Design of VMS Volume Shadowing Phase II-Host-based Shadowing

By Scott H. Davis

Abstract

VMS Volume Shadowing Phase II is a fully distributed, clusterwide data availability product designed to replace the obsolete controller-based shadowing implementation. Phase II is intended to service current and future generations of storage architectures. In these architectures, there is no intelligent, multiunit controller that functions as a centralized gateway to the multiple drives in the shadow set. The new software makes many additional topologies suitable for shadowing, including DSSI drives, DSA drives, and shadowing across VMS MSCP servers. This last configuration allows shadow set members to be separated by any supported cluster interconnect, including FDDI. All essential shadowing functions are performed within the VMS operating system. New MSCP controllers and drives can optionally implement a set

Overview

Volume shadowing is a technique that provides data availability to computer systems by protecting against data loss from media deterioration, communication path failures, and controller or device failures. The process of volume shadowing entails maintaining multiple copies of the same data on two or more physical volumes. Up to three physical devices are bound together by the volume shadowing software and present a virtual device to the system. This device is referred to as a shadow set or a virtual unit. The volume shadowing software replicates data across the physical devices. All shadowing mechanisms are hidden from the users of the system, i.e., applications access the virtual unit as if it were a standard, physical disk. Figure 1 shows a VMS of shadowing performance assists, which Digital intends to support in a future release of the shadowing product.

Volume Shadowing Phase II set for a Digital Storage Systems Interconnect (DSSI) configuration of two VAX host computers.

Product Goals

The VMS host-based shadowing project was undertaken because the original controller shadowing product is architecturally incompatible with many prospective storage devices and their connectivity requirements. Controller shadowing requires an intelligent, common controller to access all physical devices in a shadow set. Devices such as the RF-series integrated storage elements (ISEs) with DSSI adapters and the RZ-series small computer systems interface (SCSI) disks present configurations that conflict with this method of access.

To support the range of configurations required by our customers, the new product had to be capable of shadowing physical devices located anywhere within a VAXcluster system and of doing so in a controller-independent fashion. The VAXcluster I/O system provides parallel access to storage devices from all nodes in a cluster simultaneously. In order to meet its performance goals, our shadowing product had to preserve this semantic also. Figure 2 shows clusterwide shadow sets for a hierarchical storage controller (HSC) configuration with multiple computer interconnect (CI) buses. When compared to Figure 1, this figure shows a larger cluster containing several clusterwide shadow sets. Note that multiple nodes in the cluster have direct, writable access to the disks comprising the shadow sets.

In addition to providing highly available access to shadow sets from anywhere in a cluster, the new shadowing implementation had other requirements. Phase II had to deliver performance comparable to that of controller-based shadowing, maximize application I/O impact on the design of the host-based shadowing implementation. Our goals to maximize application I/O availability during transient states, to provide customizable, event-driven design and fail-over, to enable all cluster nodes to manage the shadow sets, and to enhance availability, and ensure data integrity for critical applications.

In designing the new product, we benefited from customer feedback about the existing implementation. This feedback had a positive

system disk capabilities were all affected by customer feedback.

Technical Challenges

To provide volume shadowing in a VAXcluster environment running under the VMS operating system required that we solve complex, distributed systems problems.[1] This section describes the most significant technical challenges we encountered and the solutions we arrived at during the design and development of the product.

Membership Consistency. To ensure the level of integrity required for high availability systems, the shadowing design must guarantee that a shadow set has the same membership and states on all nodes in the cluster. A simple way to guarantee this property would have been a strict client-server implementation, where one VAX computer serves the shadow set to the remainder of the cluster. This approach, however, would have violated several design goals; the intermediate hop required by data transfers would decrease system performance, and any failure of the serving CPU would require a lengthy fail-over and rebuild operation, thus negatively impacting system availability.

To solve the problem of membership consistency, we used the VMS distributed lock manager through a new executive thread-level interface.[2,3] We cluster. Membership and state information about the shadow set is stored on all physical members in an on-disk data structure called the storage control block (SCB). One way that shadowing uses this SCB information is to automatically determine the most up-to-date shadow set member(s) when the set is created. In addition to distributed synchronization primitives, the VMS lock manager provides a capability for managing a distributed state variable called a lock value block. Shadowing uses the lock value block to define a disk that is guaranteed to be a current member of the shadow set. Whenever a membership change is made, all nodes take part in a protocol of lock operations; the value block and the on-disk SCB are the final arbiters of set constituency.

Sequential Commands. A sequential I/O command, i.e., a Mass Storage Control Protocol (MSCP) concept, forces all commands in progress to complete before the sequential command begins execution. While a sequential command is pending, all new I/O requests are stalled until that sequential designed a set of event-driven protocols that shadowing uses to guarantee membership consistency. These protocols allowed us to make the shadow set virtual unit a local device on all nodes in the

command completes execution. Shadowing requires the capability to execute a clusterwide, sequential command during certain operations. This capability, although a simple design goal for a client-server implementation, is a complex one for a distributed access model. We chose an event-driven, request/response protocol to create the sequential command capability.

Since sequential commands have a negative impact on performance, we limited the use of these commands to performing membership changes, mount/dismount operations, and bad block and merge difference repairs. Steady state processing never requires using sequential commands.

Full Copy. A full copy is the means by which a new member of the shadow set is made current with the rest of the set. The challenge is to make copy operations unintrusive; application I/Os must proceed with minimal impact so that the level of service provided by the system is both acceptable and predictable. VMS file I/O provides record-level sharing through the application transparent locking provided by the VAX RMS software, Digital's record management services. Shadowing operates at the physical device level to handle a variety of low-level errors. Because shadowing has no knowledge of the higher-layer on application I/O performance.

Merge Operations. Merge operations are triggered when a CPU with write access to a shadow set fails. (Note that with controller shadowing, merge operations are copy operations that are triggered when an HSC fails.) Devices may still be valid members of the shadow set but may no longer be identical, due to outstanding writes in progress when the host CPU failed. The merge operation must detect and correct these differences, so that successive application reads for the same data produce consistent results. As for full copy operations, the challenge with merge processing is to generate consistent results with minimal impact on application I/O performance.

Booting and Crashing. System disk shadowing presents some special problems because the shadow set must be accessible to CPUs in the cluster when locking protocols and inter-CPU communication are disabled. In addition, crashing must ensure appropriate behavior for writing crash dumps through the primitive bootstrap driver, including how record locking, a copy operation must guarantee that the application I/Os and the copy operation itself generate the correct results and do so with minimal impact

to propagate the dump to the shadow set. It was not practical to modify the bootstrap drivers because they are stored in read-only memory (ROM) on various CPU platforms that shadowing would support.

Error Processing. One major function of volume shadowing is to perform appropriate error processing for members of the shadow set, while maximizing data availability. To carry out this function, the software must prevent deadlocks between nodes and decide when to remove devices from the shadow set. We adopted a simple recovery ethic: a node that detects an error is responsible for fixing that error. Membership changes are serialized in the cluster, and a node only makes a membership change if the change is accompanied by improved access to the shadow set. A node never makes a change in membership without having access to some source members of the set.

Architecture

Phase II shadowing provides a local virtual unit on each node in the cluster with distributed control of that unit. Although the virtual unit is not served to the cluster, the underlying physical units that constitute a shadow set are served to the cluster using the standard VMS mechanisms. This scheme speeds. Controller shadowing does not provide this capability.

- Allows each node in the cluster to perform error recovery based on access to physical data source members. The shadowing software treats communication failures between any cluster node and shadow set members as normal shadowing events with customer-definable recovery metrics.

Major Components

VMS Volume Shadowing Phase II consists of two major components: SHDRIVER and SHADOW_SERVER. SHDRIVER is the shadowing virtual unit driver. As a client of disk class drivers, SHDRIVER is responsible for handling all I/O operations that are directed to the virtual unit. SHDRIVER issues physical I/O operations to the disk class driver to satisfy the shadow set virtual unit I/O requests. SHDRIVER is also responsible for performing all distributed locking and for driving error recovery.

SHADOW_SERVER is a VMS ancillary control process (ACP) responsible for driving copy and merge operations performed on has many data availability advantages. The Phase II design

- Allows shadowing to use all the VMS controller fail-over mechanisms for physical devices. As a result, member fail-over approaches hardware

the local node. Only one optimal node is responsible for driving a copy or merge operation on a given shadow set, but when a failure occurs the operation will fail over and resume on another CPU. Several factors determine this optimal node including the types of access paths, and controllers for the members and user-settable, per-node copy quotas.

Primitives

This section describes the locking protocols and error recovery processing functions that are used by the shadowing software. These primitives provide basic synchronization and recovery mechanisms for shadow sets in a VAXcluster system.

Locking Protocols. The shadowing software uses event-driven locking protocols to coordinate clusterwide activity. These request/response protocols provide maximum application I/O performance. A VMS executive interface to the distributed lock manager allows shadowing to make efficient use of locking directly from SHDRIVER.

One example of this use of locking protocols in VMS Volume Shadowing Phase II is the sequential command protocol. As mentioned in the Technical Challenges section, shadowing requires the sequential command capability but minimizes the use of this primitive. Phase II implements the capability by using several locks, as described in the following series of events.

A node that needs to sequential stall requests to other nodes that have the shadow set mounted. This initiating thread waits until all other nodes in the cluster have flushed their I/Os and responded to the node requesting the sequential operation. Once all nodes have responded or left the cluster, the operations that compose the sequential command execute. When this process is complete, the locks are released, allowing asynchronous threads on the other nodes to proceed and automatically resume I/O operations. The local node resumes I/O as well.

Error Recovery Processing. Error recovery processing is triggered by either asynchronous notification of a communication failure or a failing I/O operation directed towards a physical member of the shadow set. Two major functions of error recovery are built into the virtual unit driver: active and passive volume processing.

Active volume processing is triggered directly by events that occur on a local node in the cluster. This type of volume processing uses a simple, localized ethic for error recovery from communication or controller failures. Shadow set membership execute a sequential command first stalls I/O locally and flushes operations in progress. The node then performs lock operations that ensure serialization and sends

decisions are made locally, based on accessibility. If no members of a shadow set are currently accessible from a node, then the membership does not change. If some but not all members of the set are accessible, the local node, after attempting fail-over, removes some members to allow application I/O to proceed. The system manager sets the time period during which members may attempt fail-over. The actual removal operation is a sequential command. The design allows for maximum flexibility and quick error recovery and implicitly avoids deadlock scenarios.

Passive volume processing responds to events that occur elsewhere in the cluster; messages from nodes other than the local one trigger the processing by means of the shadowing distributed locking protocols. This volume processing function is responsible for verifying the shadow set membership and state on the local node and for modifying this membership to reflect any changes made to the set by the cluster. To accomplish these operations, the shadowing software first reads the lock value block to find a disk guaranteed to still be in the shadow set. Then the recovery process retrieves the physical member's on-disk SCB data and uses this information to perform the relevant data structure updates on the local node. volume processing, the I/O requests are stalled because the membership of the set is in doubt, and correct processing of the request cannot be performed until the situation is corrected.

Steady State Processing

The shadowing virtual unit driver receives application read and write requests and must direct the I/O appropriately. This section describes these steady state operations.

Read Algorithms

The shadowing virtual unit driver receives application read requests and directs a physical I/O to an appropriate member of the set. SHDRIVER attempts to direct the I/O to the optimum device based on locally available data. This decision is based on (1) the access path, i.e., local or served by the VMS operating system, (2) the service queue lengths at the candidate controller, and (3) a round-robin algorithm among equal paths. Figure 3 shows a shadow set read operation. An application read to the shadow set causes a single physical read to be sent to an optimal member of the set. In Figure 3, there is

Application I/O requests to the virtual unit are always stalled during volume processing. In the case of active volume processing, the stalling is necessary because many I/Os would fail until the error was corrected. In passive

one local and one remote member, so the read is sent to the local member.

Data repair operations caused by media defects are triggered by a read operation failing with an appropriate error, such as forced error or parity. The shadowing driver attempts this repair using another member of the shadow set. This repair operation is performed with the synchronization of a sequential command. Sequential protection is required because a read operation is being converted into a write operation without explicit, RMS-layer synchronization.

Write Algorithms

The shadowing virtual unit driver receives application write requests and then issues, in parallel, write requests to the physical members of the set. The virtual unit write operation does not complete until all physical writes complete. A shadow set write operation is shown in Figure 4. Physical write operations to member units can fail or be timed out; either condition triggers the shadowing error recovery logic and can cause a fail-over or the removal of the erring device from the shadow set.

Transient State Processing

Shadowing performs a variety of operations in order to maintain consistency among the members of the set. These operations include full copy, merge, and data repair and recovery. This section describes these transient state operations.

Full Copy

Full copy operations are performed under direct system manager control. When a disk is added to the shadow set, copy operations take place to make the contents of this new set member identical to that of the other members. Copy operations are transparent to application processing. The new member of the shadow set does not provide any data availability protection until the copy completes.

There is no explicit gatekeeping during the copy operation. Thus, application read and write operations occur in parallel with copy thread reads and writes. As shown in Figure 5, correct results are accomplished by the following algorithm. During the full copy, the shadowing driver processes application write operations on each logical block number (LBN) range until the compare operation succeeds. If an LBN range has such frequent activity that the compare fails many times, SHDRIVER performs a synchronized update. A distributed fence provides a clusterwide boundary between the copied and the uncopied areas of the new member. This fence is used to avoid performing the special full copy mechanisms on application writes to that area of the disk already processed by the copy thread.

This algorithm meets the goal of operational correctness (both the application and the copy thread achieve the proper results with regard to the contents of the shadow set members) and requires no synchronization with the copy thread. Thus, the algorithm achieves maximum application I/O availability during the transient state. Crucial to achieving this goal is the fact that, by design, the copy thread does not perform I/O optimization techniques such as double buffering. The copy operations receive equal service as application I/Os.

Merge Operations

operations in two groups: first, those directed to all source members and second, writes to all full copy targets. The copy thread performs a sequence of read source, compare target, and write target

The VMS Volume Shadowing Phase II merge algorithm meets the product goals of operational correctness, while maintaining high application I/O availability and minimal synchronization. A merge operation is required when a CPU crashes with the shadow set mounted for write operations. A merge is needed to correct for the possibility of partially completed writes that may have been outstanding to the physical set members when the failure occurred. The merge operation ensures that all members contain identical data, and thus the shadow set virtual unit behaves like a single, highly available disk. It does not matter which data is more recent, only that the members are the same. This satisfies the purpose of shadowing, which is to provide data availability. But since the failure occurred while a write operation was in progress, this consistent shadow set can contain either old or new data. To make sure that the shadow set contains the most recent data, a data integrity technique such as journaling must be employed.

In Phase II shadowing, merge processing is distinctly different from copy processing. The

shadow set provides full availability protection during the merge. As a result, merge processing is intentionally designed to be a background activity and to maximize application I/O throughput while the merge is progressing. The merge thread carefully monitors I/O rates and inserts a delay between its I/Os if it detects contention for shared system resources, such as adapters and interconnects.

In addition to maximizing I/O availability, the merge algorithm is designed to minimize synchronization with application I/Os and to identify and correct data inconsistencies. Synchronization takes place only when a rare difference is found. When an application read operation is issued to a shadow set in the merge state, the set executes the read with merge semantics. Thus, a read to a source and a parallel compare with the other members of the set are performed. Usually the compare matches and the operation is complete. If a mismatch is detected, a sequential disk already processed by the merge thread. Figure 6 illustrates the merge algorithm.

Note that controller shadowing performs an operation called a merge copy. Although this HSC merge copy operation is designed for the same purpose as the Phase II operation, the approaches differ greatly. An HSC merge copy is triggered when an HSC, not a shadow set, fails and performs a copy operation; the HSC merge copy does not detect differences.

Performance Assists

A future version of the shadowing product is intended to utilize controller performance assists to improve copy and merge operations. These assists will be used automatically, if supported by the controllers involved in accessing the physical members of a shadow set.

COPY_DATA is the ability of a host to control a direct disk-to-disk transfer without the data entering or leaving the host CPU I/O adapters and memory. This capability will be used by full copy processing to decrease the system impact, the bandwidth, and the time required for a full copy.

repair operation begins. The merge thread scans the entire disk in the same manner as the read, looking for differences. A distributed fence is used to avoid performing merge mechanisms for application reads to that area of the

The members of the set and /or their controllers must share a common interconnect in order to use this capability. The COPY_ DATA operation performs specific shadowing around the active, copy LBN range to ensure correctness.

This operation involves
LBN range-based gatekeeping
in the copy target device
controller.

Controller write logging is a future capability in controllers, such as HSCs, that will allow more efficient merge processing. Shadowing write operation messages will include information for the controller to log I/Os in its memory. These logs will then be used by the remaining host CPUs during merge processing to determine exactly which blocks contain outstanding write operations from a failed CPU. With such a performance assist, merge operations will take less time and will have less impact on the system.

Data Repair and Recovery

As discussed in the Primitives section,

data repair operations are triggered by failing reads and are repaired as sequential commands. Digital Storage Architecture (DSA) devices support two primitive capabilities that are key to this repair mechanism. When a DSA controller detects a media error, the block in question is sometimes repaired automatically, thus requiring no shadowing intervention. When the controller cannot repair the data, a spare block is revectored to this LBN,

The forced error returned on a read operation is the signal to the shadowing software to execute a repair operation. SHDRIVER attempts to read usable data from another source device. If such data is available, the software writes the data to the revectored block and then returns the data to the application. If no usable data source is available, the software performs write operations with a forced error to all set members and signals the application that this error condition has occurred. Note that a protected system buffer is used for this operation because the application reading the data may not have write access.

A future shadowing product is intended to support SCSI peripherals, which do not have the DSA primitives outlined above. There is no forced error indicator in the SCSI architecture, and the revector operation is nonatomic. To perform shadowing data repair on such devices, we will use the READL/WRITEL capability optionally supported by SCSI devices. These I/O functions allow blocks to be read and written with error correction code (ECC) data. Shadowing emulates forced error

and the contents of the
block are marked with a
forced error. This causes
subsequent read operations
to fail, since the contents
of the block are lost.

by writing data with an
intentionally incorrect
ECC. To circumvent the
lack of atomicity on the
revector operation, a
device being repaired is
temporarily marked as a
full copy target until

the conclusion of the repair operation. If the CPU fails in the middle of a repair operation, the repair target is now a full copy target, which preserves correctness in the presence of these nonatomic operations.

System Disk

System disk shadow sets presented some unique design problems. The system disk must be accessed through a single bootstrap driver and hence, a single controller type. This access takes place when multihost synchronization is not possible. These two access modes occur during system bootstrap and during a crash dump write.

Shadowed Booting

The system disk must be accessed by the system initialization code executing on the booting node prior to any host-to-host communication. Since the boot drivers on many processors reside in ROM, it was impractical to make boot driver modifications to support system disk processing. To solve this problem, the system disk operations performed prior to the controller initialization routine of the system device driver are read-only. It is safe to read data from a time, shadowing builds a read-only shadow set that contains only the boot member. Once locking is enabled, shadowing performs a variety of checks on the system disk shadow set to determine whether or not the boot is valid. If the boot is valid, shadowing turns the single-member, read-only set into a multimember, writable set with preserved copy states. If this node is joining an existing cluster, the system disk shadow set uses the same set as the rest of the cluster.

Crash Dumps

The primitive boot driver uses the system disk to write crash dumps when a system failure occurs. This driver only knows how to access a single physical disk in the shadow set. But since a failing node automatically triggers a merge operation on shadow sets mounted for write, we can use the merge thread to process the dump file. The merge occurs either when the node leaves the cluster (if there are other nodes present) or later, when the set is reformed. As the source for merge difference repairs, the merge process attempts to use the member to which the dump file was written and propagates the dump file to the remainder of the set.

clusterwide, shared device without synchronization when there is little or no risk of the data being modified by another node in the cluster. At controller initialization

The mechanism here for dump file propagation is best-effort, not guaranteed; but since writing the dump is always best-effort, this solution is considered acceptable.

Conclusion

VMS Volume Shadowing Phase II is a state-of-the-art implementation of distributed data availability. The project team arrived at innovative solutions to problems attributable to a set of complex, conflicting goals. Digital has applied for four patents on various aspects of this technology.

Acknowledgements

I would like to acknowledge the efforts and contributions of the other members of the VMS shadowing engineering team: Renee Culver, William Goleman, and Wai Yim. In addition, I would also like to acknowledge Sandy Snaman for Fork Thread Locking, Ravindran Jagannathan for performance analysis, and David Thiel for general consulting.

References

1. N. Kronenberg, H. Levy, W. Strecker, and R. Merewood, "The VAXcluster Concept: An Overview of a Distributed System," Digital Technical Journal (September 1987): 7-21.

2. W. Snaman, Jr. and D. Thiel, "The VAX/VMS Distributed Lock Manager," Digital Technical Journal (September 1987): 29-44.

3. W. Snaman, Jr., "Application Design in a VAXcluster System," Digital Technical Journal, vol 3. no. 3 (Summer 1991, this issue): 16-26.