourth IEEE International Symposium on High Performance Distributed Computing, 1995. p.167-178.

[13] Chervenak, A., Vellanki, V., Kurmas, Z., "Protecting File Systems: A Survey of Backup Techniques", Proceedings *Joint NASA and IEEE Mass Storage Conference,* March 1998.

[14] EMC TimeFinder, 2000, EMC Corporation,
http://www.emc.com/products/product_pdfs/ds/timefinder_l700-4.pdf

[15] Pate, A., Vaia, C., Todd, J., and Aigner, H. *Implementing DSMSdss SnapShot and Virtual Concurrent Copy,* June 1998, IBM

[16] Blunden, M., Bergum, S., Dovidauskas J., and Vaia, C. *Implementing ESS Copy Services on S/390,* Dec. 2000, IBM.

[i] Or tracks and volumes for zSeries; we focus on open systems.