VMware Storage APIs for Array Integration with the Pure Storage FlashArray

August 2015

Contents

Executive Summary

This document describes the purpose and performance characterizations of the VMware Storage APIs for Array Integration (VAAI) with the Pure Storage FlashArray. The Pure Storage FlashArray includes general support for VMware ESXi as well as the most important VAAI primitives that enable administrators to enhance and simplify the operation and management of VMware vSphere virtualized environments. Throughout this paper, specific best practices on using VAAI with Pure Storage will be discussed.

Goals and Objectives

This paper is intended to provide insight for the reader into how VMware VAAI behaves when utilized on the Pure Storage FlashArray. This includes why a feature is important, any best practices, expected operational behavior, and performance.

Audience

This document is intended for use by pre-sales consulting engineers, sales engineers and customers who want to deploy the Pure Storage FlashArray in VMware vSphere-based virtualized datacenters.

Pure Storage Introduction

Pure Storage is the leading all-flash enterprise array vendor, committed to enabling companies of all sizes to transform their businesses with flash.

Built on 100% consumer-grade MLC flash, Pure Storage FlashArray delivers all-flash enterprise storage that is 10X faster, more space and power efficient, more reliable, and infinitely simpler, and yet typically costs less than traditional performance disk arrays.

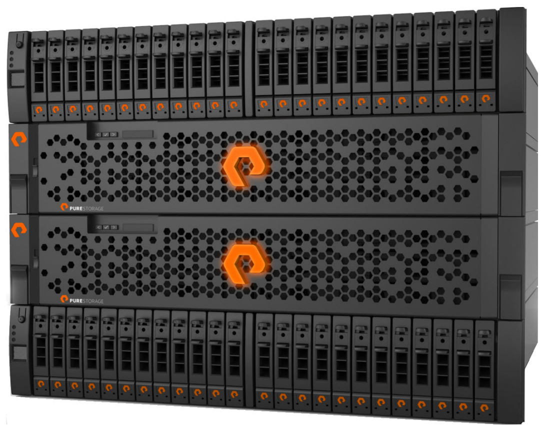


Figure 1. Pure Storage FlashArray 400 Series

Figure 2. Pure Storage FlashArray//m

The Pure Storage FlashArray is ideal for:

Accelerating Databases and Applications Speed transactions by 10x with consistent low latency, enable online data analytics across wide datasets, and mix production, analytics, dev/test, and backup workloads without fear.

Virtualizing and Consolidating Workloads Easily accommodate the most IO-hungry Tier 1 workloads, increase consolidation rates (thereby reducing servers), simplify VI administration, and accelerate common administrative tasks.

Delivering the Ultimate Virtual Desktop Experience Support demanding users with better performance than physical desktops, scale without disruption from pilot to >1000's of users, and experience all-flash performance for under $100/desktop.

Protecting and Recovering Vital Data Assets Provide an always-on protection for business-critical data, maintain performance even under failure conditions, and recover instantly with FlashRecover.

Pure Storage FlashArray sets the benchmark for all-flash enterprise storage arrays. It delivers:

Consistent Performance FlashArray delivers consistent <1ms average latency. Performance is optimized for the real-world applications workloads that are dominated by I/O sizes of 32K or larger vs. 4K/8K hero performance benchmarks. Full performance is maintained even under failures/updates.

Less Cost than Disk Inline de-duplication and compression deliver 5 – 10x space savings across a broad set of I/O workloads including Databases, Virtual Machines and Virtual Desktop Infrastructure.

Mission-Critical Resiliency FlashArray delivers >99.999% proven availability, as measured across the Pure Storage installed base and does so with non-disruptive everything without performance impact.

Disaster Recovery Built-In FlashArray offers native, fully-integrated, data reduction-optimized backup and disaster recovery at no additional cost. Setup disaster recovery with policy-based automation within minutes. And, recover instantly from local, space-efficient snapshots or remote replicas.

Simplicity Built-In FlashArray offers game-changing management simplicity that makes storage installation, configuration, provisioning and migration a snap. No more managing performance, RAID, tiers or caching. Achieve optimal application performance without any tuning at any layer. Manage the FlashArray the way you like it: Web-based GUI, CLI, VMware vCenter, Rest API, or OpenStack.

FlashArray scales from smaller workloads to data center-wide consolidation. And because upgrading performance and capacity on the FlashArray is always non-disruptive, you can start small and grow without impacting mission-critical applications. Coupled with Forever Flash, a new business model for storage acquisition and lifecycles, FlashArray provides a simple and economical approach to evolutionary storage that extends the useful life of an array and does away with the incumbent storage vendor practices of forklift upgrades and maintenance extortion.

VAAI Best Practices Checklist

The following section is intended as a quick-start guide for using VAAI functionality on the Pure Storage FlashArray. Refer to the relevant sections in the rest of the document for more information.

Acknowledged/Done?	Description
☐	Ensure proper multipathing configuration is complete. This means more than one HBA and connections to at least four FlashArray ports (two on each controller). All Pure Storage devices should be controlled by the VMware Native Multipathing Plugin (NMP) Round Robin Path Selection Policy (PSP). Furthermore, it is recommended that each device be configured to use an I/O Operation Limit of 1.
☐	Ensure that all VAAI primitives are enabled.
☐	For XCOPY, set the maximum transfer size (DataMover.MaxHWTransferSize) to 16 MB.
☐	When running UNMAP in ESXi 5.5 and later, use a block count that is equal to or less than 1% of the free capacity on the target VMFS. A PowerCLI script to execute UNMAP can be found here.
☐	WRITE SAME and ATS have no specific recommendations.
☐	In ESXi 6.0+, set the EnableBlockDelete option to "enabled". This is the only primitive not enabled by default.

! Pure Storage does not support DISABLING VAAI features on ESXi hosts—if you must to disable it for some reason please must contact Pure Storage support if the affected ESXi hosts also have access to Pure Storage FlashArray devices.

Introduction to VAAI

The VMware Storage APIs for Array Integration (VAAI) is a feature set first introduced in vSphere 4.1 that accelerates common tasks by offloading certain storage-related operations to compatible arrays. With storage hardware assistance, an ESXi host can perform these operations faster and more efficiently while consuming far less CPU, memory, and storage fabric bandwidth. This behavior allows for far greater scale, consolidation and flexibility in a VMware-virtualized infrastructure. VAAI primitives are enabled by default and will automatically be invoked if ESXi detects that there is support from the underlying storage.

The Pure Storage FlashArray supports VAAI in ESXi 5.0 and later. The following five primitives are available for block-storage hardware vendors to implement and support:

- **Hardware Assisted Locking**—commonly referred to as Atomic Test & Set (ATS), this uses the SCSI command COMPARE and WRITE (0x89), which is invoked to replace legacy SCSI reservations during the creation, alteration and deletion of files and metadata on a VMFS volume.
- **Full Copy**—leverages the SCSI command XCOPY (0x83), which is used to copy or move virtual disks.
- **Block Zero**—leverages the SCSI command WRITE SAME (0x93) which is used to zero-out disk regions during virtual disk block allocation operations.
- **Dead Space Reclamation**—leverages the SCSI command UNMAP (0x42) to reclaim previously used but now deleted space on a block SCSI device.
- **Thin Provisioning Stun and Resume**[1]—allows for underlying storage to inform ESXi that capacity has been entirely consumed which causes ESXi to immediately "pause" virtual machines until additional capacity can be provisioned/installed.

Pure Storage FlashArray supports ATS, XCOPY, WRITE SAME and UNMAP in Purity release 3.0.0 onwards on ESXi 5.x and 6.x.

[1] Thin Provisioning Stun & Resume is not currently supported by the Pure Storage FlashArray.

Enabling VAAI

In order to determine if VAAI is enabled on an ESXi host use the esxcli command or the vSphere Web Client to check if the value of the line "`Int Value`" is set to 1 (enabled):

All vSphere versions:

```
esxcli system settings advanced list -o /DataMover/HardwareAcceleratedMove
esxcli system settings advanced list -o /DataMover/HardwareAcceleratedInit
esxcli system settings advanced list -o /VMFS3/HardwareAcceleratedLocking
```

vSphere 5.5 U2 and later:

```
esxcli system settings advanced list -o /VMFS3/useATSForHBOnVMFS5
```

vSphere 6.0 and later:

```
esxcli system settings advanced list -o /VMFS3/EnableBlockDelete
```
[2]

You will see an output similar to:

```
   Path: /VMFS3/HardwareAcceleratedLocking
   Type: integer
   Int Value: 1            ← Value is 1 if enabled
   Default Int Value: 1
   Min Value: 0
   Max Value: 1
   String Value:
   Default String Value:
   Valid Characters:
   Description: Enable hardware accelerated VMFS locking
```

Hardware acceleration is enabled by default within ESXi (except EnableBlockDelete) and all options never require any configuration on the array to use out of the box. In the case they were somehow disabled in ESXi, follow these steps to re-enable the primitives:

1. To enable atomic test and set (ATS) AKA hardware accelerated locking:

```
esxcli system settings advanced set -i 1 -o
/VMFS3/HardwareAcceleratedLocking
```

2. To enable hardware accelerated initialization AKA WRITE SAME:

```
esxcli system settings advanced set --int-value 1 --option
/DataMover/HardwareAcceleratedInit
```

[2] This is the only VAAI option of the five that is not enabled by default in ESXi.

3. To enable hardware accelerated move AKA XCOPY (full copy):

```
esxcli system settings advanced set --int-value 1 --option
/DataMover/HardwareAcceleratedMove
```

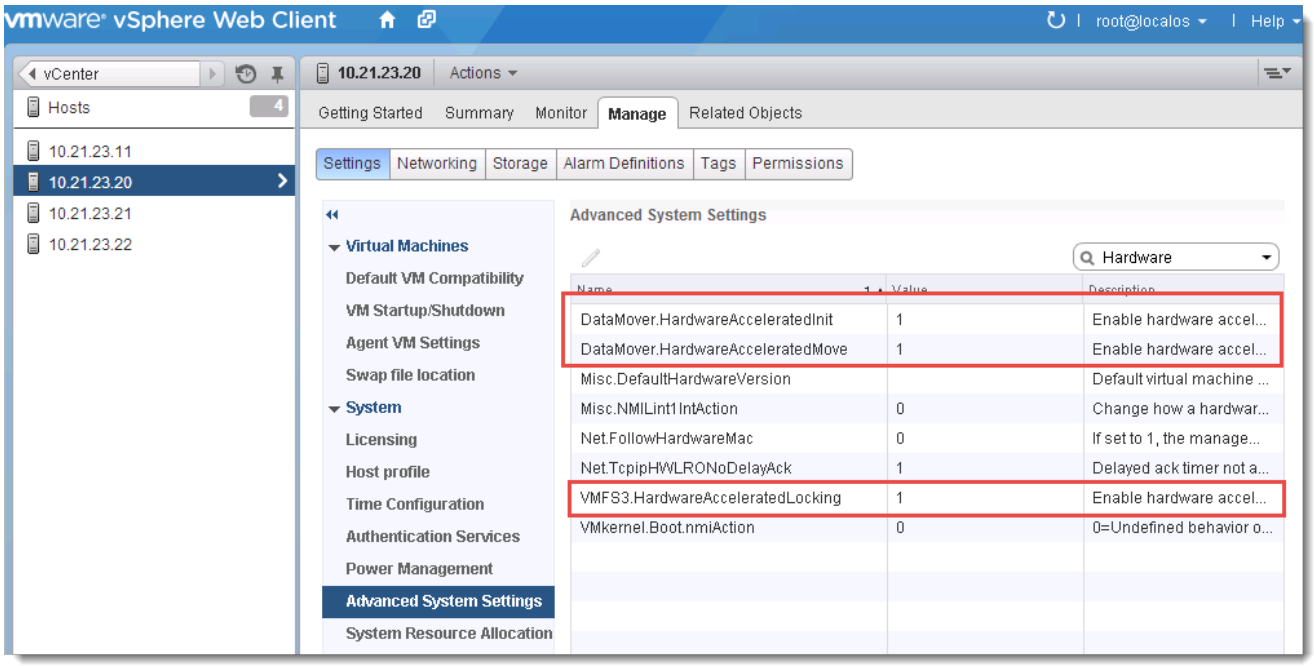
4. In VMware ESXi 5.5 U2 and later, VMware introduced using ATS for VMFS heartbeats. To enable this setting:

```
esxcli system settings advanced set -i 1 -o /VMFS3/useATSForHBOnVMFS5
```

5. To enable guest OS UNMAP in vSphere 6.x only:

```
esxcli system settings advanced set -i 1 -o /VMFS3/EnableBlockDelete
```

The figure below describes the above steps pictorially using the vSphere Web Client. Go to an ESXi host and then Settings, then Advanced System Settings and search for "Hardware" or "EnableBlockDelete", as the case may be, to find the settings[3].

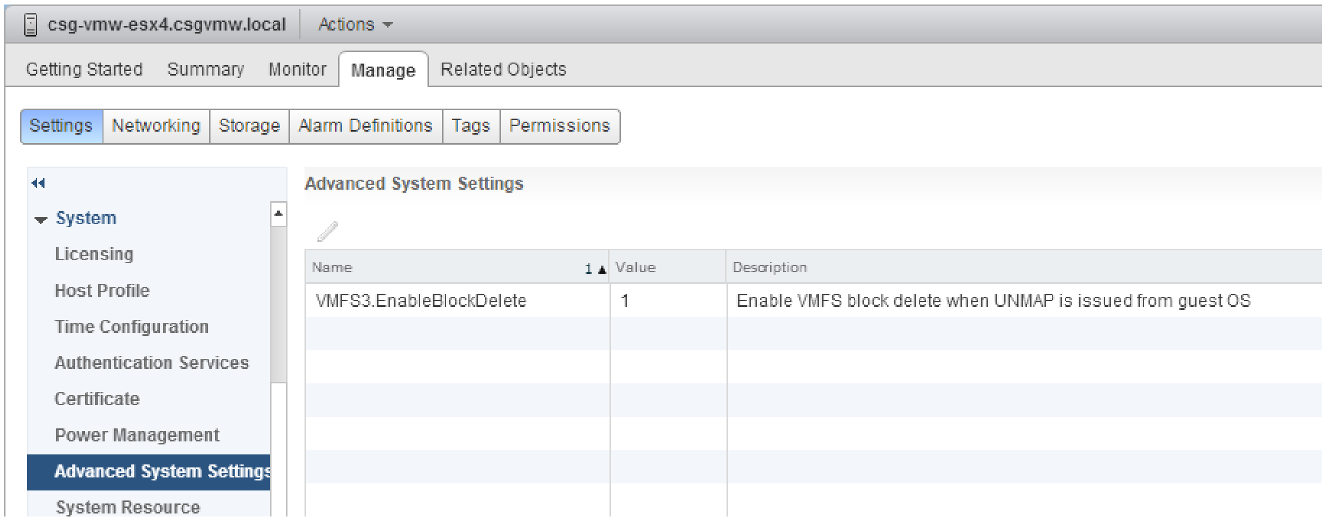


Advanced System Settings

Name	Value	Description
DataMover.HardwareAcceleratedInit	1	Enable hardware accel...
DataMover.HardwareAcceleratedMove	1	Enable hardware accel...
Misc.DefaultHardwareVersion		Default virtual machine ...
Misc.NMILint1IntAction	0	Change how a hardwar...
Net.FollowHardwareMac	0	If set to 1, the manage...
Net.TcpipHWLRONoDelayAck	1	Delayed ack timer not a...
VMFS3.HardwareAcceleratedLocking	1	Enable hardware accel...
VMkernel.Boot.nmiAction	0	0=Undefined behavior o...

Figure 4. VAAI advanced options in the vSphere Web Client

[3] The option useATSForHBOnVMFS5 is not currently available in the GUI and can only be set in the CLI.



Name	Value	Description
VMFS3.EnableBlockDelete	1	Enable VMFS block delete when UNMAP is issued from guest OS

Figure 5. Enable Guest OS UNMAP in vSphere 6.x

Disabling Hardware Assisted Locking on Incompatible Arrays

Pure Storage does NOT support disabling hardware assisted locking on FlashArray devices. The aforementioned setting is a host-wide setting and there may be situations when arrays that are incompatible with hardware assisted locking are present. Leaving hardware assisted locking enabled on the host may cause issues with that array. Therefore, instead of disabling hardware assisted locking host-wide, disable hardware assisted locking for the legacy/incompatible array on a per-device basis. For instructions, refer to the following VMware KB article:

http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=2006858

Hardware Assisted Locking or Atomic Test & Set

Prior to the introduction of VAAI Atomic Test & Set (ATS), ESXi hosts used device-level locking via full SCSI reservations to get and control access to the metadata associated with a VMFS volume. In a cluster with multiple hosts, all metadata operations for a given volume were serialized and I/O from other hosts had to wait until whichever host currently holding the lock released it. This behavior not only caused metadata lock queues, which slowed down operations like virtual machine provisioning, but also delayed any standard I/O to a volume from ESXi hosts not currently holding the lock.

With VAAI ATS, the locking granularity is reduced to a much smaller level of control by only locking specific metadata segments, instead of an entire volume. This behavior makes the metadata change process not only very efficient, but importantly provides a mechanism for parallel metadata access while still maintaining data integrity. ATS allows for ESXi hosts to no longer have to queue metadata change requests, which accordingly accelerates operations that previously had to wait for a lock to release. Therefore, situations with large amounts of simultaneous virtual machine provisioning/configuration operations will see the most benefit. The standard use cases benefiting the most from ATS include:

- Large number of virtual machines on a single datastore (100s+).
- Extremely dynamic environments—numerous provisioning and de-provisioning of VMs.
- Virtual Desktop Infrastructure (VDI) common bulk operations such as boot storms.

Unlike some of the other VAAI primitives, the benefits of hardware assisted locking are not always readily apparent in day to day operations. That being said, there are some situations where the benefit arising from the enablement of hardware assisted locking can be somewhat profound. For example, see the following case.

Hardware assisted locking provides the most notable assistance in situations where traditionally there would be an exceptional amount of SCSI reservations over a short period of time. The most standard example of this would be a mass power-on of a large number of virtual machines, commonly known as a boot storm. During a boot storm, the host or hosts booting up the virtual machines require at least an equivalent number of locks to the target datastore(s) of the virtual machines. These volume-level locks cause other workloads to have reduced and unpredictable performance for the duration of the boot storm. Refer to the following charts that show throughput and IOPS of a workload running during a boot storm with hardware accelerated locking enabled and disabled.

In this scenario, a virtual machine ran a workload across five virtual disks residing on the same datastore as 150 virtual machines that were all powered-on simultaneously. By referring to the previous charts, it's clear that with hardware assisted locking disabled the workload is deeply disrupted, resulting in inconsistent and inferior performance during the boot storm. Both the IOPS and throughput[4] vary wildly throughout the test. When hardware assisted locking is enabled the disruption is almost entirely gone and the workload proceeds essentially unfettered.

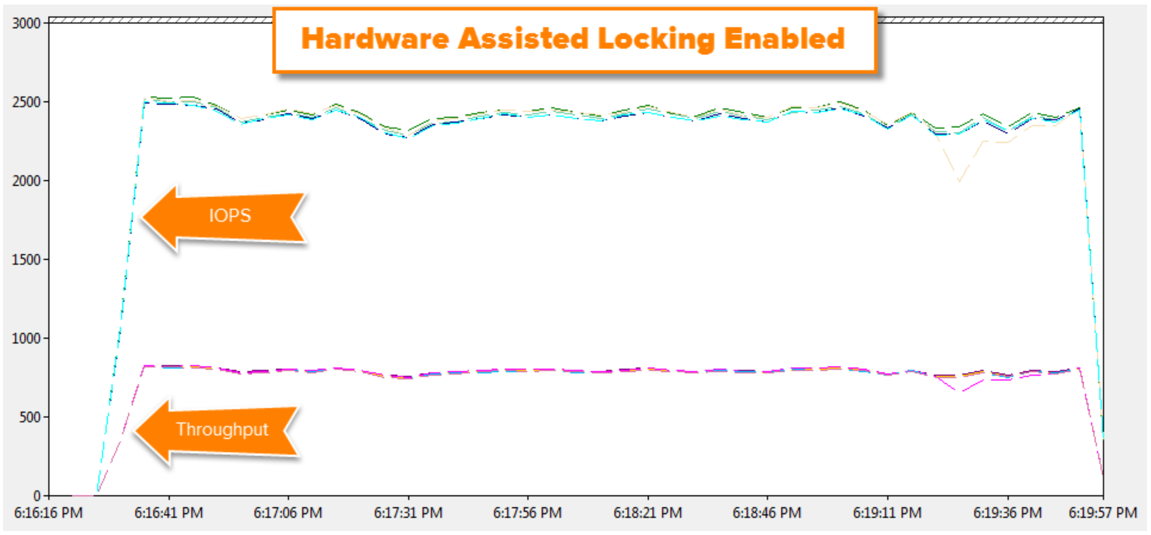


Figure 6. Performance test with hardware assisted locking enabled

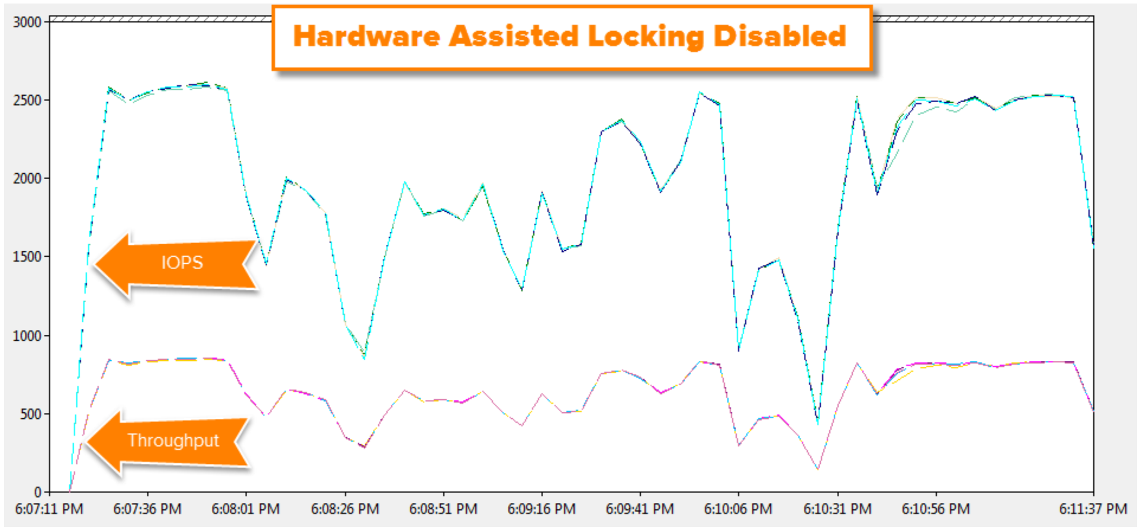


Figure 7. Performance test with hardware assisted locking disabled

[4] The scale for throughput is in MB/s but is reduced in scale by a factor of ten to allow it to fit in a readable fashion on the chart with the IOPS values. So a throughput number on the chart of 1,000 is actually a throughput of 100 MB/s.

Full Copy or Hardware Accelerated Copy

Prior to Full Copy (XCOPY) API support, when data needed to be copied from one location to another such as with Storage vMotion or a virtual machine cloning operation, ESXi would issue a series of standard SCSI read/write commands between the source and target storage location (the same or different device). This resulted in a very intense and often lengthy additional workload to the source and target storage for the duration of the copy operation. This I/O consequently stole available bandwidth from more "important" I/O such as the I/O issued from virtualized applications. Therefore, copy or movement operations often had to be scheduled to occur only during non-peak hours in order to limit interference with normal production storage performance. This restriction effectively decreased the stated dynamic abilities and benefits offered by a virtualized infrastructure.

The introduction of XCOPY support for virtual machine data movement allows for this workload to be almost entirely offloaded from the virtualization stack onto the storage array. The ESXi kernel is no longer directly in the data copy path and the storage array instead does all the work. XCOPY functions by having the ESXi host identify a region that needs to be copied. ESXi then describes this space in a series of XCOPY SCSI commands and sends them to the array. The array then translates these block descriptors and copies the data at the described source location to the described target location entirely within the array. This architecture does not require any data to be sent back and forth between the host and array—the SAN fabric does not play a role in traversing the actual virtual machine data. The host only tells the array where the data that needs to be moved resides and where to move it to—it does not need to tell the array what the data actually is and subsequently has a profound effect on reducing the time to move data. XCOPY benefits are leveraged during the following operations[5]:

- Virtual machine cloning
- Storage vMotion or offline migration
- Deploying virtual machines from template

During these offloaded operations, the throughput required on the data path is greatly reduced as well as the load on the ESXi hardware resources (HBAs, CPUs etc.) initiating the request. This frees up resources for more important virtual machine operations by letting the ESXi resources do what they do best: host virtual machines, and lets the storage do what it does best: manage the storage.

On the Pure Storage FlashArray, XCOPY sessions are exceptionally quick and efficient. Due to the Purity FlashReduce technology (features like deduplication, pattern removal and compression) similar data is not stored on the FlashArray more than once. Therefore, during a host-initiated copy operation such as XCOPY, the FlashArray does not need to copy the data—this would be wasteful. Instead, Purity simply accepts and acknowledges the XCOPY requests and just creates new (or in the case of Storage vMotion, redirects

[5] Note that there are VMware-enforced caveats in certain situations that would prevent XCOPY behavior and revert to traditional software copy. Refer to VMware documentation for this information at www.vmware.com.

existing) metadata pointers. By not actually having to copy/move data the offload process duration is even faster. In effect, the XCOPY process is a 100% inline deduplicated operation.

A standard copy process for a virtual machine containing, for example, 50 GB of data can take many minutes or more. When XCOPY is enabled and properly configured, this time drops to a matter of a few seconds—usually around ten for a virtual machine of that size.

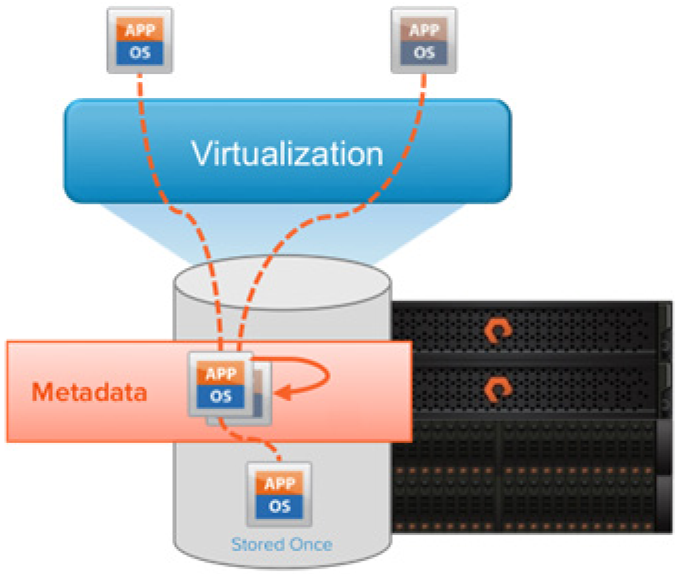


Figure 8. Pure Storage XCOPY implementation

XCOPY on the Pure Storage FlashArray works directly out of the box without any pre-configuration required. But, there is one simple configuration change on the ESXi hosts that can increase the speed of XCOPY operations. ESXi offers an advanced setting called the MaxHWTransferSize that controls the maximum amount of data space that a single XCOPY SCSI command can describe. The default value for this setting is 4 MB. This means that any given XCOPY SCSI command sent from that ESXi host cannot exceed 4 MB of described data.

The FlashArray, as previously noted, does not actually copy the data described in a XCOPY transaction—it just moves or copies metadata pointers. Therefore, for the most part, the bottleneck of any given virtual machine operation that leverages XCOPY is not the act of moving the data (since no data is moved), but it is instead a factor of how quickly an ESXi host can send XCOPY SCSI commands to the array. As a result, copy duration depends on the number of commands sent (dictated by both the size of the virtual machine and the maximum transfer size) and correct multipathing configuration.

Accordingly, if more data can be described in a given XCOPY command, less commands overall need to be sent and will subsequently take less time for the total operation to complete. For this reason Pure Storage recommends setting the transfer size to the maximum value of 16 MB[6].

The following commands can respectively be used to retrieve the current value and for setting a new one:

```
esxcfg-advcfg -g /DataMover/MaxHWTransferSize
```

```
esxcfg-advcfg -s 16384 /DataMover/MaxHWTransferSize
```

As mentioned earlier, general multipathing configuration best practices play a role in the speed of these operations. Changes like setting the Native Multipathing Plugin (NMP) Path Selection Plugin (PSP) for Pure devices to Round Robin and configuring the Round Robin IO Operations Limit to 1 can also provide an improvement in copy durations (offloaded or otherwise). Refer to the VMware and Pure Storage Best Practices Guide on www.purestorage.com for more information.

The following sections will outline a few examples of XCOPY usage to describe expected behavior and performance benefits with the Pure Storage FlashArray. Most tests will use the same virtual machine:

- Windows Server 2012 R2 64-bit
- 4 vCPUs, 8 GB Memory
- One zeroedthick 100 GB virtual disk containing 50 GB of data (in some tests the virtual disk type is different and/or size and this is noted where necessary)

If performance is far off from what is expected it is possible that the situation is not supported by VMware for XCOPY offloading and legacy software-based copy is being used instead. The following VMware restrictions apply and cause XCOPY to not be used:

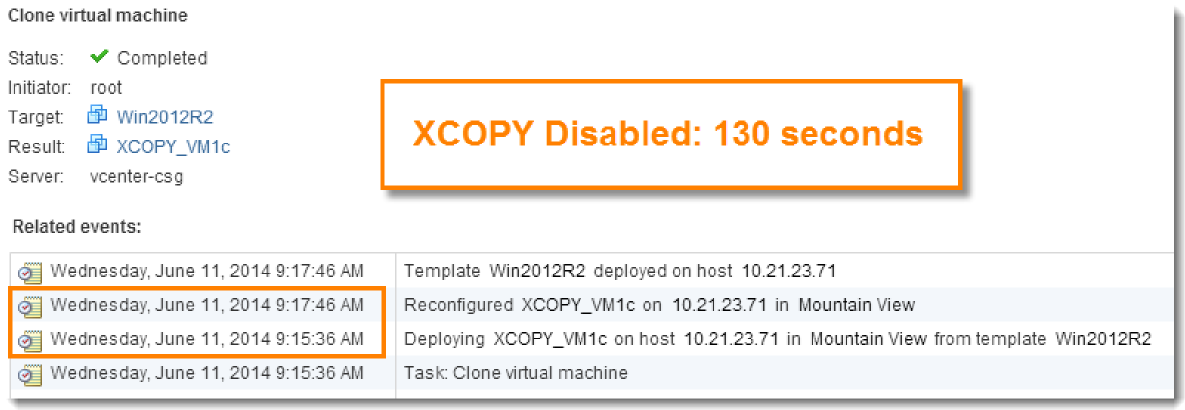
- The source and destination VMFS volumes have different block sizes
- The source file type is RDM and the destination file type is a virtual disk
- The source virtual disk type is eagerzeroedthick and the destination virtual disk type is thin
- The source or destination virtual disk is any kind of sparse or hosted format
- Target virtual machine has snapshots
- The VMFS datastore has multiple LUNs/extents spread across different arrays
- Storage vMotion or cloning between one array and another

[6] Note that this is a host-wide setting and will affect all arrays attached to the host. If a third party array is present and does not support this change leave the value at the default or isolate that array to separate hosts.

Please remember that the majority of these tests are meant to show relative performance improvements. This is why most tests are run with XCOPY on and off. Due to differing virtual machine size, guest OS workload, or resource utilization these numbers may be somewhat different across the user base. These numbers should be used to set general expectations, they are not meant to be exact predictions for XCOPY durations.

Deploy from Template

In this first test, the virtual machine was configured as a template residing on a VMFS on a FlashArray volume (naa.624a9370753d69fe46db318d000**11015**). A single virtual machine was deployed from this template onto a different datastore (naa.624a9370753d69fe46db318d0**0011014**) on the same FlashArray. The test was run twice, once with XCOPY disabled and again with it enabled. With XCOPY enabled, the "deploy from template" operation was far faster and both the IOPS and throughput from the host were greatly reduced for the duration of the operation.



Clone virtual machine

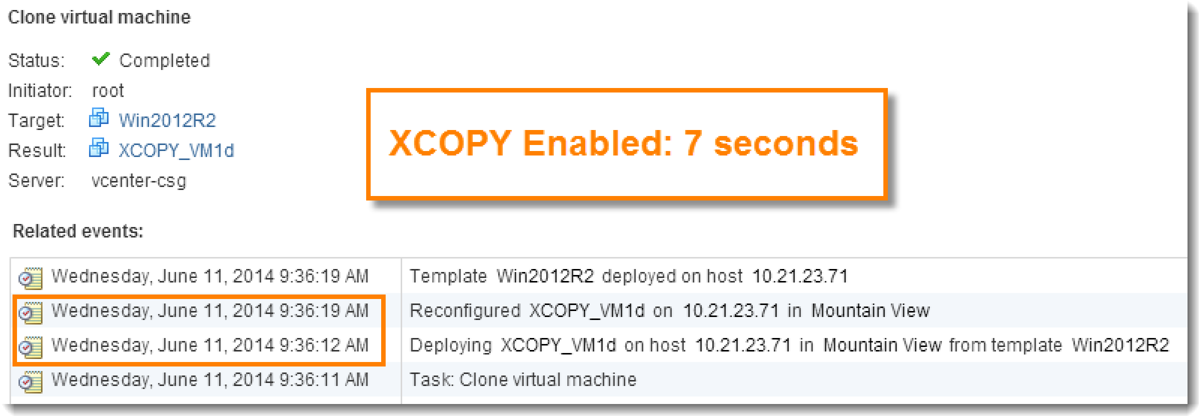
Status: Completed
Initiator: root
Target: Win2012R2
Result: XCOPY_VM1c
Server: vcenter-csg

XCOPY Disabled: 130 seconds

Related events:

Wednesday, June 11, 2014 9:17:46 AM	Template Win2012R2 deployed on host 10.21.23.71
Wednesday, June 11, 2014 9:17:46 AM	Reconfigured XCOPY_VM1c on 10.21.23.71 in Mountain View
Wednesday, June 11, 2014 9:15:36 AM	Deploying XCOPY_VM1c on host 10.21.23.71 in Mountain View from template Win2012R2
Wednesday, June 11, 2014 9:15:36 AM	Task: Clone virtual machine

Figure 9. Deploy from template operation with XCOPY disabled



Clone virtual machine

Status: Completed
Initiator: root
Target: Win2012R2
Result: XCOPY_VM1d
Server: vcenter-csg

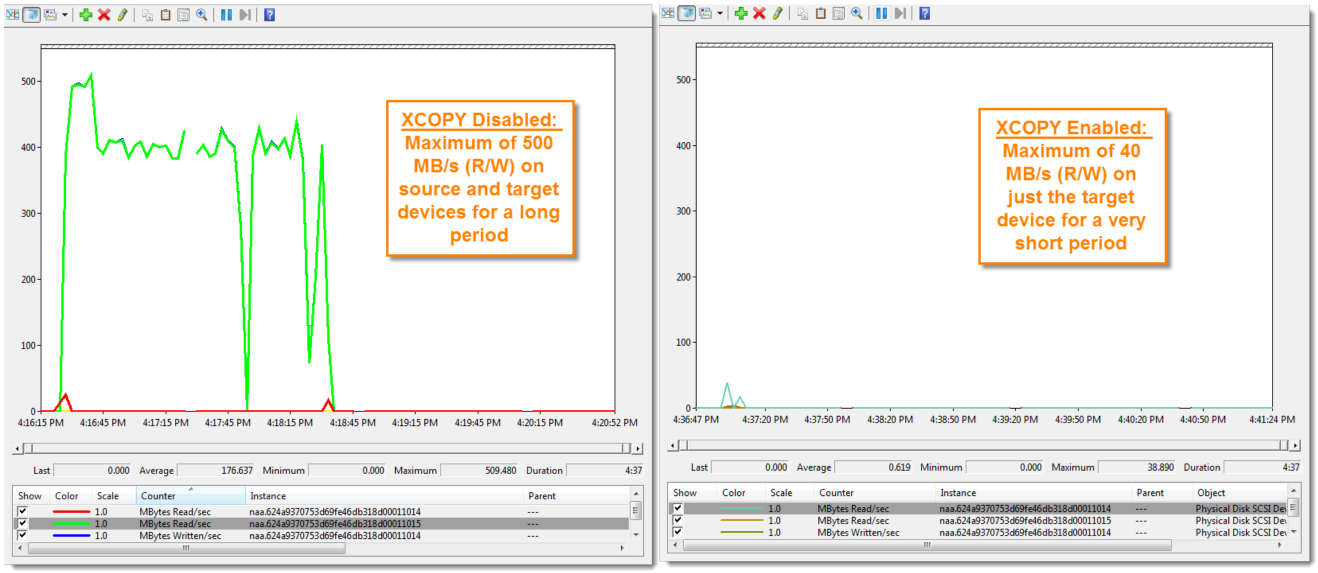
XCOPY Enabled: 7 seconds

Related events:

Wednesday, June 11, 2014 9:36:19 AM	Template Win2012R2 deployed on host 10.21.23.71
Wednesday, June 11, 2014 9:36:19 AM	Reconfigured XCOPY_VM1d on 10.21.23.71 in Mountain View
Wednesday, June 11, 2014 9:36:12 AM	Deploying XCOPY_VM1d on host 10.21.23.71 in Mountain View from template Win2012R2
Wednesday, June 11, 2014 9:36:11 AM	Task: Clone virtual machine

Figure 10. Deploy from template operation with XCOPY enabled

The above images show the vSphere Web Client log of the "deploy from template" operation times, and it can be seen from the comparison that the deployment operation time was reduced from over two minutes down to seven seconds. The following images show the perfmon graphs gathered from esxtop comparing total IOPS and total throughput when XCOPY is enabled and disabled. Note that the scales are identical for both the XCOPY-enabled and XCOPY-disabled charts.



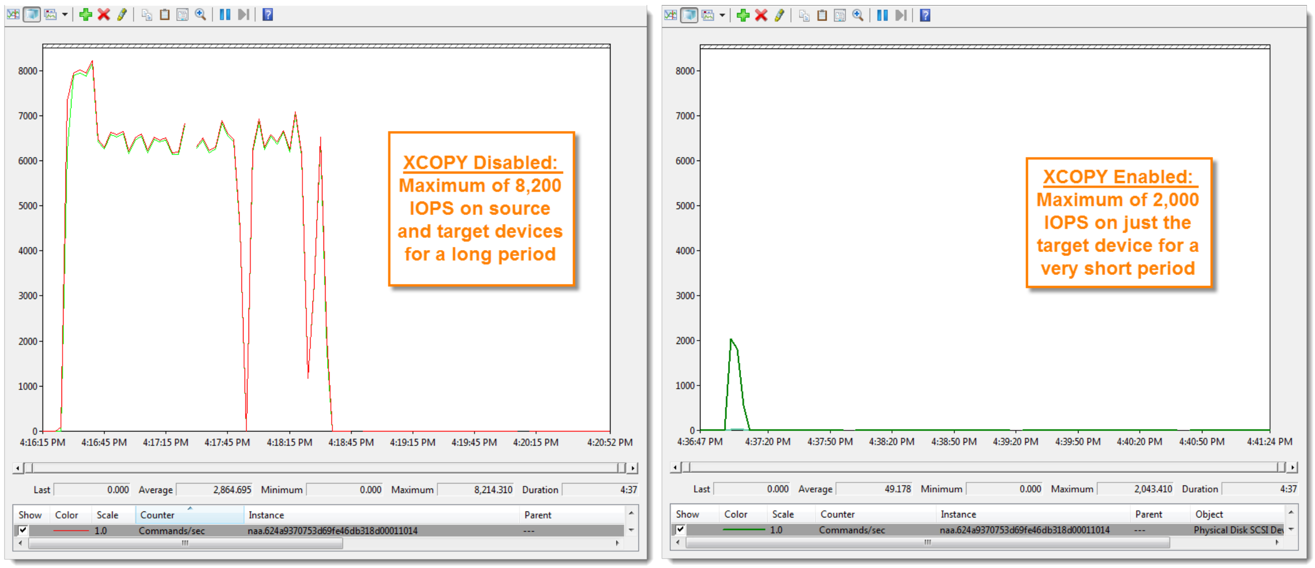
Last	Average	Minimum	Maximum	Duration
0.000	176.637	0.000	509.480	4:37

Show	Color	Scale	Counter	Instance	Parent
✓	🟥	1.0	MBytes Read/sec	naa.624a9370753d69fe46db318d00011014	---
✓	🟩	1.0	MBytes Read/sec	naa.624a9370753d69fe46db318d00011015	---
✓	🟦	1.0	MBytes Written/sec	naa.624a9370753d69fe46db318d00011014	---

Last	Average	Minimum	Maximum	Duration
0.000	0.619	0.000	38.890	4:37

Show	Color	Scale	Counter	Instance	Parent	Object
✓	🟩	1.0	MBytes Read/sec	naa.624a9370753d69fe46db318d00011014	---	Physical Disk SCSI Dev
✓	🟧	1.0	MBytes Read/sec	naa.624a9370753d69fe46db318d00011015	---	Physical Disk SCSI Dev
✓	🟫	1.0	MBytes Written/sec	naa.624a9370753d69fe46db318d00011014	---	Physical Disk SCSI Dev

Figure 12. Deploy from template throughput improvement with XCOPY



Last	Average	Minimum	Maximum	Duration
0.000	2,864.695	0.000	8,214.310	4:37

Show	Color	Scale	Counter	Instance	Parent
✓	🟥	1.0	Commands/sec	naa.624a9370753d69fe46db318d00011014	---

Last	Average	Minimum	Maximum	Duration
0.000	49.178	0.000	2,043.410	4:37

Show	Color	Scale	Counter	Instance	Parent	Object
✓	🟩	1.0	Commands/sec	naa.624a9370753d69fe46db318d00011014	---	Physical Disk SCSI Dev

Figure 11. Deploy from template IOPS improvement without XCOPY

Simultaneous Deploy From Template Operations

This improvement does not diminish when many virtual machines are deployed at once. In the next test the same template was used but instead of one virtual machine being deployed, 8 virtual machines were concurrently deployed from the template. This process was automated using the following basic PowerCLI script.

```
for ($i=0; $i -le 7; $i++)

{

   New-vm -vmhost <host name> -Name "<VM name prefix>$i" -Template <template name> -Datastore <datastore name> -
   runasync

}
```

The preceding script deploys all 8 VMs to the same target datastore. It took 4.5 minutes for the deployment of 16 VMs with XCOPY disabled to complete and it only took 35 seconds when XCOPY was enabled. For an improvement of about 8x. A single VM deployment improvement (as revealed in the previous example) was about 5x so the efficiency gains actually *improve* as deployment concurrency is scaled up. Non-XCOPY based deployment characteristics (throughput/IOPS/duration) increase in an almost linear fashion along with an increased VM count, while XCOPY-based deployment characteristics increase at a much slower comparative rate due to the great ease at which the FlashArray can handle XCOPY operations. The chart shows this scaled up from 1 to 16 VMs at a time and the difference between enabling and disabling XCOPY. As can be noted, the difference really materializes as the concurrency increases.

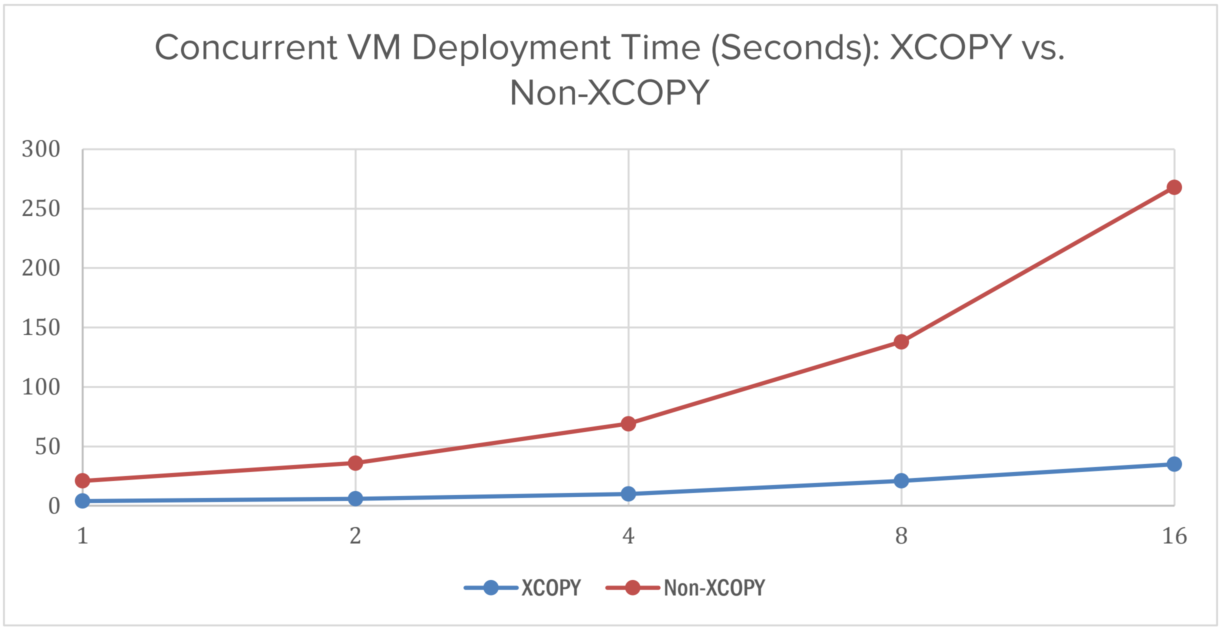


Figure 13. XCOPY vs. Non-XCOPY Concurrent VM deployment time

Storage vMotion

Storage vMotion operations can also benefit from XCOPY acceleration and offloading. Using the same VM configuration as the previous example, the following will show performance differences of migrating the VM from one datastore to another with and without XCOPY enabled. Results will be shown for three different scenarios:

1. Powered-off virtual machine

2. Powered-on virtual machine—but mostly idle
3. Powered-on virtual machine running a workload. 32 KB I/O size, mostly random, heavy on writes.

The chart below shows the results of the tests.

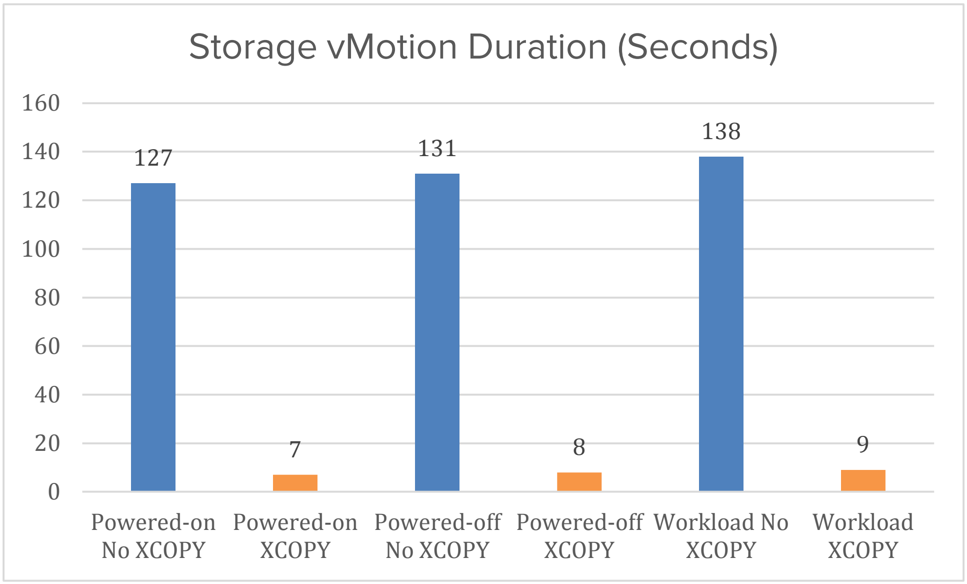


Figure 14. Storage vMotion duration

The chart shows that both a "live" (powered-on) Storage vMotion and a powered-off migration can equally benefit from XCOPY acceleration. The presence of a workload slows down the operation somewhat but nevertheless a substantial benefit can still be observed.

Virtual Disk Type Effect on XCOPY Performance

In vSphere 5.x, the allocation method of the source virtual disk(s) can have a perceptible effect on the copy duration of an XCOPY operation. Thick-type virtual disks (such as zeroedthick or eagerzeroedthick) clone/migrate much faster than a thin virtual disk of the same size with the same data[7]. According to VMware this performance delta is a design decision and is to be expected, refer to the following VMware KB for more information:

http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=2070607

[7] For this reason, it is recommended to never use thin-type virtual disks for virtual machine templates as it will significantly increase the deploy-from-template duration for new virtual machines.

The reason for this is that thin virtual disks are often fragmented on the VMFS volume as they grow in 1 MB increments (the block size of the VMFS) as needed. Since this growth is non-uniform and sporadic, it cannot be guaranteed to be contiguous, and in fact probably is not. This caused XCOPY sessions involving thin virtual disks to ignore the configured setting of the MaxHWTransferSize, which as best practices dictate should be 16 MB. Instead, it would use the block size of the VMFS, which is 1 MB, making the largest transfer size a thin virtual disk could use limited to 1 MB. This markedly slows down XCOPY sessions.

This behavior has been fixed in vSphere 6.0, the performance delta between different types of virtual disks is now gone. Thin virtual disk XCOPY sessions will now adhere to the MaxHWTransferSize and will attempt to use as large as a transfer size as possible.

The following chart shows the duration in seconds of the three types of virtual disks during a “deploy from template operation”. Note the specific mention of vSphere versions, the thin virtual disk test was run twice, once for ESXi 5.x and once for ESXi 6.0. For comparative purposes it shows the durations for both XCOPY-enabled operations and XCOPY-disabled operations. All virtual machines contained the same 75 GB of data in one disk.

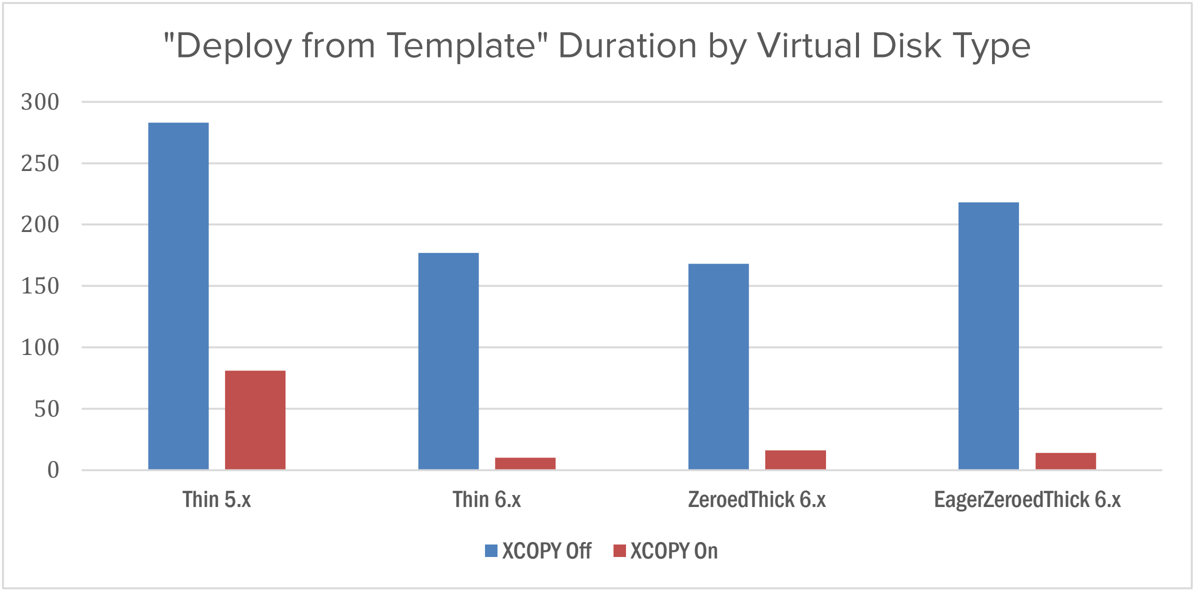


Figure 15. Source virtual disk type effect on XCOPY performance

It can be noted that while each virtual disk type benefits from XCOPY acceleration, thick-type virtual disks benefit the most when it comes to duration reduction of cloning operations in vSphere 5.x. Regardless, all types benefit equally in reduction of IOPS and throughput. Also, standard VM clone or migration operations display similar duration differences as the above “deploy from template” examples.

In vSphere 6.x, the difference between deployment time is gone across the three types of virtual disks. Furthermore, improvement can be noted in non-XCOPY sessions as well for thin virtual disks.

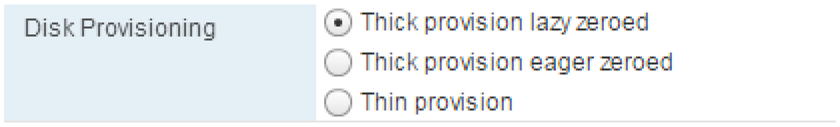
Block Zero or WRITE SAME

ESXi supports three disk formats for provisioning virtual disks:

1. Eagerzeroedthick (thick provision eager zeroed)—the entirety of the virtual disk is completely reserved and pre-zeroed upon creation on the VMFS. This virtual disk allocation mechanism offers the most predictable performance and highest level of protection against capacity exhaustion.

2. Zeroedthick (thick provision lazy zeroed)—this format reserves the space on the VMFS volume upon creation but does not pre-zero the encompassed blocks until the guest OS writes to them. New writes cause iterations of on-demand zeroing in segments of the block size of the target VMFS (almost invariably 1 MB with VMFS 5). There is a slight performance impact on writes to new blocks due to the on-demand zeroing.

3. Thin (thin provision)—this format neither reserves space on the VMFS volume nor pre-zeros blocks. Space is reserved and zeroed on-demand in segments in accordance to the VMFS block size. Thin virtual disks allow for the highest virtual machine density but provide the lowest protection against possible capacity exhaustion. There is a slight performance impact on writes to new blocks due to the on-demand zeroing.



Disk Provisioning
[✓ Thick provision lazy zeroed]
Thick provision eager zeroed
Thin provision

Prior to WRITE SAME support, the performance differences between these allocation mechanisms were distinct. This was due to the fact that before any unallocated block could be written to, zeros would have to be written first, causing an allocate-on-first-write penalty. Therefore, for every new block that was to be written to, there were two writes, the zeros *then* the actual data. For thin and zeroedthick virtual disks, this zeroing was on-demand so the effect was observed by the guest OS in the virtual machine that was issuing writes to new blocks. For eagerzeroedthick, zeroing occurred during deployment and therefore large virtual disks took a long time to create but with the benefit of eliminating any zeroing penalty for new writes.

To reduce this latency, VMware introduced WRITE SAME support. WRITE SAME is a SCSI command that tells a target device (or array) to write a pattern (in this case, zeros) to a target location. ESXi utilizes this command to avoid having to actually send a payload of zeros. Instead, ESXi simply communicates to an array that it needs to write zeros to a certain location on a certain device. This not only reduces traffic on the SAN fabric, but also speeds up the overall process since the zeros do not have to traverse the data path.

This process is optimized even further on the Pure Storage FlashArray. Since the array does not store space-wasting patterns like contiguous zeros, the metadata is created or changed to simply denote that these locations are supposed to be all-zero so any subsequent reads will result in the array returning contiguous zeros to the host. This additional array-side optimization further reduces the time and penalty caused by pre-zeroing of newly-allocated blocks.

The following sections will outline a few examples of WRITE SAME usage to describe expected behavior and performance benefits of using WRITE SAME on the Pure Storage FlashArray.

Deploying Eagerzeroedthick Virtual Disks

The most noticeable operation in which WRITE SAME helps is with the creation of eagerzeroedthick virtual disks. Due to the fact that the zeroing process must be completed during the create operation, WRITE SAME has a dramatic impact on the duration of virtual disk creation and practically eliminates the added traffic that used to be caused by the traditional zeroing behavior.

The following chart shows the deployment time of four differently sized eagerzeroedthick virtual disks when WRITE SAME was enabled (in orange) and disabled (in blue). The enabling of WRITE SAME, on average, reduces the deployment time of these types of virtual disks to about 6x faster regardless of the size.

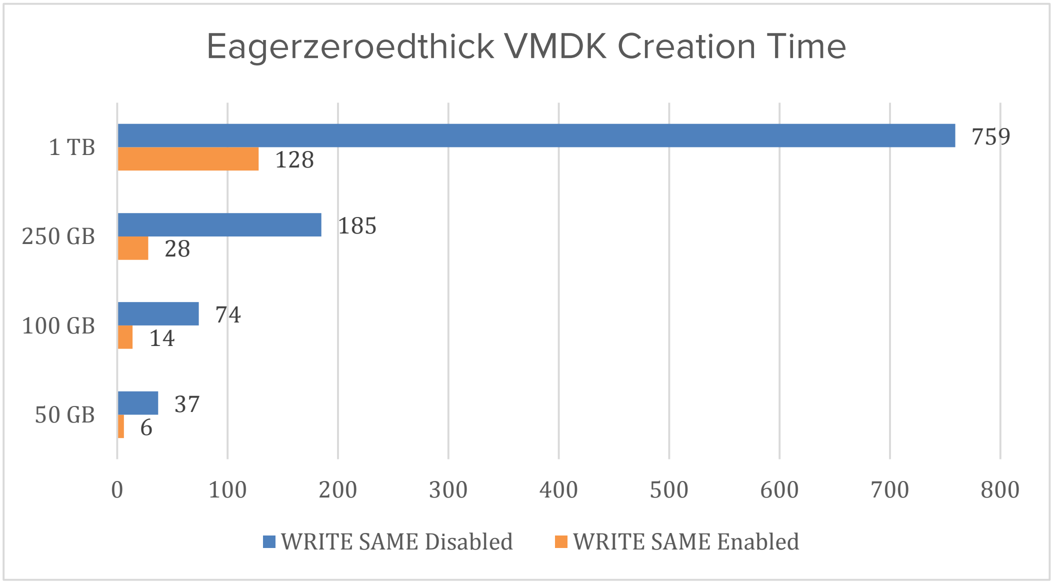


Figure 16. Eagerzeroedthick virtual disk deployment time differences

WRITE SAME also works well in scale on the FlashArray. Below are the results of a test when four 100GB eagerzeroedthick virtual disks were deployed (with vmkfstools) simultaneously.

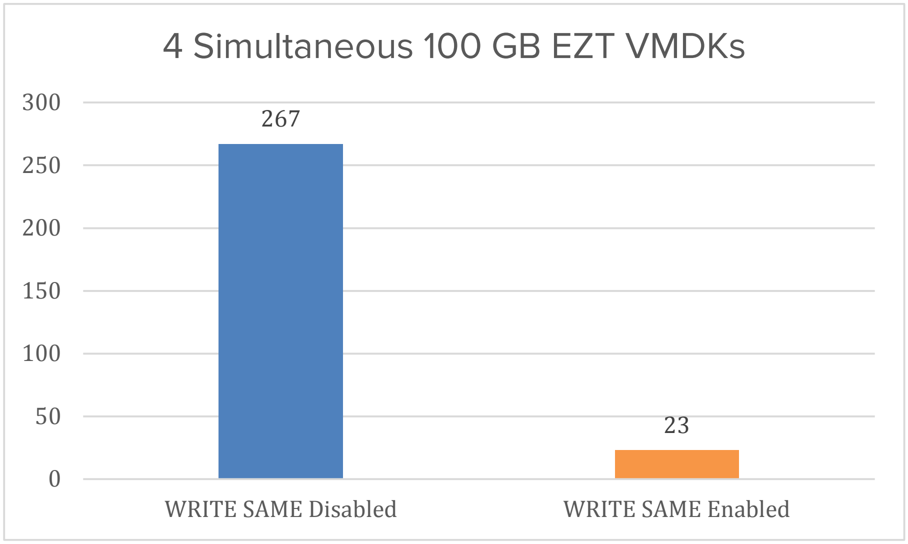


Figure 17. Total simultaneous deployment time for eagerzeroedthick virtual disks

In comparison to the previous chart, where only one virtual disk was deployed at a time, the deployment duration of an eagerzeroedthick virtual disk without WRITE SAME increased almost linearly with the added number of virtual disks (taking 3.5x longer with 4x more disks). When WRITE SAME was enabled, the increase wasn't even twofold (taking 1.6x times longer with 4x more disks). It can be concluded that the Pure Storage FlashArray can easily handle and scale with additional simultaneous WRITE SAME activities.

Zeroedthick and Thin Virtual Disks Zero-On-New-Write Performance

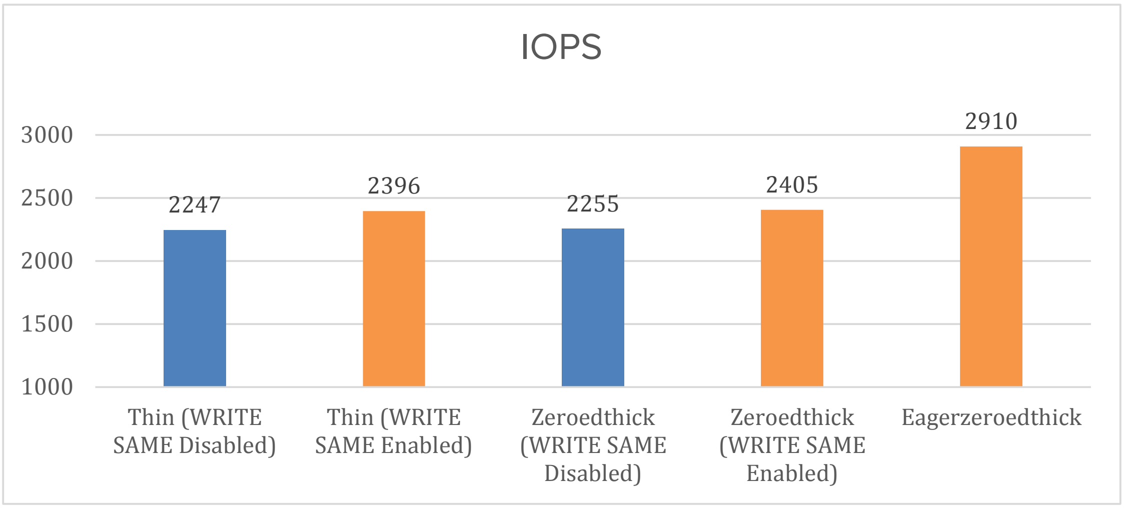
In addition to accelerating up eagerzeroedthick deployment, WRITE SAME also improves performance within thin and zeroedthick virtual disks. Since both types of virtual disks zero-out blocks only upon demand (new writes to previously unallocated blocks) these new writes suffer from additional latency when compared to over-writes. The introduction of WRITE SAME reduces this latency by speeding up the process of initializing this space.

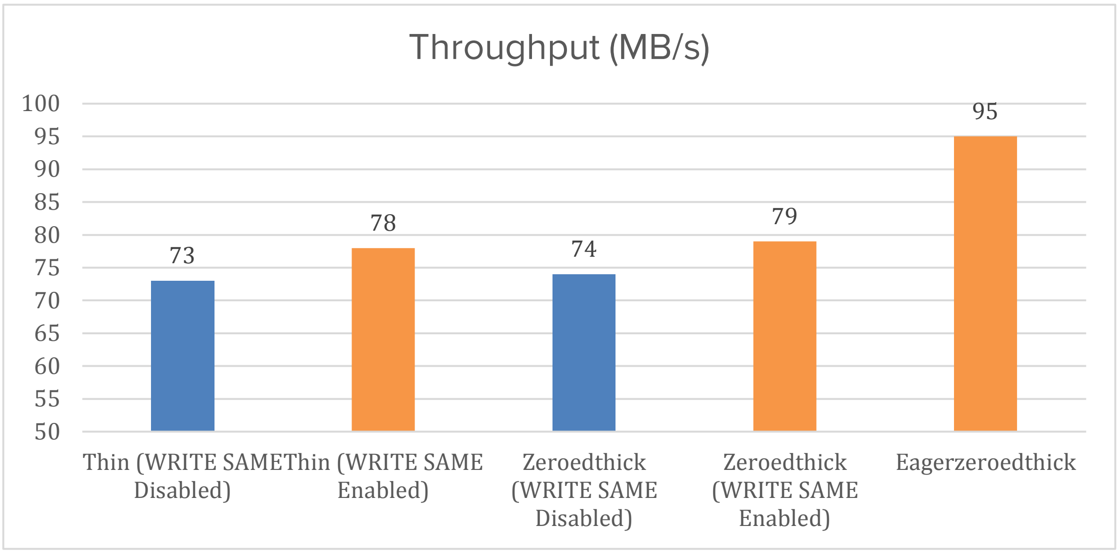
The following test was created to ensure that a large proportion of the workload was new writes so that the write workload always encountered the allocation penalty from pre-zeroing (with the exception of the eagerzeroedthick test which was more or less a control). Five separate tests were run:

1. Thin virtual disk with WRITE SAME disabled.
2. Thin virtual disk with WRITE SAME enabled.
3. Zeroedthick virtual disk with WRITE SAME disabled.
4. Zeroedthick virtual disk with WRITE SAME enabled.
5. Eagerzeroedthick virtual disk

The workload was a 100% sequential 32 KB write profile in all tests. As expected the lowest performance (lowest throughput, lowest IOPS and highest latency) was with thin or zeroedthick with WRITE SAME disabled (zeroedthick slightly out-performed thin). Enabling WRITE SAME improved both, but eagerzeroedthick virtual disks out-performed all of the other virtual disks regardless of WRITE SAME use. With WRITE SAME enabled eagerzeroedthick performed better than thin and zeroedthick by 30% and 20% respectively in both IOPS and throughput, and improved latency from both by 17%.

The following three charts show the results for throughput, IOPS and latency.





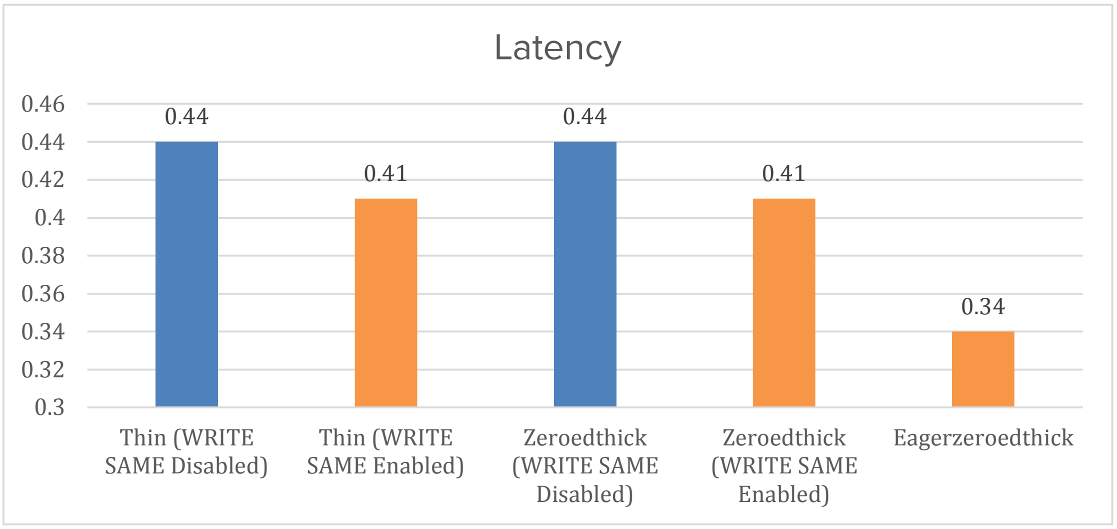


Figure 18. IOPS, Throughput and Latency differences across virtual disk types

Note that all of the charts do not start the vertical axis at zero—this is to better illustrate the deltas between the different tests.

It is important to understand that these tests are **not meant** to authoritatively describe performance differences between virtual disks types—instead they are meant to express the performance improvement introduced by WRITE SAME for writes to uninitialized blocks. Once blocks have been written to, the performance difference between the various virtual disk types evaporates. Furthermore, as workloads become more random and/or more read intensive, this overall performance difference will become less perceptible. This section is mostly a lab exercise, except for the most performance-sensitive workloads, performance should not be a huge factor in virtual disk type choice.

From this set of tests we can conclude:

1. Regardless of WRITE SAME status, eagerzeroedthick virtual disks will always out-perform the other types for new writes.
2. The latency overhead of zeroing-on-demand with WRITE SAME disabled is about 30% (in other words the new write latency of thin/zeroedthick is 30% greater than with eagerzeroedthick).
    a. The latency overhead is reduced from 30% to 20% when WRITE SAME is enabled. The duration of the latency overhead is also reduced when WRITE SAME is enabled.
3. The IOPS and throughput reduction caused by zeroing-on-demand with WRITE SAME disabled is about 23% (in other words the possible IOPS/throughput of thin/zeroedthick to new blocks is 23% lower than with eagerzeroedthick).
    a. The possible IOPS/throughput reduction to new blocks is reduced from 23% to 17% when WRITE SAME is enabled.

Dead Space Reclamation or UNMAP

In block-based storage implementations, the file system is managed by the host, not the array. Because of this, the array does not typically know when a file has been deleted or moved from a storage volume and therefore does not know when or if to release the space. This behavior is especially detrimental in thinly-provisioned environments where that space could be immediately allocated to another device/application or just returned to the pool of available storage.

In vSphere 5.0 Update 1, VMware introduced Dead Space Reclamation which makes use of the SCSI UNMAP command to help remediate this issue. UNMAP enables an administrator to initiate a reclaim operation from an ESXi host to compatible block storage devices. The reclaim operation instructs ESXi to inform the storage array of space that previously had been occupied by a virtual disk and is now freed up by either a delete or migration and can be reclaimed. This enables an array to accurately manage and report space consumption of a thinly-provisioned datastore and enables users to better monitor and forecast new storage requirements.

In ESXi 5.x, the advanced ESXi option EnableBlockDelete is defunct and does not enable or disable any behavior when changed. This option is re-introduced in ESXi 6.0 and is fully functional. Please refer to the end of this section for discussion concerning this parameter.

To reclaim space in vSphere 5.0 U1 through 5.1, SSH into the ESXi console and run the following commands:

1. Change into the directory of the VMFS datastore you want to run a reclamation on:

   ```
   cd /vmfs/volumes/<datastore name>
   ```

2. Then run `vmkfstools` to reclaim the space by indicating the percentage of the free space you would like to reclaim (up to 99%):

   ```
   vmkfstools -y 99
   ```

It should be noted that while UNMAP on the FlashArray is a quick and unobtrusive process, ESXi does create a balloon file when using the vmkfstools method to fill the entire specified range of free space for the duration of the reclaim process. This could possibly lead to a temporary out-of-space condition on the datastore if there are thin virtual disks that need to grow. When a datastore contains a large amount of thin virtual disks, large UNMAP reclaim percentages should be entered with care or avoided entirely. This is not an issue if the virtual disks are all thick provisioned, or the ESXi server is version 5.5 or later.

To reclaim space in vSphere 5.5, the `vmkfstools -y` option has been deprecated and UNMAP is now available in `esxcli`. UNMAP can be run anywhere `esxcli` is installed and therefore does not require an SSH session:

1. Run `esxcli` and supply the datastore name. Optionally, a block iteration count can be specified, otherwise it defaults to reclaiming 200 MB per iteration:

```
esxcli storage vmfs unmap -l <datastore name> -n (blocks per iteration)
```

The `esxcli` option can also be leveraged from the VMware vSphere PowerCLI using the cmdlet `GetEsxCli`*:*

```
$esxcli=get-esxcli -VMHost <ESXi host>

$esxcli.storage.vmfs.unmap(10000, "<datastore name>", $null)
```

The `esxcli` version of UNMAP allows the user to delineate the number of blocks unmapped per iteration of the process. The default value is 200 (in other words 200 MB) and can be increased or decreased as necessary. Since the UNMAP process is a workload of negligible impact on the Pure Storage FlashArray, this value can be increased to dramatically reduce the duration of the reclaim process. The time to reclaim reduces exponentially as the block count increases. Therefore, increasing the block count to something sufficiently higher is recommended. While the FlashArray can handle very large values for this, ESXi does not support increasing the block count any larger than 1% of the free capacity of the target VMFS volume (one block equals one megabyte). Consequently, the best practice for block count during UNMAP is no greater than 1% of the free space.

So as an example, if a VMFS volume has 1,048,576 MB free, the largest block count supported is 10,485 (always round down). If you specify a larger value the command will still be accepted, but ESXi will override the value back down to the default of 200 MB, which will profoundly slow down the operation. In order to see if the value was overridden or not, you can check the hostd.log file in the /var/log/ directory on the target ESXi host. For every UNMAP operation there will be a series of messages that dictate the block count for every iteration. Examine the log and look for a line that indicates the UUID of the VMFS volume being reclaimed, the line will look like the example below:

```
Unmap: Async Unmapped 5000 blocks from volume 545d6633-4e026dce-d8b2-
90e2ba392174
```

A simple method to calculate this 1 % value is via PowerCLI. Below is a simple example to take in a datastore object by name and return a proper block count to enter into an UNMAP command.

```
$datastore = get-datastore <datastore name>

$blockcount = [math]::floor($datastore.FreeSpaceMB * .01)
```

That will change free capacity into a rounded off number and then give you the proper block count for any given datastore. Enter that number ($blockcount in the above instance) into an UNMAP command and start the UNMAP procedure.

It is imperative to calculate the block count value based off of the 1% of the free space only when that capacity is expressed in megabytes—since VMFS 5 blocks are 1 MB each. This will allow for simple and accurate identification of the largest allowable block count for a given datastore. Using GB or TB can lead to rounding errors, and as a result, too large of a block count value. Always round off decimals to the lowest near MB in order to calculate this number (do not round up).

An example full UNMAP script can be [found here]().

The UNMAP procedure (regardless of the ESXi version) causes Purity to remove the metadata pointers to the data for that given volume and if no other pointers exist, the data is also tagged for removal. Therefore, used capacity numbers may not change on the array space reporting after an UNMAP operation. Since the data is often heavily deduplicated, it is highly possible that the data that was reclaimed is still in use by another volume or other parts within the same volume on the array. In that case, the specific metadata pointers are only removed and the data itself remains since it is still in use by other metadata pointers. That being said, it is still important to UNMAP regularly to make sure stale pointers do not remain in the system. Regular reclamation in the long term allows this data to eventually be removed as the remaining pointers are deleted.

! From ESXi 5.5 Patch 3 and later, any UNMAP operation against a datastore that is 75% or more full will use a block count of 200 regardless to any block count specified in the command. For more information refer to the VMware KB article here.

UNMAP Operation Duration

The following section will outline a few examples of UNMAP usage to describe expected behavior and performance characteristics of using UNMAP on the Pure Storage FlashArray.

For the `esxcli` UNMAP method, the only configurable option is the block count per iteration—and as previously mentioned Pure Storage recommends setting this to a higher number in order to complete the UNMAP operation as fast as possible. The below chart shows the indirect relationship between the duration of the UNMAP operation and the number of blocks per iteration.

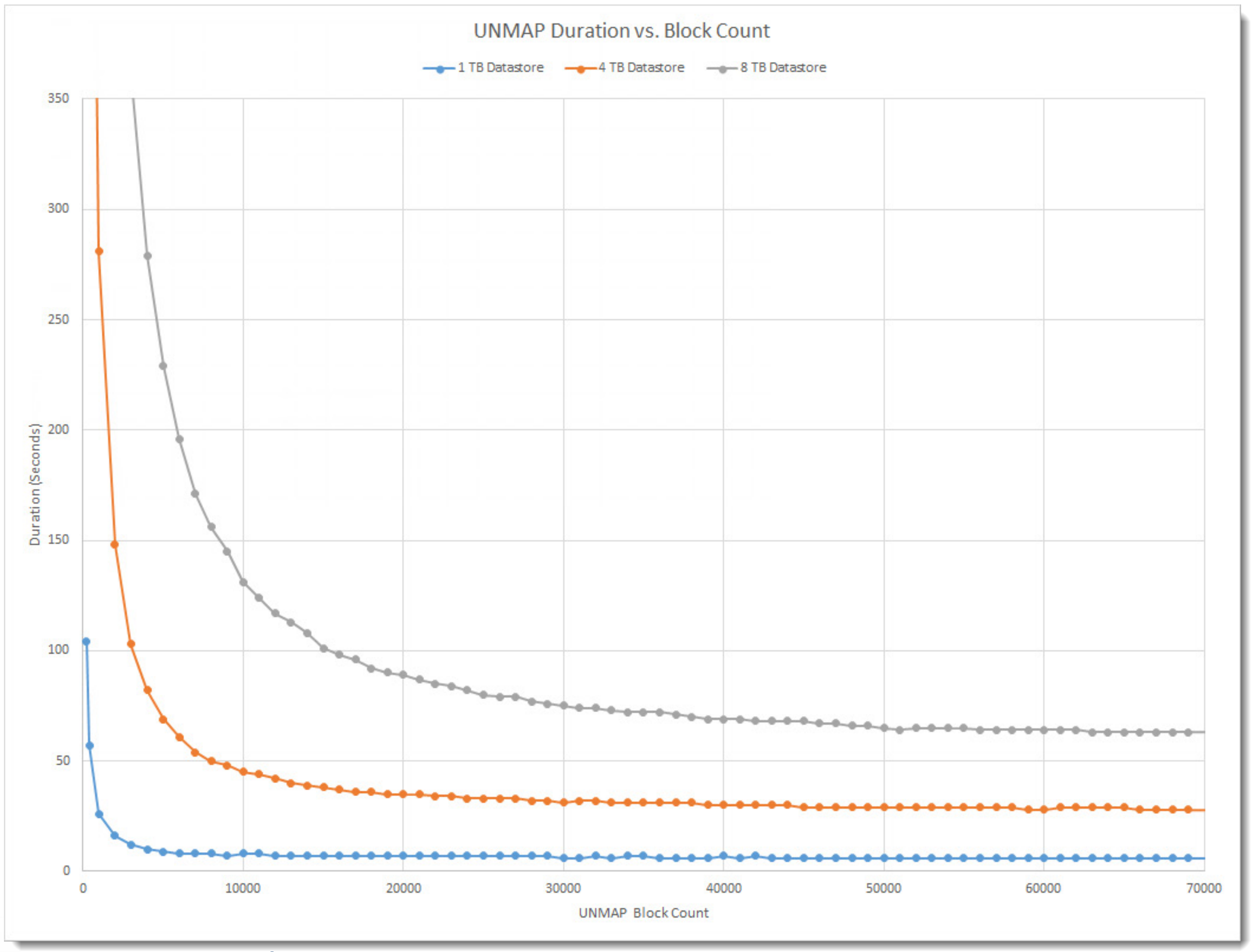


Figure 19. Relationship of block counts and UNMAP duration

In the tests shown in the above charts, three datastores with different amounts of free capacity (1 TB, 4 TB and 8 TB) were tested with a large variety of block counts. As can be noted, all increments show great improvement in UNMAP operation duration as the block count increases from 200 (default) to about 40,000. At this point, increases in reclaim times start to display diminishing returns.

It is important to note that the duration of the UNMAP procedure is dictated by the following things:

- The amount of free capacity on the VMFS volume. The total capacity is irrelevant.
- The specified block count.
- The workload on the target VMFS (heavy workloads can slow the rate at which ESXi issues UNMAP).

In ESXi 5.5 GA release through ESXi 5.5 U2, the block count was configurable to any number the user desired. It was discovered that the use of large block counts in certain situations occasionally caused ESXi physical CPU lockups which then could lead to a crash of the ESXi host. Therefore, VMware introduced a fix in ESXi 5.5 Patch 3, which causes the following block count behavior changes:

1. ESXi will override any block count larger than 1% of the free capacity of the target VMFS datastore and force it back to 200. Therefore, use block counts no greater than 1% of the free space to provide for the best UNMAP duration performance.
2. For volumes that are 75% full or greater no block count other than 200 is supported. Any other block count entered will be ignored in this situation and 200 will be used instead.

The Pure Storage FlashArray, as displayed in the previous chart, prefers a large as possible block count in order to finish the reclaim operation as quickly as possible. Since ESXi now enforces an upper limit of 1 % of the free space, that is the recommended best practice for reclamation on the Pure Storage FlashArray for all versions of ESXi 5.5 and later (before and after the VMware patch). This provides for the quickest possible reclaim duration while also removing susceptibility to ESXi host crashes.

The VMware KB article:

http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=2086747

The VMware patch:

http://kb.vmware.com/selfservice/search.do?cmd=displayKC&docType=kc&docTypeID=DT_KB_1_1&externalId=2087358

In-Guest UNMAP in ESXi 6.x

The discussion above speaks only about space reclamation directly on a VMFS volume. This pertains to dead space accumulated by the deletion or migration of virtual disks, ISOs or swap files (mainly). The CLI-initiated UNMAP operation does not pertain though to dead space accumulated inside of a virtual disk. Dead space accumulates inside of a virtual disk in the same way that it accumulates on a VMFS volume—deletion or movement of files.

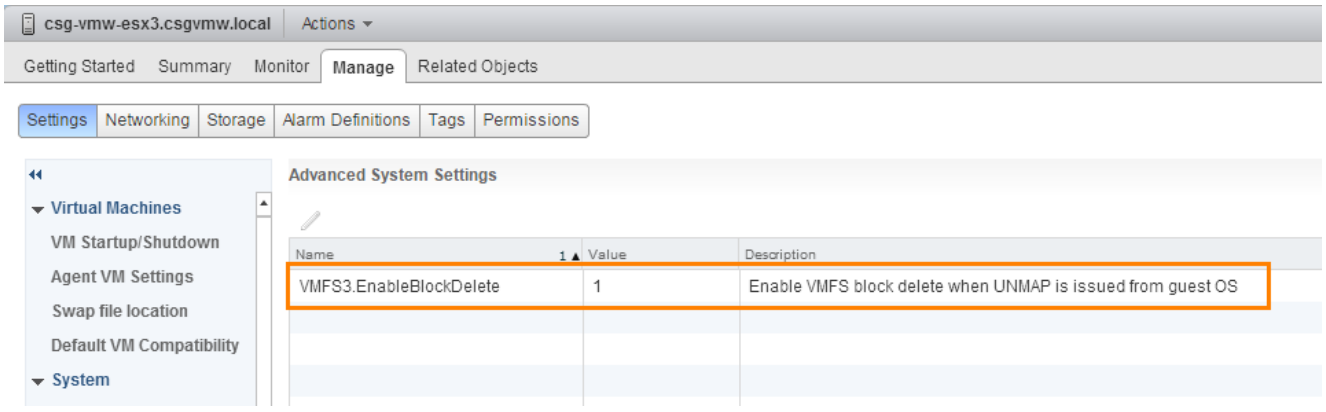
Prior to ESXi 6.0 and virtual machine hardware version 11, guests could not leverage native UNMAP capabilities on a virtual disk because ESXi virtualized the SCSI layer and did not report UNMAP capability to the guest.

In ESXi 6.0, guests running in a virtual machine using hardware version 11 can now issue UNMAP directly to thin virtual disks. The process is as follows:

1. A guest application or user deletes a file from a file system residing on a thin virtual disk
2. The guest automatically (or manually) issues UNMAP to the guest file system on the virtual disk
3. The virtual disk is then shrunk in accordance to the amount of space reclaimed inside of it.
4. If EnableBlockDelete is enabled, UNMAP will then be issued to the VMFS volume for the space that previously was held by the thin virtual disk. The capacity is then reclaimed on the FlashArray.

Currently, in-guest UNMAP support is limited to Windows 2012 R2 or Windows 8. Linux requires a newer version of SCSI support that is under consideration for future versions of ESXi.

Prior to ESXi 6.0, the parameter EnableBlockDelete was a defunct option that was previously only functional in very early versions of ESXi 5.0 to enable or disable automated VMFS UNMAP. This option is now functional in ESXi 6.0 and has been re-purposed to allow in-guest UNMAP to be translated down to the VMFS and accordingly the SCSI volume. By default, EnableBlockDelete is disabled and can be enabled via the Web Client or CLI utilities.



Advanced System Settings

Name	Value	Description
VMFS3.EnableBlockDelete	1	Enable VMFS block delete when UNMAP is issued from guest OS

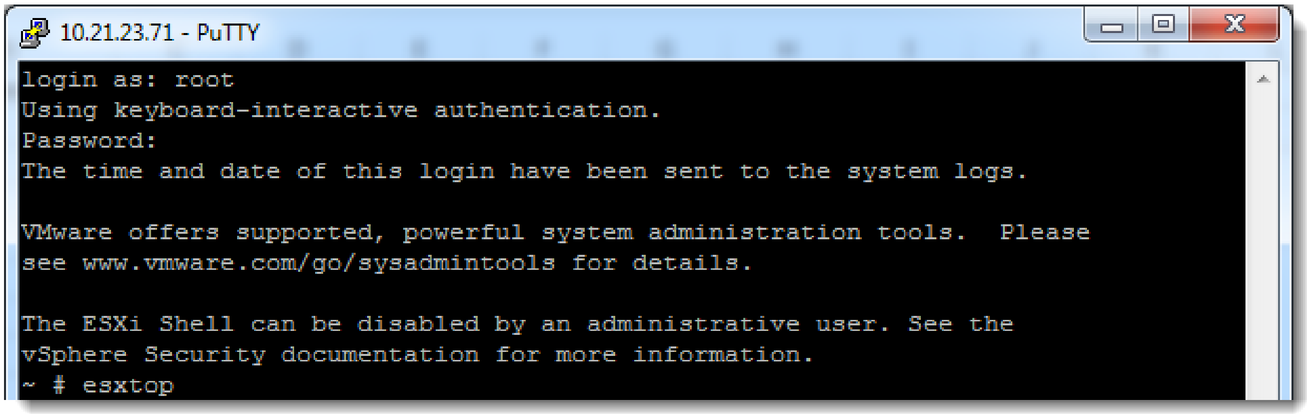
Figure 20. Enabling EnableBlockDelete in the vSphere 6.0 Web Client interface

In-guest UNMAP support does not require this parameter to be enabled. Enabling this parameter only allows in-guest UNMAP to be translated down to the VMFS layer. For this reason, enabling this option is a best practice for ESXi 6.x and later.

Monitoring VAAI with ESXTOP

The simplest way to specifically monitor VAAI activity is through the ESXi performance monitoring tool ESXTOP. ESXTOP is both a real-time monitoring tool and a time-period batch performance gathering tool.

To monitor activity in real-time, log into the ESXi shell via SSH (SSH is not enabled by default—this can be done through the console or from within the vSphere Web Client in the security profile area) and run the ESXTOP command.

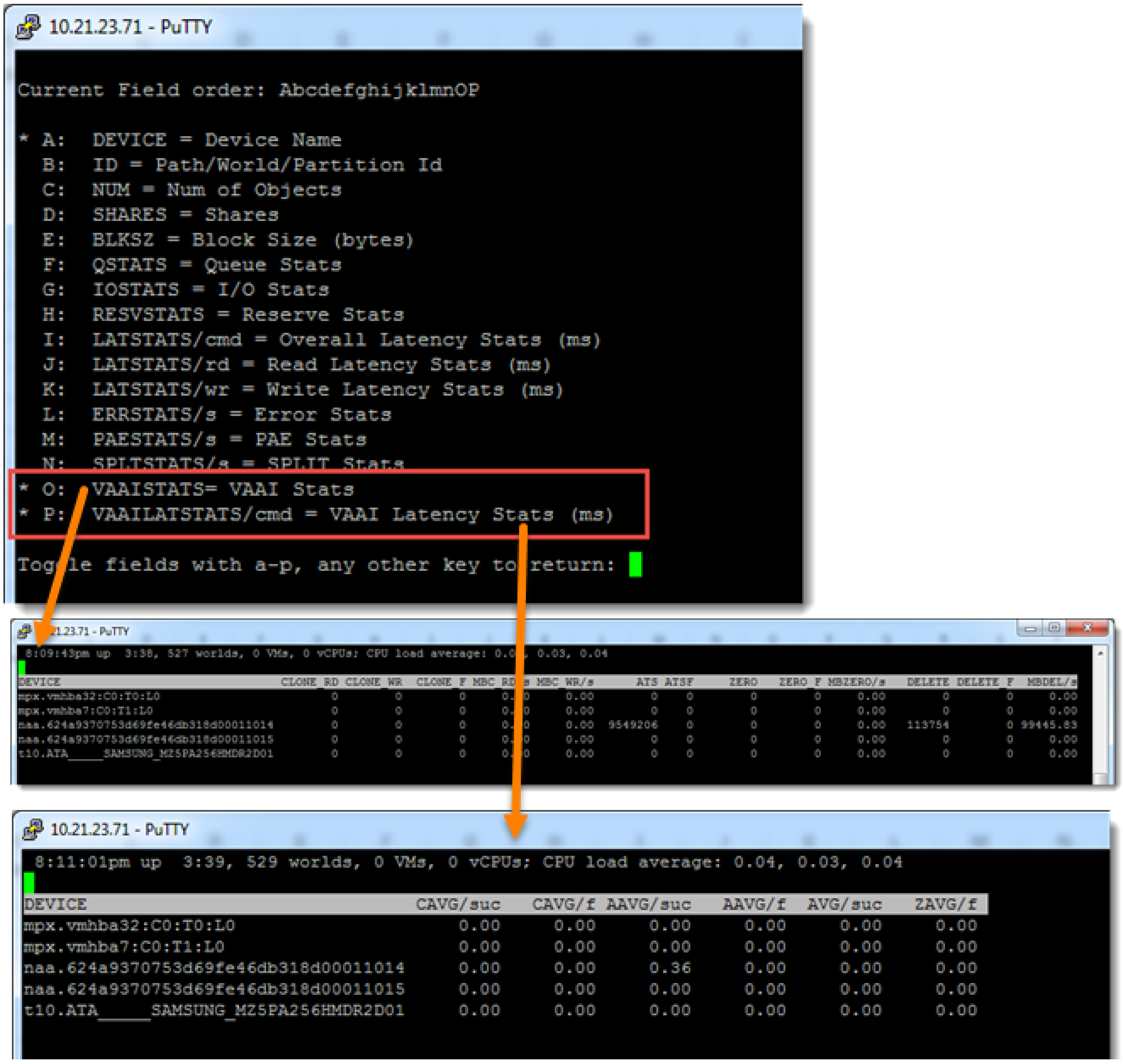


```
login as: root
Using keyboard-interactive authentication.
Password:
The time and date of this login have been sent to the system logs.

VMware offers supported, powerful system administration tools.  Please
see www.vmware.com/go/sysadmintools for details.

The ESXi Shell can be disabled by an administrative user. See the
vSphere Security documentation for more information.
~ # esxtop
```

Figure 21. Starting ESXTOP

ESXTOP offers a variety of different screens for the different performance areas it monitors. Each screen can be switched to by pressing a respective key. See the table below for the options.

Key	Description
c	CPU resources
p	CPU power
m	Memory resources
d	Disk adapters
u	Disk devices
v	Virtual machine disks
n	IP Network resources
i	Interrupts



```
Current Field order: AbcdefghijklmnOP

* A:  DEVICE = Device Name
  B:  ID = Path/World/Partition Id
  C:  NUM = Num of Objects
  D:  SHARES = Shares
  E:  BLKSZ = Block Size (bytes)
  F:  QSTATS = Queue Stats
  G:  IOSTATS = I/O Stats
  H:  RESVSTATS = Reserve Stats
  I:  LATSTATS/cmd = Overall Latency Stats (ms)
  J:  LATSTATS/rd = Read Latency Stats (ms)
  K:  LATSTATS/wr = Write Latency Stats (ms)
  L:  ERRSTATS/s = Error Stats
  M:  PAESTATS/s = PAE Stats
  N:  SPLTSTATS/s = SPLIT Stats
* O:  VAAISTATS= VAAI Stats
* P:  VAAILATSTATS/cmd = VAAI Latency Stats (ms)

Toggle fields with a-p, any other key to return:
```

```
 8:09:43pm up  3:38, 527 worlds, 0 VMs, 0 vCPUs; CPU load average: 0.04, 0.03, 0.04

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
mpx.vmhba32:C0:T0:L0                          0        0        0     0.00     0.00        0    0         0         0     0.00       0        0     0.00
mpx.vmhba7:C0:T1:L0                           0        0        0     0.00     0.00        0    0         0         0     0.00       0        0     0.00
naa.624a9370753d69fe46db318d00011014          0        0        0     0.00     0.00  9549206    0         0         0     0.00  113754        0 99445.83
naa.624a9370753d69fe46db318d00011015          0        0        0     0.00     0.00        0    0         0         0     0.00       0        0     0.00
t10.ATA_____SAMSUNG_MZ5PA256HMDR2D01          0        0        0     0.00     0.00        0    0         0         0     0.00       0        0     0.00
```

```
 8:11:01pm up  3:39, 529 worlds, 0 VMs, 0 vCPUs; CPU load average: 0.04, 0.03, 0.04

DEVICE                                 CAVG/suc   CAVG/f  AAVG/suc   AAVG/f  AVG/suc   ZAVG/f
mpx.vmhba32:C0:T0:L0                       0.00     0.00      0.00     0.00     0.00     0.00
mpx.vmhba7:C0:T1:L0                        0.00     0.00      0.00     0.00     0.00     0.00
naa.624a9370753d69fe46db318d00011014       0.00     0.00      0.36     0.00     0.00     0.00
naa.624a9370753d69fe46db318d00011015       0.00     0.00      0.00     0.00     0.00     0.00
t10.ATA_____SAMSUNG_MZ5PA256HMDR2D01       0.00     0.00      0.00     0.00     0.00     0.00
```

Figure 22. ESXTOP columns

In most cases for VAAI information the disk devices screen is the pertinent one, so press “u” to navigate to that screen. By default, VAAI information is not listed and must be added. To add VAAI information press “f” and then “o” and “p”. To deselect other columns simply type their corresponding letter. Press enter to return.

The CLONE or MBC counters refer to XCOPY, ATS refers to Hardware Assisted Locking, ZERO refers to WRITE SAME and DELETE refers to UNMAP. If a collection of this data for later review is preferred over real time analysis esxtop can be gathered for a specific amount of time and saved as a CSV file.

Batch mode, as it is referred to, takes a configuration of ESXTOP and runs for a given interval. This can create a tremendous amount of data so it is advised to remove any and all counters that are not desired using the "f" option and saving the configuration with "W".

Once the configuration is complete (usually just VAAI info and possibly some other storage counters) ESXTOP can be run again in batch mode by executing:

```
esxtop -b -d 5 -n 50 > /vmfs/volumes/datastore/esxtopresults.csv
```

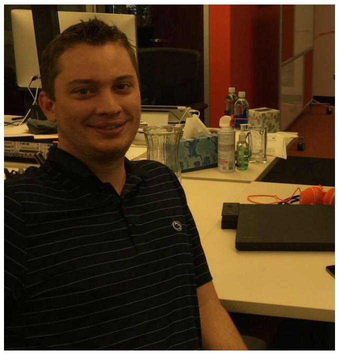
Make sure the csv file is output onto a VMFS volume and not onto the local file system. Putting it locally will often truncate the csv file and performance data will be lost.

The "-b" indicates batch mode, "-d" is a sample period and "-n" is how many intervals should be taken, so in this example it would take counters every 5 seconds, 50 times. So a total capture period of 250 seconds.

References

1. Interpreting esxtop statistics - http://communities.vmware.com/docs/DOC-11812
2. Disabling VAAI Thin Provisioning Block Space Reclamation (UNMAP) in ESXi 5.0 - kb.vmware.com/kb/**2007427**
3. Pure Storage and VMware vSphere Best Practices Guide http://info.purestorage.com/WP-PureStorageandVMwarevSphereBestPracticesGuide_Request.html
4. VMware Storage APIs –Array Integration http://www.vmware.com/files/pdf/techpaper/VMware-vSphere-Storage-API-Array-Integration.pdf
5. Frequently Asked Questions for vStorage APIs for Array Integration http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1021976

About the Author



Cody Hosterman is a VMware-focused Solutions Architect at Pure Storage since 2014. His primary responsibility is overseeing, testing, documenting, and demonstrating VMware-based integration with the Pure Storage FlashArray platform. Cody has been working in vendor enterprise storage/VMware integration roles since 2008.

Cody graduated from the Pennsylvania State University with a bachelors degreee in Information Sciences & Technology in 2008. Special areas of focus include core ESXi, vRealize, Site Recovery Manager and PowerCLI. Cody has been named a VMware vExpert from 2013 through 2015.

Blog: http://www.codyhosterman.com

Twitter: @codyhosterman

PURESTORAGE

Pure Storage, Inc.
Twitter: @purestorage
www.purestorage.com

650 Castro Street, Suite #260
Mountain View, CA 94041

T: 650-290-6088
F: 650-625-9667

Sales: sales@purestorage.com
Support: support@purestorage.com
Media: pr@purestorage.com
General: info@purestorage.com