VMware vSphere APIs: Array Integration (VAAI)

Table of contents

VMware vSphere APIs: Array Integration (VAAI)

Introduction to VAAI

This section provides an introduction to VAAI and how it provides hardware assistance to storage vendors.

In a virtualized environment, storage operations traditionally have been expensive from a resource perspective. Functions such as cloning and snapshots can be performed more efficiently by the storage device than by the host.

VMware vSphere® Storage APIs – Array Integration (VAAI), also referred to as hardware acceleration or hardware offload APIs, are a set of APIs to enable communication between VMware vSphere ESXi™ hosts and storage devices. The APIs define a set of "storage primitives" that enable the ESXi host to offload certain storage operations to the array, which reduces resource overhead on the ESXi hosts and can significantly improve performance for storage-intensive operations such as storage cloning, zeroing, and so on. The goal of VAAI is to help storage vendors provide hardware assistance to speed up VMware® I/O operations that are more efficiently accomplished in the storage hardware.

Without the use of VAAI, cloning or migration of virtual machines by the vSphere VMkernel Data Mover involves software data movement. The Data Mover issues I/O to read and write blocks to and from the source and destination datastores. With VAAI, the Data Mover can use the API primitives to offload operations to the array if possible. For example, if the desired operation were to copy a virtual machine disk (VMDK) file from one datastore to another inside the same array, the array would be directed to make the copy completely inside the array. Whenever a data movement operation is invoked and the corresponding hardware offload operation is enabled, the Data Mover will first attempt to use the hardware offload. If the hardware offload operation fails, the Data Mover reverts to the traditional software method of data movement.

In nearly all cases, hardware data movement will perform significantly better than software data movement. It will consume fewer CPU cycles and less bandwidth on the storage fabric. Improvements in performance can be observed by timing operations that use the VAAI primitives and using *esxtop* to track values such as CMDS/s, READS/s, WRITES/s, MBREAD/s, and MBWRTN/s of storage adapters during the operation.

In the initial VMware vSphere 4.1 implementation, three VAAI primitives were released. These primitives applied only to block (Fibre Channel, iSCSI, FCoE) storage. There were no VAAI primitives for NAS storage in this initial release. In vSphere 5.0, VAAI primitives for NAS storage and VMware vSphere Thin Provisioning were introduced. In various releases of vSphere since then, different methods to handle dead space reclamation via UNMAP have been introduced. This paper focuses on the most recent mechanism introduced in vSphere 6.5.

VAAI Block Primitives

This section describes the different types of VAAI Block Primitives, their functionalities and capabilities.

Atomic Test & Set (ATS)

In VMware vSphere VMFS, many operations must establish a lock on the volume when updating a resource. Because VMFS is a clustered file system, many ESXi hosts can share the volume. When one host must make an update to the VMFS metadata, a locking mechanism is required to maintain file system integrity and prevent another host from coming in and updating the same metadata. The following operations require this lock:

- Acquire on-disk locks
- Upgrade an optimistic lock to an exclusive/physical lock
- Unlock a read-only/multi-writer lock
- Acquire a heartbeat
- Maintain “liveness” of a heartbeat
- Clear a heartbeat
- Replay a heartbeat
- Reclaim a heartbeat
- Acquire on-disk lock with dead owner

It is not essential to understand all of these operations in the context of this paper. It is sufficient to understand that various VMFS metadata operations require a lock.

ATS is an enhanced locking mechanism designed to replace the use of SCSI reservations on VMFS volumes when doing metadata updates. A SCSI reservation locks a whole LUN and prevents other hosts from doing metadata updates of a VMFS volume when one host sharing the volume has a lock. This can lead to various contention issues when many virtual machines are using the same datastore. It is a limiting factor for scaling to very large VMFS volumes.

ATS is a lock mechanism that modifies only a disk sector on the VMFS volume. When successful, it enables an ESXi host to perform a metadata update on the volume. This includes allocating space to a VMDK during provisioning, because certain characteristics must be updated in the metadata to reflect the new size of the file. The introduction of ATS addresses the contention issues with SCSI reservations and enables VMFS volumes to scale to much larger sizes. ATS has the concept of a test-image and set-image. So long as the image on-disk is as expected during a “compare”, the host knows that it can continue to update the lock.

In vSphere 4.0, VMFS3 used SCSI reservations for establishing the lock, because there was no VAAI support in that release. In vSphere 4.1, on a VAAI-enabled array, VMFS3 used ATS for only operations 1 and 2 listed previously, and only when there was no contention for disk lock acquisitions. VMFS3 reverted to using SCSI reservations if there was a multi-host collision when acquiring an on-disk lock using ATS.

The use of ATS to maintain “liveness” of heartbeats (operation 5 above) was introduced in vSphere 5.5U2. Prior to this release, SCSI reservations were used to maintain the “liveness” of the heartbeat.

One point of note with VMFS3 using ATS; if there was an mid-air collision from 2 hosts trying to lock the same sector via ATS, the hosts would revert to SCSI reservations on retry. For VMFS5 and VMFS6 datastores formatted on a VAAI-enabled array, all the critical section functionality from operations 1 to 9 listed above is done using ATS. There no longer should be any SCSI reservations on VAAI-enabled VMFS5. ATS continues to be used even if there is contention.

In the initial VAAI release, the ATS primitives had to be implemented differently on each storage array, requiring a different ATS opcode depending on the vendor. ATS is now a standard T10 SCSI command and uses opcode **0x89** (COMPARE AND WRITE).

On non-VAAI enabled storage, SCSI reservations continue to be used for establishing critical sections in VMFS5.

ATS Only Flag

An **ATS only** flag appears on newly created VMFS5 and VMFS6 datastores. When the host detects that the array supports ATS on the device, the ATS only flag is written to the datastore. From that point on, ATS always will be used on that datastore, so it can be used for all metadata operations.

The flag is not enabled, however, on VMFS5 volumes that were upgraded from VMFS3, facilitating seamless upgrades from VMFS3 to VMFS5. This is because VMFS3 was not ATS complete, and during the upgrade process, some hosts might continue to use SCSI reservations for certain metadata operations.

You cannot turn on the ATS only flag on VMFS5 volumes upgraded from VMFS3. The ATS only flag can be manually enabled or disabled on a newly created VMFS5. To manually enable it, you can use the hidden option:

```
# vmkfstools –configATSOnly 1 [device path]
```

Since there is no upgrade path from VMFS3 or VMFS5 to VMFS6, this is not a concern for VMFS6 volumes.

ATS Miscompare

When ATS was introduced for maintaining heartbeat “liveness” in vSphere 5.5U2, some issues arose. Some storage arrays returned ATS miscompares when the array was overloaded. In another scenario, reservations on the LUN on which the VMFS resided also caused ATS miscompares. In another case, the ATS “set” was correctly written to disk, but the array still returned a miscompare.

But it wasn’t all array related issues. Situations arose where VMFS also detected an ATS miscompare incorrectly. In this case, a heartbeat I/O (1) got timed-out and VMFS canceled that I/O, but before canceling the I/O, the I/O (ATS “set”) actually made it to the disk. VMFS next re-tried the ATS using the original “test-image” in step (1) since the previous one was canceled, and the assumption was that the ATS didn’t make it to the disk. Since the ATS “set” made it to the disk before the cancel, the ATS “test” meant that the in-memory and on-disk images no longer matched, so the array returned “ATS miscompare”.

When an ATS miscompare is received, all outstanding IO is canceled. This led to additional stress and load being placed on the storage arrays and degraded performance.

In vSphere 6.5, there are new heuristics added so that when an ATS miscompare event is encountered, VMFS reads the on-disk heartbeat data and verifies it against the ATS “test” and “set” images. This check to see if there is actually a real miscompare. If the miscompare is real, then we do the same as before, which is to cancel all the outstanding I/O. If the on-disk heartbeat data has *not* changed, then this is a false miscompare. In the event of a false miscompare, VMFS will not immediately cancled IOs. Instead, VMFS will now re-attempt the ATS heartbeat operation after a short interval (usually less than 100ms).

XCOPY (Extended Copy)

Without VAAI, a clone or migrate operation must use the VMkernel software Data Mover driver. If the files being cloned are multiple GB in size, the operation might last for many minutes to many hours. This full copy consumes host resources such as CPU cycles, DMA buffers and SCSI commands in the HBA queue. This VAAI primitive requests that the array performs a full copy of blocks on behalf of the Data Mover. It primarily is used in clone and migration operations (such as a VMware vSphere Storage vMotion®). The SCSI opcode for XCOPY is **0x83**.

Write Same (Zero)

When provisioning an eagerzeroedthick VMDK, a SCSI command is issued to write zeroes to disk blocks. Again, this initialization request consumes host resources such as CPU cycles, DMA buffers and SCSI commands in the HBA queue.

One of the most common operations on virtual disks is initializing large extents of the disk with zeroes to isolate virtual machines and promote security. ESXi hosts may call the WRITE_SAME SCSI command to zero out large portions of a disk. This offload task will zero large numbers of disk blocks without transferring the data over the transport link. The WRITE_SAME SCSI opcode is **0x93** .

The following provisioning tasks are accelerated by the use of the WRITE_SAME command:

- Cloning operations for eagerzeroedthick target disks
- Allocating new file blocks for thin-provisioned virtual disks
- Initializing previous unwritten file blocks for zeroedthick virtual disks

NOTE: Some storage arrays, on receipt of the WRITE_SAME SCSI command, will write zeroes directly to disk. Other arrays do not need to write zeroes to every location; they simply do a metadata update to write a page of all zeroes. Therefore, it is possible that you will observe significant differences in the performance of this primitive, depending on the storage array

UNMAP

One VAAI primitive, UNMAP, enables an ESXi host to inform the storage array that space, previously occupied by a virtual machine that has been migrated to another datastore or deleted, can now be reclaimed. This is commonly referred to as ‘dead space reclamation’ and enables an array to accurately report space consumption of a thin-provisioned datastore as well as enabling users to monitor and correctly forecast new storage requirements.

The mechanism for conducting a space reclamation operation has been regularly updated since the primitive was introduced in vSphere 5.0. Initially, the operation was automatic. When a virtual machine was migrated from a datastore or deleted, the UNMAP was called immediately and space was reclaimed on the array. There were some issues with this approach, primarily regarding performance and an array's ability to reclaim the space in an optimal time frame. For this reason, the UNMAP operation was changed to a manual process. However, with the release of vSphere 6.5, the UNMAP primitive is once again automatic, as long as the underlying datastore is formatted with VMFS6. It is enabled automatically on VMFS6 volumes. The automated UNMAP crawler mechanism for reclaiming dead or stranded space on VMFS datastores now runs continuously in the background.

With esxcli, users can display device-specific details regarding Thin Provisioning and VAAI.

```
[root@esxi-dell-e:~] esxcli storage core device list -d naa.624a9370d4d78052ea564a7e00011030
naa.624a9370d4d78052ea564a7e00011030
   Display Name: PURE Fibre Channel Disk (naa.624a9370d4d78052ea564a7e00011030)
   Has Settable Display Name: true
   Size: 10240
   Device Type: Direct-Access
   Multipath Plugin: NMP
   Devfs Path: /vmfs/devices/disks/naa.624a9370d4d78052ea564a7e00011030
   Vendor: PURE
   Model: FlashArray
   Revision: 8888
   SCSI Level: 6
   Is Pseudo: false
   Status: on
   Is RDM Capable: true
   Is Local: false
   Is Removable: false
   Is SSD: true
   Is VVOL PE: false
   Is Offline: false
   Is Perennially Reserved: false
   Queue Full Sample Size: 0
   Queue Full Threshold: 0
   Thin Provisioning Status: yes
   Attached Filters:
   VAAI Status: supported
   Other UIDs: vml.0200f40000624a9370d4d78052ea564a7e00011030466c61736841
   Is Shared Clusterwide: true
   Is Local SAS Device: false
   Is SAS: false
   Is USB: false
   Is Boot USB Device: false
   Is Boot Device: false
   Device Max Queue Depth: 64
   No of outstanding IOs with competing worlds: 32
   Drive Type: unknown
   RAID Level: unknown
   Number of Physical Drives: unknown
   Protection Enabled: false
   PI Activated: false
   PI Type: 0
   PI Protection Mask: NO PROTECTION
   Supported Guard Types: NO GUARD SUPPORT
   DIX Enabled: false
   DIX Guard Type: NO GUARD SUPPORT
   Emulated DIX/DIF Enabled: false
[root@esxi-dell-e:~]
```

Users can also display the VAAI primitives supported by the array for that device, including whether the array supports the UNMAP primitive for dead space reclamation (referred to as the Delete Status). Another esxcli command is used for this step, as is shown in the following:

```
[root@esxi-dell-e:~] esxcli storage core device vaai status get -d  naa.624a9370d4d78052ea564a7e00011030
naa.624a9370d4d78052ea564a7e00011030
   VAAI Plugin Name:
   ATS Status: supported
   Clone Status: supported
   Zero Status: supported
   Delete Status: supported
[root@esxi-dell-e:~]
```

The device displays “Delete Status: supported” to indicate that it is capable of sending SCSI UNMAP commands to the array when a space reclamation operation is requested . If a Storage vMotion operation is initiated and a virtual machine is moved from a source datastore to a destination datastore, the array will reclaim that space.

The granularity of the reclaim is set to 1MB chunks. Automatic UNMAP is not supported on arrays with UNMAP granularity greater than 1MB. Therefore, customers should check the VMware Hardware Compatibility Guide (HCL) footnotes of your storage array to check if the Auto UNMAP feature is supported.

Disabling the UNMAP primitive does not affect any of the other Thin Provisioning primitives such as Thin Provisioning Stun and the space threshold warning. All primitives are orthogonal.

Note that UNMAP is only automatically issued to VMFS datastores that are VMFS-6 and have powered-on VMs. Since UNMAP is automated, it can take 12-24 hours to fully reclaim any dead space on the datastore. UNMAP can still be run manually against older VMFS volumes.

UNMAP from inside a Guest OS

In vSphere 6.0, additional improvements to UNMAP facilitate the reclaiming of stranded space from within a Guest OS. Effectively, there is now the ability for a Guest OS with a thinly provisioned VMDK to inform the backing storage that blocks can be reclaimed, and in turn, shrink the size of the VMDK.

Initially in-guest UNMAP support to reclaim in-guest dead space natively was limited to Windows 2012 R2. Linux distributions could not do in-guest reclaim due to the SCSI version implemented in the Virtual Hardware (SCSI version 2). Linux distributions check the SCSI version, and unless it is version 5 or greater, it does not send UNMAPs. With SPC-4 support introduced in vSphere 6.5 for Virtual Hardware, Linux Guest OS are now able to issue UNMAPs.

A best practice is to align Guest OS partitions to the 1MB granularity boundary. If the Guest OS filesystem is not aligned, UNMAPs may not work correctly.

For Linux distributions, *fstrim* command can be used to manually reclaim dead space from within the Guest OS. To have this happen automatically, the filesystem must be mounted with the *–o discard* option. With this option, the OS issues discard requests automatically as files are deleted from the filesystem.

One final point to make on in-guest UNMAPS. Prior to vSphere 6.5, in-guest UNMAP did not work if Change Block Tracking (CBT) was enable on the VM. CBT could be in use for backups or for replication of the VM. In vSphere 6.5, in-guest UNMAP also works if CBT is enabled on the VM, which is a great improvement.

TRIM Considerations

TRIM is the ATA equivalent of SCSI UNMAP. A TRIM operation gets converted to UNMAP in the I/O stack, which is SCSI. However, there are some issues with TRIM getting converted into UNMAP. UNMAP works at certain block boundaries on VMFS, whereas TRIM does not have such restrictions. While this should be fine on VMFS-6, which is now 4K aligned, certain TRIMs converted into UNMAPs may fail due to block alignment issues on previous versions of VMFS.

VAAI NAS Primitives

This section explains the functions and capabilities of various types of VAAI NAS Primitives.

Prior to vSphere 5.0, VMware supported hardware acceleration on block storage devices. As of the vSphere 5.0 release, VMware has also included a number of new NAS hardware acceleration primitives. Unlike block primitives, VAAI NAS primitives will be available only through the use of a VAAI-NAS vendor plug-in, which is provided by your NAS storage array vendor.

Full File Clone

Full File Clone, or Full Copy, is similar to the Extended Copy (XCOPY) hardware acceleration primitive provided for block arrays. This primitive enables virtual disks to be cloned by the NAS array rather than by using the Data Mover, which consumes ESXi host CPU and memory resources as well as network bandwidth. There is one noticeable difference between this primitive and the XCOPY primitive on block storage. This VAAI-NAS primitive cannot be called for Storage vMotion operations, which continue to use the Data Mover. Only virtual machines that are powered off and then migrated can offload to the storage array by using this primitive. The Storage vMotion restriction aside, the benefit is that cold clones or “deploy from template” operations can be offloaded to the storage hardware, and this reduces the resource consumption of the ESXi host.

Fast File Clone/Native Snapshot Support

Fast File Clone enables the creation of virtual machine snapshots to be offloaded to an NAS storage array. This is a primitive that forms the basis of VMware View Composer™ Array Integration (VCAI), which was released as a technical preview in VMware Horizon View™ 5.1. VMware View Composer can elect to have desktops based on linked clones created directly by the storage array rather than by using ESXi host CPU and memory resources for the same task. This became fully supported in VMware Horizon View™ 5.3.

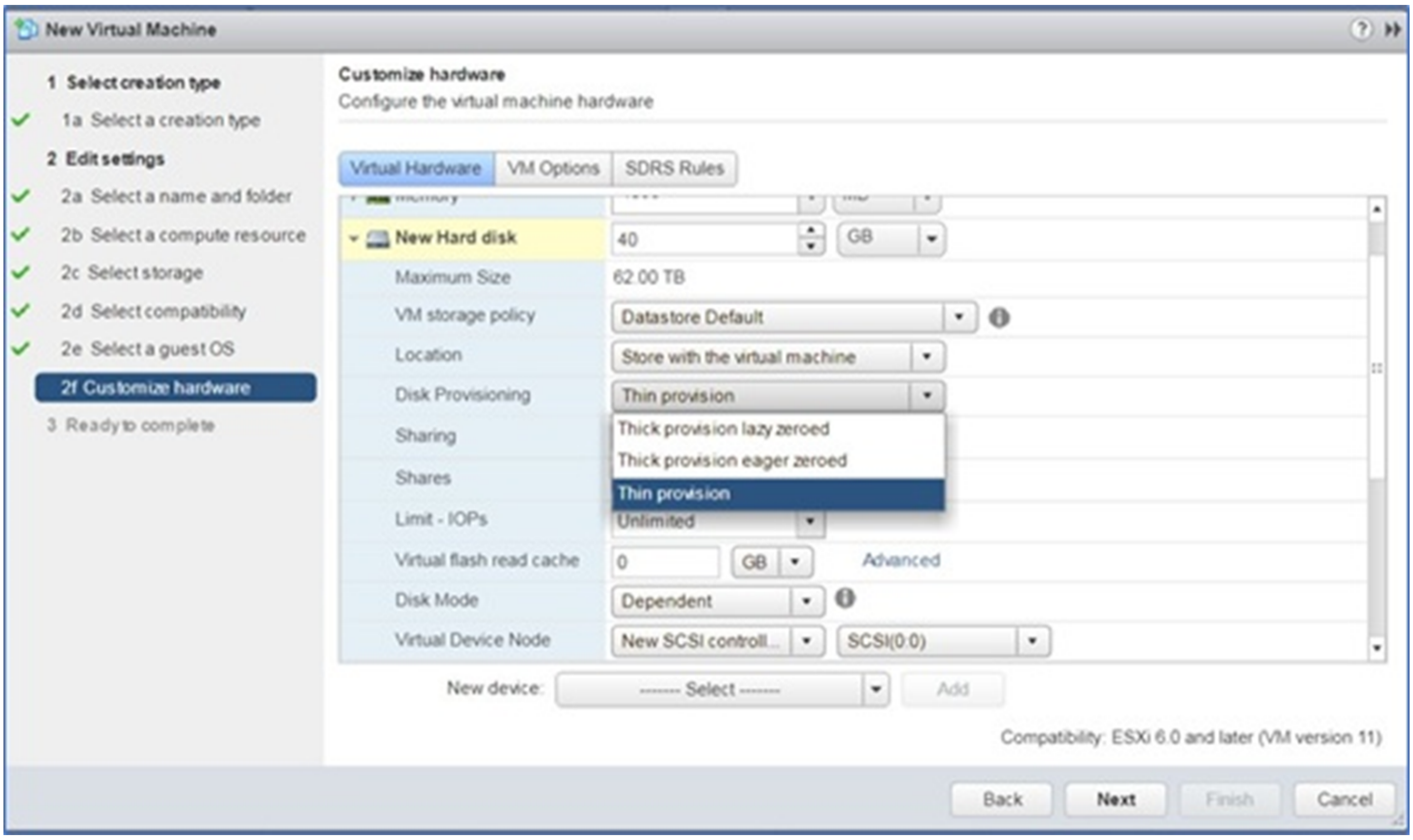
In vSphere 5.1, this primitive has been extended to VMware vCloud® vApps™, whereby VMware vCloud Director™ can elect to have vCloud vApps based on linked clones instantiated by the storage array rather than the ESXi host.

Extended Statistics

This primitive enables visibility into space usage on NAS datastores. This is especially useful for thin-provisioned datastores because it enables vSphere to display actual space usage statistics in situations where oversubscription was used. Previously, vSphere administrators needed to use array-based tools to manage and monitor how much space a thinly provisioned VMDK was consuming on a back-end datastore. With this new primitive, this information can be surfaced up in the VMware vSphere Client™. This enables vSphere administrators to have a much better insight into storage resource consumption, without the need to rely on third-party tools or engage with their storage array administrators.

Reserve Space

In previous versions of vSphere, only a thin VMDK could be created on NAS datastores. This new reserve space primitive enables the creation of thick VMDK files on NAS datastores, so administrators can reserve all of the space required by a VMDK, even when the datastore is NAS. As shown in Figure 1, users now have the ability to select lazy-zero or eager-zero disks on NAS datastore.


New Virtual Machine

1 Select creation type
✓ 1a Select a creation type
2 Edit settings
✓ 2a Select a name and folder
✓ 2b Select a compute resource
✓ 2c Select storage
✓ 2d Select compatibility
✓ 2e Select a guest OS
2f Customize hardware
3 Ready to complete

Customize hardware
Configure the virtual machine hardware

Virtual Hardware | VM Options | SDRS Rules

New Hard disk	40	GB
Maximum Size	62.00 TB	
VM storage policy	Datastore Default	
Location	Store with the virtual machine	
Disk Provisioning	Thin provision	
Sharing		
Shares		
Limit - IOPs	Unlimited	
Virtual flash read cache	0	GB
Disk Mode	Dependent	
Virtual Device Node	New SCSI controll...	SCSI(0:0)

Thick provision lazy zeroed
Thick provision eager zeroed
Thin provision

New device: -------- Select -------- Add

Compatibility: ESXi 6.0 and later (VM version 11)

Back Next Finish Cancel

Figure 1. Thick-Disk Provisioning on NFS Datastores

VAAI Thin Provisioning Primitives

This section provides an introduction to the different types of VAAI Thin Provisioning Primitives and explains why the new primitives have been added.

In environments where thin provisioning is used in the management and monitoring of datastores, there are frequent complaints that the array is not informed that blocks have become free when virtual machines have been deleted or migrated from a datastore. This leads to array management tools' reporting much higher space consumption than is actually the case. Out-of-space (OOS) concerns are also common. In the past, OOS conditions could impact all virtual machines on the OOS thin-provisioned datastore. To address these concerns, multiple enhancements were made to VAAI in vSphere 5.0. In particular, several new primitives have been added to better support Thin Provisioning.

Thin Provisioning Stun

The first enhancement was introduced to address concern regarding the impact on virtual machines when thin-provisioned datastore usage reaches 100 percent of capacity. Historically, this affected all virtual machines running on the datastore. With the release of the vSphere 5.0 VAAI Thin Provisioning primitives, if a thin-provisioned datastore reaches 100 percent usage, only those virtual machines requiring extra blocks of storage space are paused; those not needing additional space continue to run. After additional space is allocated to the thin-provisioned datastore, the paused virtual machines can be resumed.

Thin Provisioning Space Threshold Warning

vSphere cannot detect how much space a thin-provisioned datastore is consuming at the back end. The first indication a vSphere administrator has that an overcommitted datastore has no available space is that no more virtual machines can be provisioned and other virtual machines stop running. With this new primitive, a warning is raised and surfaced in VMware vCenter™ via VAAI if a thin-provisioned datastore reaches a specific threshold. The threshold values are set on the array, if there is a means to do so, and are defined by the storage administrator; vSphere does not set this threshold. This enables an administrator to proactively add more storage, extend the datastore, or run a Storage vMotion operation to migrate some virtual machines off of the datastore to avoid OOS conditions. By default, it seems the threshold on the majority of arrays that support this primitive is 75%.

VAAI Tuning and Monitoring

This section describes some of the options available for fine-tuning and monitoring the VAAI primitives. This will help administrators determine whether VAAI is working correctly in their respective environments.

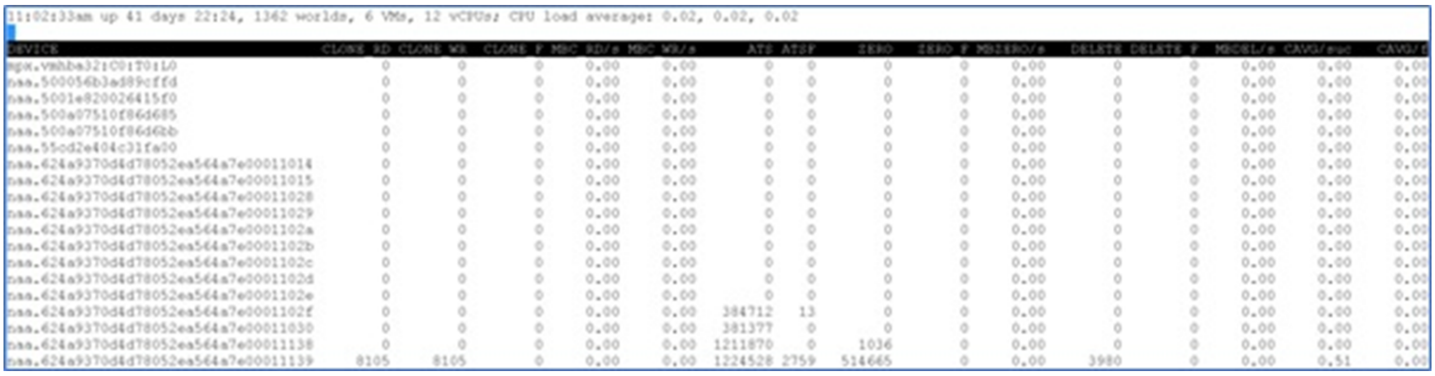
We first will discuss XCOPY. The default XCOPY size is 4MB. With a 32MB I/O, the expectation would be to see this counter in *esxtop* incrementing in batches of eight, the number of work items that will be created to deliver a 32MB I/O. The default XCOPY size can be incremented to a maximum value of 16MB if required, but that should be done only on the advice of your storage array vendor. The following command shows how to query the current transfer size and how to change it:

```
# esxcfg-advcfg -g /DataMover/MaxHWTransferSize
Value of MaxHWTransferSize is 4096
# esxcfg-advcfg -s 16384 /DataMover/MaxHWTransferSize
 Value of MaxHWTransferSize is 16384
```

NOTE: Please use extreme caution when changing this advanced parameter. This parameter is a global parameter, so it will impact ALL storage arrays attached to the host. While a storage array vendor might suggest making a change to this parameter for improved performance on their particular array, it may lead to issues on other arrays which do not work well with the new setting, including degraded performance.

The data-out buffer of the WRITE_SAME command contains all zeroes. A single zero operation has a default zeroing size of 1MB. The maximum number of outstanding WRITE_SAME commands is 32. We currently do not support changing the WRITE_SAME size of 1MB.

One can observe the clone operations (XCOPY), ATS operations, and zero operations (WRITE_SAME) in an *esxtop* output. The following is an *esxtop* output showing the various block primitives:



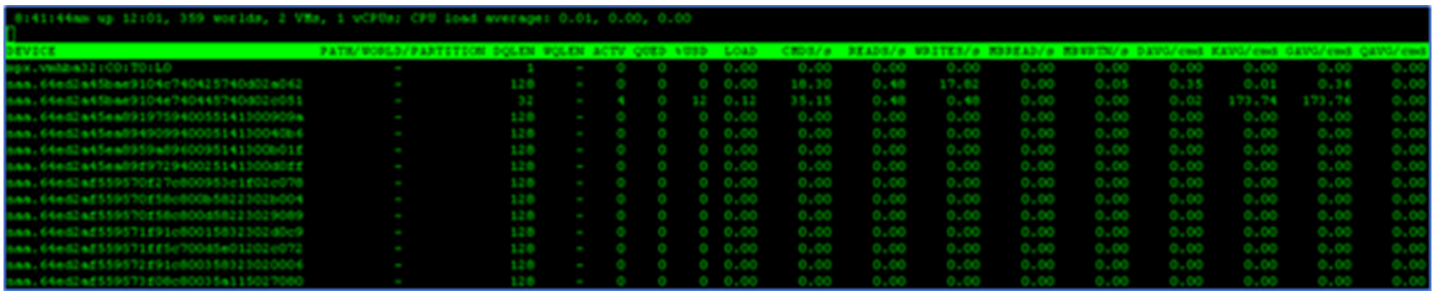
```
11:02:33am up 41 days 22:24, 1362 worlds, 6 VMs, 12 vCPUs; CPU load average: 0.02, 0.02, 0.02

DEVICE                                CLONE RD CLONE WR  CLONE F MBC RD/s MBC WR/s      ATS ATSF     ZERO   ZERO F MBZERO/s   DELETE DELETE F  MBDEL/s CAVG/suc   CAVG/f
mpx.vmhba32:C0:T0:L0                         0        0        0     0.00     0.00        0    0        0        0     0.00        0        0     0.00     0.00     0.00
naa.500056b3ad89cffd                         0        0        0     0.00     0.00        0    0        0        0     0.00        0        0     0.00     0.00     0.00
naa.5001e820026415f0                         0        0        0     0.00     0.00        0    0        0        0     0.00        0        0     0.00     0.00     0.00
naa.500a07510f86d685                         0        0        0     0.00     0.00        0    0        0        0     0.00        0        0     0.00     0.00     0.00
naa.500a07510f86d6bb                         0        0        0     0.00     0.00        0    0        0        0     0.00        0        0     0.00     0.00     0.00
naa.55cd2e404c31fa00                         0        0        0     0.00     0.00        0    0        0        0     0.00        0        0     0.00     0.00     0.00
naa.624a9370d4d78052ea564a7e00011014         0        0        0     0.00     0.00        0    0        0        0     0.00        0        0     0.00     0.00     0.00
naa.624a9370d4d78052ea564a7e00011015         0        0        0     0.00     0.00        0    0        0        0     0.00        0        0     0.00     0.00     0.00
naa.624a9370d4d78052ea564a7e00011028         0        0        0     0.00     0.00        0    0        0        0     0.00        0        0     0.00     0.00     0.00
naa.624a9370d4d78052ea564a7e00011029         0        0        0     0.00     0.00        0    0        0        0     0.00        0        0     0.00     0.00     0.00
naa.624a9370d4d78052ea564a7e0001102a         0        0        0     0.00     0.00        0    0        0        0     0.00        0        0     0.00     0.00     0.00
naa.624a9370d4d78052ea564a7e0001102b         0        0        0     0.00     0.00        0    0        0        0     0.00        0        0     0.00     0.00     0.00
naa.624a9370d4d78052ea564a7e0001102c         0        0        0     0.00     0.00        0    0        0        0     0.00        0        0     0.00     0.00     0.00
naa.624a9370d4d78052ea564a7e0001102d         0        0        0     0.00     0.00        0    0        0        0     0.00        0        0     0.00     0.00     0.00
naa.624a9370d4d78052ea564a7e0001102e         0        0        0     0.00     0.00        0    0        0        0     0.00        0        0     0.00     0.00     0.00
naa.624a9370d4d78052ea564a7e0001102f         0        0        0     0.00     0.00   384712   13        0        0     0.00        0        0     0.00     0.00     0.00
naa.624a9370d4d78052ea564a7e00011030         0        0        0     0.00     0.00   381377    0        0        0     0.00        0        0     0.00     0.00     0.00
naa.624a9370d4d78052ea564a7e00011138         0        0        0     0.00     0.00  1211870    0     1036        0     0.00        0        0     0.00     0.00     0.00
naa.624a9370d4d78052ea564a7e00011139      8105     8105        0     0.00     0.00  1224528 2759   514665        0     0.00     3980        0     0.00     0.51     0.00
```

Figure 3. esxtop Output – Block Primitives

To see this output in *esxtop* , first select **u** for device view, and then **f** to change which fields are displayed.
Options **o** and **p** display VAAI statistics. Each primitive also has a column for failure statistics, that is, CLONE_F, ATSF and ZERO_F. These should be monitored for any failed operations.

One side effect is that VAAI offloads can lead to distorted kernel average latencies (KAVGs). When VAAI commands are issued via the VAAI filter, there actually are two commands sent. These are top-layer commands that are issued but never sent to the actual device (they stay within the VMkernel). These commands are intercepted by the VAAI filter and the VAAI plug-in and are replaced by the vendor-specific commands, which are issued to the device.

For this reason, *esxtop* shows device statistics for only the top-level commands. As a result, the values for latencies seem unusual when compared to results obtained when VAAI is not enabled.



```
 8:41:44am up 12:01, 359 worlds, 2 VMs, 1 vCPUs; CPU load average: 0.01, 0.00, 0.00

DEVICE                                PATH/WORLD/PARTITION DQLEN WQLEN ACTV QUED %USD  LOAD   CMDS/s  READS/s WRITES/s MBREAD/s MBWRTN/s DAVG/cmd KAVG/cmd GAVG/cmd QAVG/cmd
mpx.vmhba32:C0:T0:L0                                     -     1     -    0    0    0  0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
naa.64ed2a45bae9104c740425740d02a062                     -   128     -    0    0    0  0.00    18.30     0.48    17.82     0.00     0.05     0.35     0.01     0.36     0.00
naa.64ed2a45bae9104e740445740d02d051                     -    32     -    4    0   12  0.12    35.15     0.48     0.48     0.00     0.00     0.02   173.74   173.76     0.00
naa.64ed2a45ea091975940055141300909a                     -   128     -    0    0    0  0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
naa.64ed2a45ea09490994000514130040b6                     -   128     -    0    0    0  0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
naa.64ed2a45ea0959a0960095141300b01f                     -   128     -    0    0    0  0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
naa.64ed2a45ea09f97294002514130d0ff                      -   128     -    0    0    0  0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
naa.64ed2af559570f27e000953e1f02e070                     -   128     -    0    0    0  0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
naa.64ed2af559570f58e000b5022302b004                     -   128     -    0    0    0  0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
naa.64ed2af559570f58e000d50223029009                     -   128     -    0    0    0  0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
naa.64ed2af559571f91e00015832302d0c9                     -   128     -    0    0    0  0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
naa.64ed2af559571ff5c70065e01202e072                     -   128     -    0    0    0  0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
naa.64ed2af559572f91e00035032302000 6                    -   128     -    0    0    0  0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
naa.64ed2af559573f00e00035a115027000                     -   128     -    0    0    0  0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
```

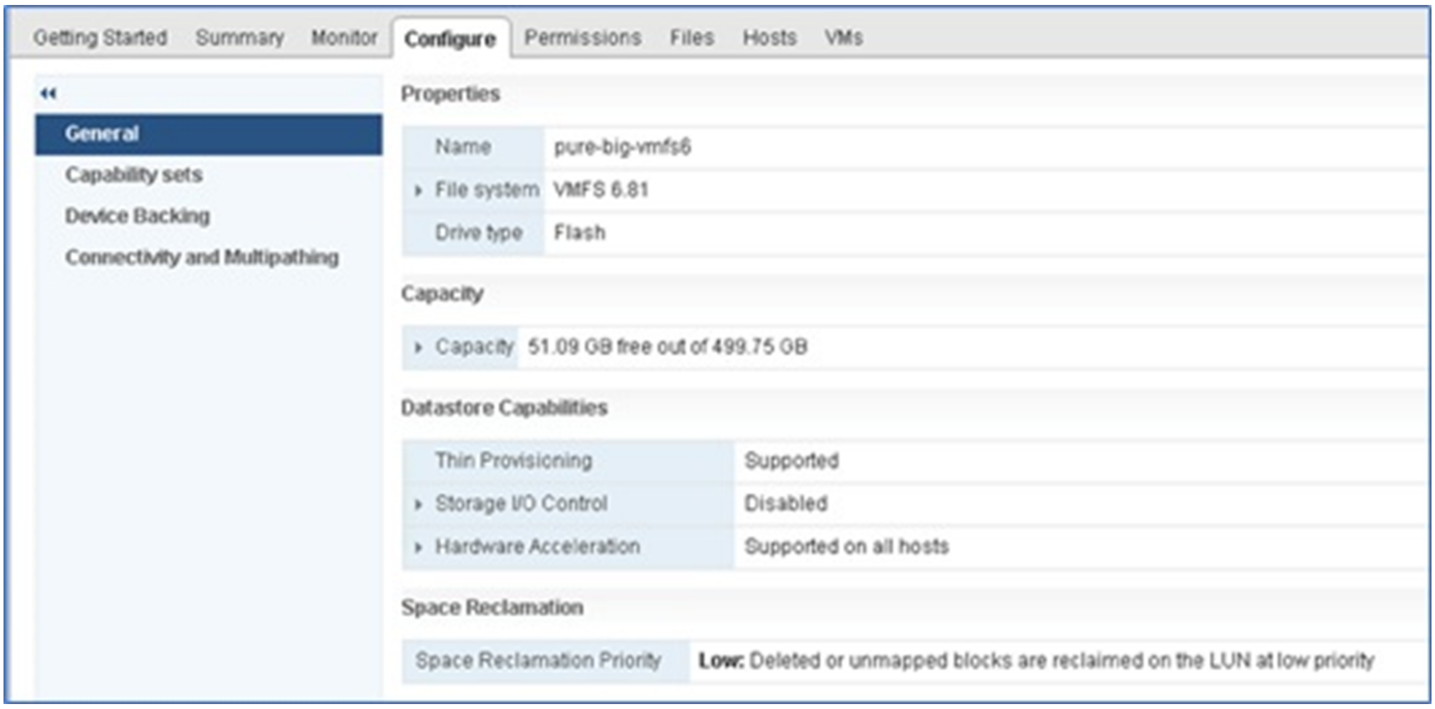
Figure 4. Esxtop – latency values

In this—and only this—instance, the latencies observed in *esxtop* should not be interpreted as a performance issue, unless there are other symptoms present.

Checking VAAI Support

This section describes the process that is followed to check the VAAI Status.

To check VAAI status, a number of esxcli commands are available. In a preceding output, the ATS Status, Clone Status (XCOPY), Zero Status (WRITE_SAME) and Delete Status (UNMAP) are shown as supported. The vSphere UI also can be used to verify VAAI support. The overall VAAI status is displayed as Supported, Not supported or Unknown under Hardware Acceleration in the UI, as is shown in the following:



Properties

Name	pure-big-vmfs6
File system	VMFS 6.81
Drive type	Flash

Capacity

Capacity	51.09 GB free out of 499.75 GB

Datastore Capabilities

Thin Provisioning	Supported
Storage I/O Control	Disabled
Hardware Acceleration	Supported on all hosts

Space Reclamation

Space Reclamation Priority	**Low:** Deleted or unmapped blocks are reclaimed on the LUN at low priority

Figure 5.

VAAI Status

Hardware Acceleration status is determined by the support state of various primitives. If ATS is supported, the UI displays Hardware Acceleration as

Supported. If, however, ATS, XCOPY and Zero all are unsupported, Hardware

Acceleration displays as Unsupported. All other support states set the Hardware Acceleration status to Unknown, which typically is a state that is displayed until the first clone and zero operation is initiated. Many users initiate an eagerzeroedthick clone of a file to test the VAAI status.

In vSphere 6.5, VMFS6 datastore will also display a Space Reclamation field.

This is indicating whether or not Automatic UNMAP is available. In vSphere 6.5, the only Space Reclamation priority available is Low . This cannot be changed.

Enabling and Disabling VAAI Primitive

This section focuses on enabling and disabling the different types of VAAI Primitives.

To disable some of the VAAI block primitives—for troubleshooting purposes, for example—see the following CLI commands, which detail how to enable and disable VAAI primitives.

ATS

To check the status of the ATS primitive and to turn it on and off at the command line, the following commands can be used:

```
# esxcli system settings advanced list --option /VMFS3/HardwareAcceleratedLocking
   Path: /VMFS3/HardwareAcceleratedLocking
   Type: integer
   Int Value: 1
   Default Int Value: 1
   Min Value: 0
   Max Value: 1
   String Value:
   Default String Value:
   Valid Characters:
   Description: Enable hardware accelerated VMFS locking (requires compliant hardware). Please see
http://kb.vmware.com/kb/2094604 before disabling this option.
# esxcli system settings advanced set -i 0 -o /VMFS3/HardwareAcceleratedLocking
# esxcli system settings advanced set -i 1 -o /VMFS3/HardwareAcceleratedLocking
```

XCOPY

To turn the Extended Copy primitive for cloning on and off, use the previous command with the following advanced setting:

```
# esxcli system settings advanced list --option /DataMover/HardwareAcceleratedMove
   Path: /DataMover/HardwareAcceleratedMove
   Type: integer
   Int Value: 1
   Default Int Value: 1
   Min Value: 0
   Max Value: 1
   String Value:
   Default String Value:
   Valid Characters:
   Description: Enable hardware accelerated VMFS data movement (requires compliant hardware)
```

WRITE_SAME

To turn the Write Same primitive for zeroing blocks on and off, use the following advanced setting:

```
# esxcli system settings advanced list --option /DataMover/HardwareAcceleratedInit
   Path: /DataMover/HardwareAcceleratedInit
   Type: integer
   Int Value: 1
   Default Int Value: 1
   Min Value: 0
   Max Value: 1
   String Value:
   Default String Value:
   Valid Characters:
    Description: Enable hardware accelerated VMFS data initialization (requires compliant hardware)
```

UNMAP

Prior to vSphere 6.5 and earlier versions of VMFS, to turn the UNMAP primitive for space reclamation on and off, use the following advanced setting:

```
# esxcli system settings advanced list --option /VMFS3/EnableBlockDelete
   Path: /VMFS3/EnableBlockDelete
   Type: integer
   Int Value: 0
   Default Int Value: 0
   Min Value: 0
   Max Value: 1
   String Value:
   Default String Value:
   Valid Characters:
   Description: Enable VMFS block delete when UNMAP is issued from guest OS
```

For VMFS6 on vSphere 6.5, there is a different advanced setting:

```
# esxcli system settings advanced list --option /VMFS3/EnableVMFS6Unmap
   Path: /VMFS3/EnableVMFS6Unmap
   Type: integer
   Int Value: 1
   Default Int Value: 1
   Min Value: 0
   Max Value: 1
   String Value:
   Default String Value:
   Valid Characters:
   Description: Enable VMFS6 unmap feature
```

In vSphere 4.1, it was possible to define how often an ESXi host verified whether hardware acceleration was supported on the storage array. This means that if at initial deployment, an array does not support the offload primitives, but at a later date the firmware on the arrays is upgraded and the offload primitives become supported, nothing must be done regarding ESXi—it automatically will start to use the offload primitives.

```
# esxcfg-advcfg -g /DataMover/HardwareAcceleratedMoveFrequency
Value of HardwareAcceleratedMoveFrequency is 16384
```

The value relates to the number of I/O requests that occur before another offload is attempted. With an I/O size of 32MB, 512GB of I/O must flow before a VAAI primitive is retried.

The same is true for an offload failure. If the array returns a failure status for an offload operation, the software Data Mover is used to continue the operation. After another 16,384 I/O requests using the software Data Mover, the VAAI offload is once again attempted.

HardwareAcceleratedMoveFrequency exists only in vSphere 4.1. In later versions of vSphere, the parameter was replaced with a periodic VAAI state evaluation that runs every 5 minutes.

Reverting to the Software Data Mover

Before VAAI, migrations of virtual machines, and their associated disks, between datastores was done by a component called the software Data Mover, a loadable VMkernel module accessible via special library calls. A data move request, in the form of a data structure, is submitted to the Data Mover for asynchronous completion.

The Data Mover takes this data structure and divides it into smaller I/O requests that it sends to the appropriate back end. The Data Mover relies on a constantly filled queue of outstanding I/O requests to achieve maximum throughput. Incoming Data Mover requests are broken down into smaller chunks, and asynchronous I/Os are simultaneously issued for each chunk until the configured Data Mover queue depth is filled. Then the asynchronous Data Mover request returns to the caller who would normally proceed with what it was doing or wait for I/O completion. Meanwhile, as and when an asynchronous request completes, the next logical request for the overall Data Mover task is issued, whether it is for writing the data that was just read or to handle the next chunk.

In the following example, a clone of a 64GB VMDK file is initiated. The Data Mover is asked to move data in terms of 32MB transfers. The 32MB is then transferred in “PARALLEL” as a single delivery but is divided up into a much smaller I/O size of 64KB, using 32 work items that are all queued in parallel (because they execute asynchronously). To transfer this 32MB, a total of 512 I/Os of size 64KB are issued by the Data Mover. By comparison, a similar 32MB transfer via VAAI uses eight work items, because XCOPY uses 4MB transfer sizes. VMkernel can choose from the following three Data Mover types:

- **fsdm** – This is the legacy Data Mover, the most basic version. It is the slowest because the data moves all the way up the stack and then down again.
- **fs3dm** (software) – This is the software Data Mover, which was introduced with vSphere 4.0 and contained some substantial optimizations whereby data does not travel through all stacks.
- **fs3dm** (hardware) – This is the hardware offload Data Mover, introduced with vSphere 4.1. It still is the fs3dm, but VAAI hardware offload is leveraged with this version. fs3dm is used in software mode when hardware offload (VAAI) capabilities are not available.

The following diagram shows the various paths taken by the various Data Movers:

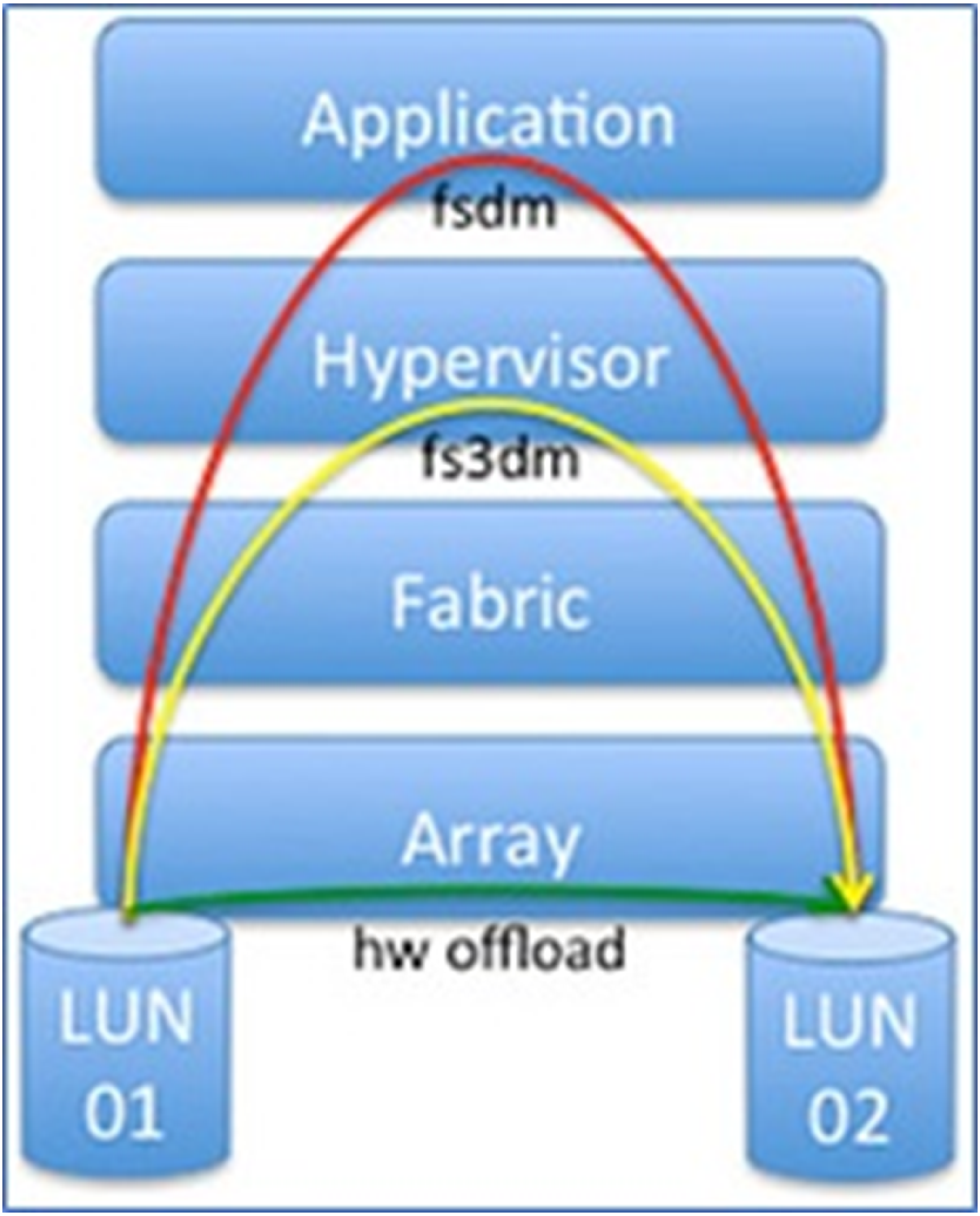


Figure 6. Data Mover Types

The decision to transfer using the software Data Mover or using hardware offloading to the array with VAAI is taken upfront by looking at storage array hardware acceleration state. If we decide to transfer using VAAI and then FAIL—a scenario called degraded mode—the VMkernel will try to accomplish the transfer using the software Data Mover. It should be noted that the operation is not restarted; rather it picks up from where the previous transfer left off as we do not want to abandon what could possibly be very many gigabytes of copied data because of a single transient transfer error. When reverting to the software Data Mover, we cannot operate in terms of the XCOPY transfer size of 4MB, due to memory buffer requirement, so the transfer size is reduced to 64KB.

Differences in VAAI between vSphere versions

This section points out the differences between the two phases- The first phase/release of VAAI in vSphere 4.1 and the second phase/ release of VAAI in vSphere 5.0.

There are a number of differences between the first phase/release of VAAI in vSphere 4.1 and the second phase of VAAI, which was released with vSphere 5.0. The following is a list of those differences:

- VAAI in 5.0 now uses standard T10 primitives rather than per-array manufacturer commands.
- There is now full ATS support for all metadata operations with VMFS5.
- vSphere 5.0 introduced support for NAS primitives, including VCAI.
- vSphere 5.0 introduced support for Thin Provisioning primitives, including UNMAP.
- VMware HCL now requires performance of primitives before an array receives VAAI certification. This is important because certain storage arrays that appear in the vSphere 4.1 HCL might not appear in the vSphere 5.0 HCL if the performance of the offload primitives does not meet requirements.
- vSphere 6.5 introduced Automatic-UNMAP for VMFS6
- VMware HCL confirms if a storage array can support Automatic UNMAP feature

VAAI Caveats

This section focuses on Data Mover and how it is used for different types of VAAI Caveats.

In cases involving the following known caveats related to VAAI, hardware offloads to the array will not be leveraged; the Data Mover will be used instead:

VMFS Block Sizes

If the source and destination VMFS volumes have different block sizes, data movement will revert to the generic FSDM layer, which will move only software data.

Raw Device Mappings

If the source file type is raw device mapping (RDM) and the destination file type is non-RDM (regular file), migration occurs via the Data Mover.

Sparse/Hosted Format VMDK

If either the source or destination VMDK involved in the operation is of sparse or hosted format, the Data Mover is used

Misalignment

The software Data Mover is used if the logical address and/or transfer length in the requested operation is not aligned to the minimum alignment required by the storage device.

If the VMFS datastores are created using vSphere Client, it is likely that all requests will be aligned. If a LUN is incorrectly partitioned using low-level tools such as *fdisk* or *partedUtil* , the operation might revert to using the software Data Mover.

Because customers typically use vSphere Client and vCenter exclusively for a created datastore, it is unlikely that this issue would be encountered. If a customer’s environment was migrated from ESX 2.x/VMFS2 in the distant past, however, the volumes might be misaligned. This can be an issue on some VAAI arrays that have a high alignment requirement.

Automatic UNMAP also relies on alignment, either on VMFS6 on in-guest. Customers are advised to check the footnote in the VMware HCL to confirm whether or not their storage array supports automatic UNMAP.

VMFS Extents & ATS

VMware supports VAAI primitives on VMFS with multiple LUNs/extents, if they all are on the same array and the array supports offloading. There are some limitations on how ATS can be used on multi-extent VMFS volumes, but that impact is relatively low.

On VAAI Hardware	New VMFS5/6	Upgrade VMFS5	VMFS3
Single-extent datastore	ATS only	ATS, but reverts to SCSI2 reservations	ATS, but reverts to SCSI2 reservations
Multi-extent datastore	Allows spanning only on ATS hardware	ATS, except when locks on non-head	ATS, except when locks on non-head

Table 1. ATS Limitations on Multi-extent VMFS Volumes

Different Destination Storage Array

VMware does not support VAAI primitives on VMFS with multiple LUNs/extents if they all are on different arrays, even if all arrays support offloading. Hardware cloning between arrays—even if it is within the same VMFS volume—will not work, so that would revert to software data movement.

Acknowledgements

This section acknowledges the contributors who have helped the author.

Thank you to Paudie O’Riordan, Duncan Epping for reviewing the contents of this paper.

About the Author

This section introduces the author of the guide.

Cormac Hogan is a Director and Chief Technologist in the Storage and Availability Business Unit at VMware. Cormac has written a book called “Essential vSAN” available from VMware Press, as well as a number of storage-related papers. He has given numerous presentations on storage best practices and new features and is a regular presenter at VMworld and VMware User Group meetings.