vsanSparse Tech Note

February 26, 2018

Table of Contents

1. Introduction

Introduction & Overview

1.1 Introduction

vSAN 6.0 introduces a new on-disk format that includes VirstoFS technology. This always-sparse filesystem provides the basis for a new snapshot format, also introduced with vSAN 6.0, called vsanSparse. Through the use of the underlying sparseness of the filesystem and a new, in-memory metadata cache for lookups, vsanSparse offers greatly improved performance when compared to previous virtual machine snapshot implementations.

1.2 Virtual Machine Snapshot Overview

VMware, through the use of VM snapshots, provides the ability to capture a point-in-time (PIT) state of a virtual machine. This includes the virtual machine's storage, memory and other devices such as virtual NICs, etc. Using the Snapshot Manager in the vSphere client, administrators can create, revert or delete VM snapshots. A chain of up to 32 snapshots is supported.

Snapshots can capture virtual machines that are powered-on, powered-off or even suspended. When the virtual machine is powered-on, there is an option to capture the virtual machine's memory state, and allow the virtual machine to be reverted to powered-on a point in time.

There is also a "quiesce" option. If this option is selected when taking a snapshot of a powered-on virtual machine, VMware Tools may use either its own sync driver or VSS (Microsoft's Volume Shadow Copy Service) to quiesce not only the guest OS filesystem, but also any Microsoft applications that understand VSS directives. This allows for application consistent backups of virtual machines, a typical use case for virtual machine snapshots.

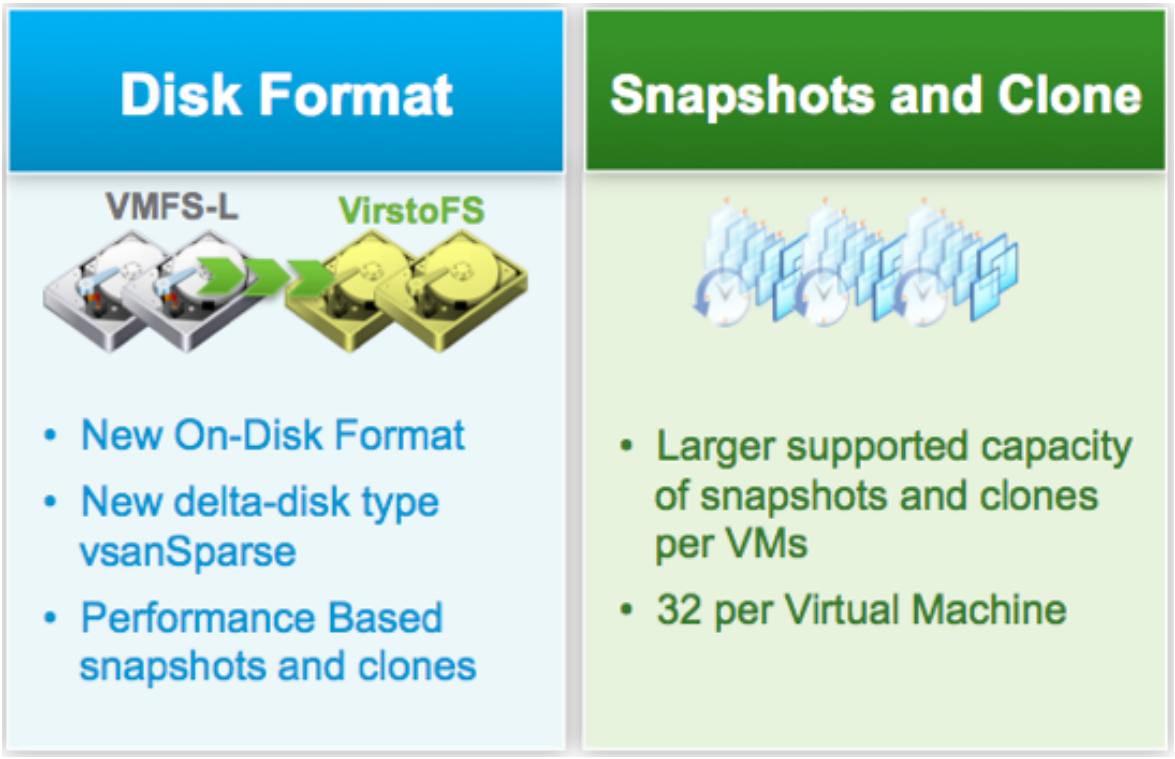
For further information on how to use virtual machine snapshots in vSphere environments, please refer to the official vSphere documentation on VMware.com.

2. Introducing vsanSparse Snapshots

introduction to vSANSparse Snapshots

2.1 Introducing vsanSparse Snapshots

As mentioned in the introduction, vSAN 6.0 has a new on-disk (v2) format that facilitates the introduction of a new type of performance-based snapshot. The new vsanSparse format leverages the underlying sparseness of the new VirstoFS filesystem (v2) on-disk format and a new in-memory caching mechanism for tracking updates. This v2 format is an always-sparse file system (512-byte block size instead of 1MB block size on VMFS-L) and is only available with vSAN 6.0.



When a virtual machine snapshot is created on vSAN 5.5, a vmfsSparse/redo log object is created (you can find out more about this format in appendix A of this paper). In vSAN 6.0, when a virtual machine snapshot is created, vsanSparse "delta" objects get created.

2.2 Why is vsanSparse Needed?

The new vsanSparse snapshot format provides vSAN administrators with enterprise class snapshots and clones. The goal is to improve snapshot performance by continuing to use the existing redo logs mechanism but now utilizing an "in-memory" metadata cache and a more efficient sparse filesystem layout.

2.3 How Does vsanSparse Work?

When a vsanSparse snapshot is taken of a base disk, a child delta disk is created. The parent is now considered a point-in-time (PIT) copy. The running point of the virtual machine is now the delta. New writes by the virtual machine go to the delta but the base disk and other snapshots in the chain satisfy reads. To get current state of the disk, one can take the "parent" disk and redo all writes from "children" chain. Thus children are referred to as "redo logs". In this way, vsanSparse format is very similar to the earlier vmfsSparse format.

However snapshots created with the vsanSparse format on VirstoFS can operate at much closer base disk performance levels than vmfsSparse/redo log format, even as the length of snapshot chain

increases. More details on improved performance are provided in the performance section of this tech-note.

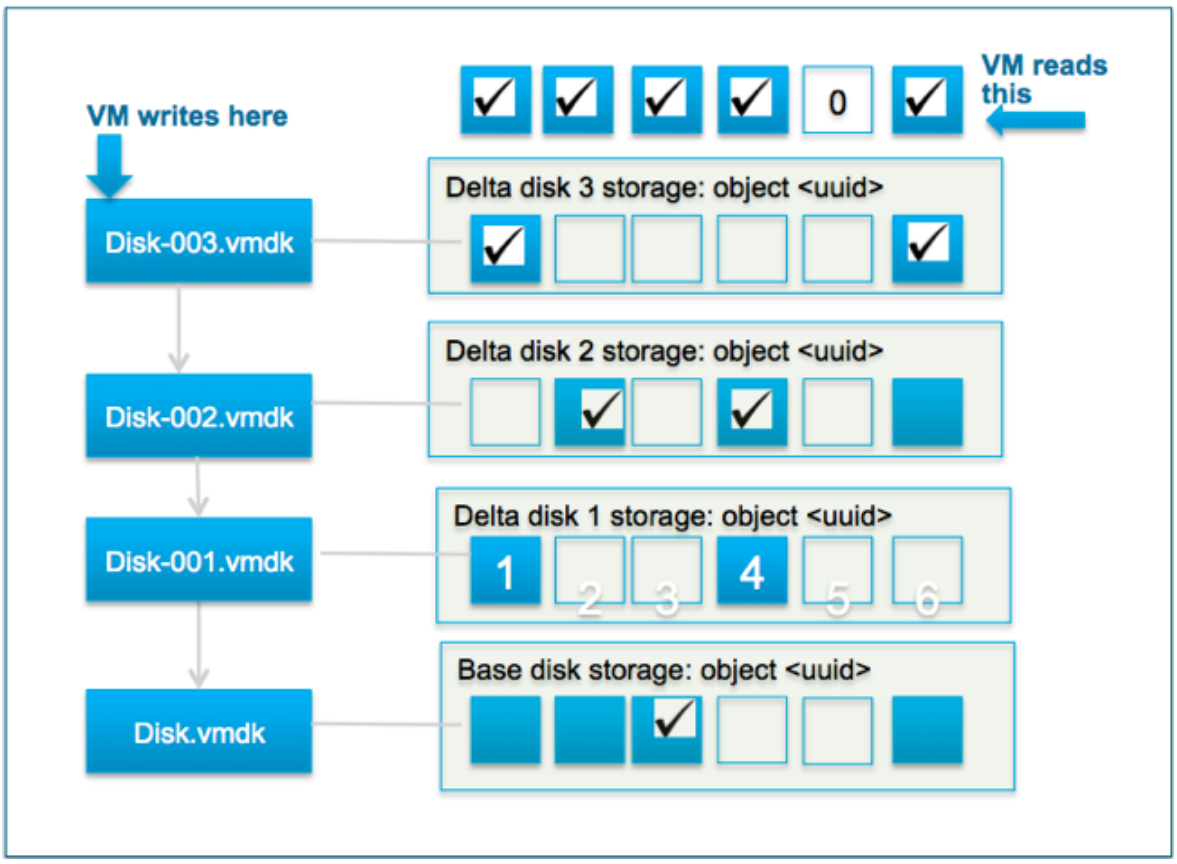
When discussing vSAN, we talk about objects. A delta disk (snapshot) object is made up of a set of grains, where each grain is a block of sectors containing virtual disk data. A VMDK object backs each delta. The deltas keep only changed grains, so they are space efficient.

In the diagram below, the Base disk object is called Disk.vmdk and is at the bottom of the chain. There are three snapshot objects (Disk-001.vmdk, Disk-002.vmdk and Disk-003.vmdk) taken at various intervals and guest OS writes are also occurring at various intervals, leading to changes in snapshot deltas.

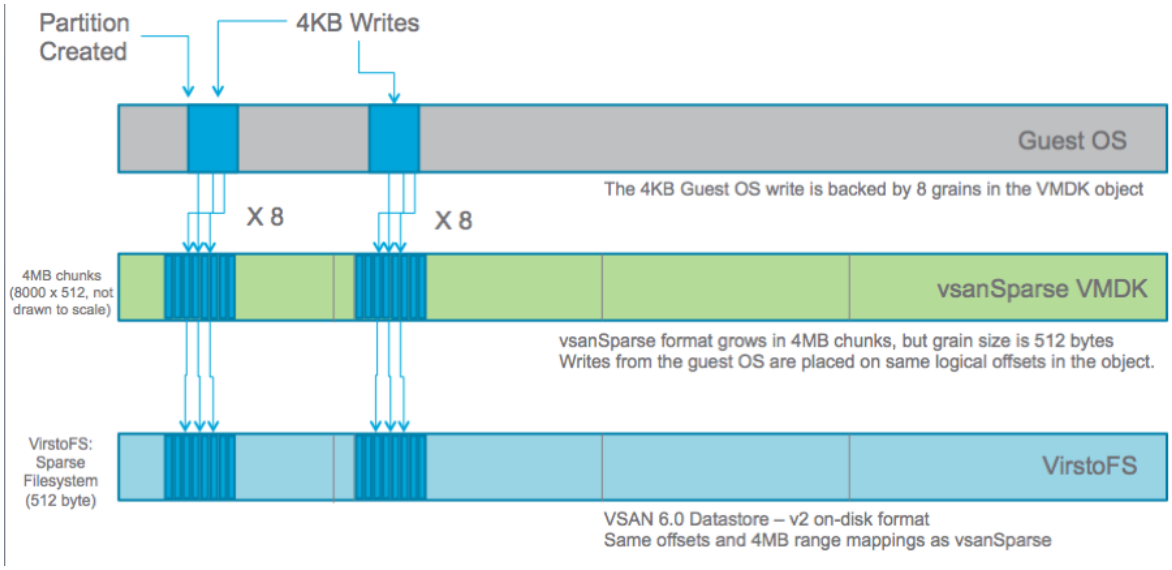
- Base object – writes to grain 1,2,3 & 5
- Delta object Disk-001 – writes to grain 1 & 4
- Delta object Disk-002 – writes to grain 2 & 4
- Delta object Disk-003 – writes to grain 1 & 6

A read by the VM will now return the following:

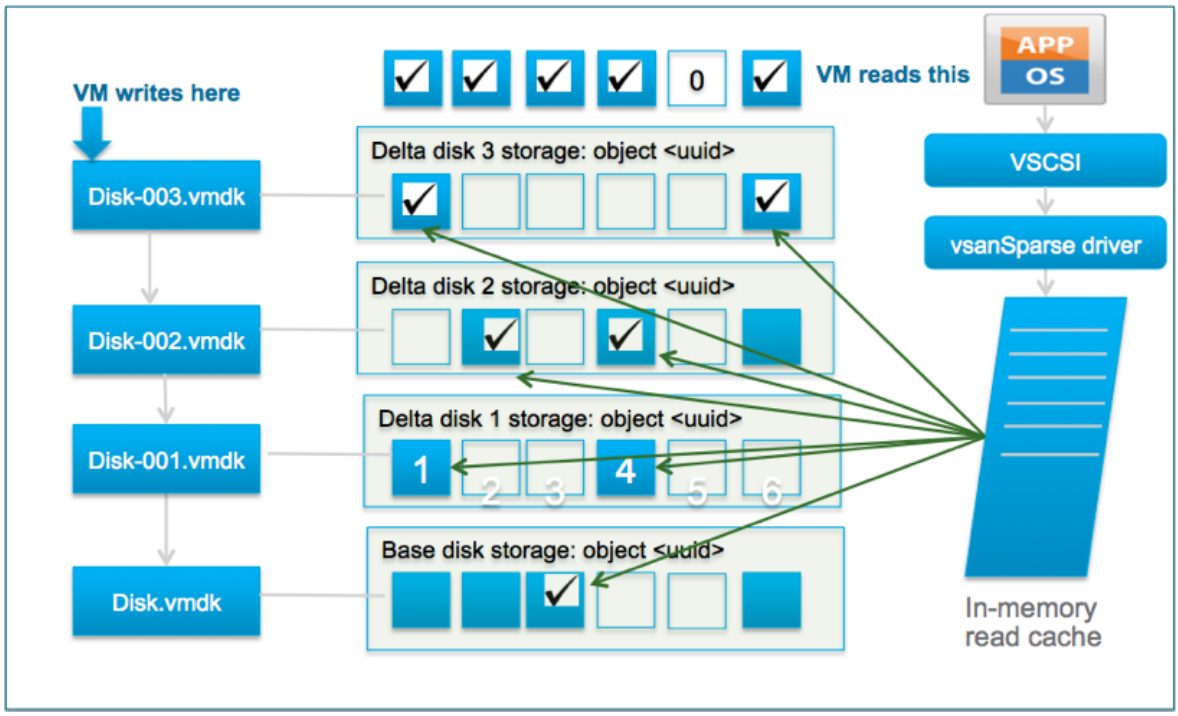
- Grain 1 – retrieved from Delta object Disk-003
- Grain 2 – retrieved from Delta object Disk-002
- Grain 3 – retrieved from Base object
- Grain 4 – retrieved from Delta object Disk-002
- Grain 5 – retrieved from Base object - 0 returned as it was never written
- Grain 6 – retrieved from Delta object Disk-003



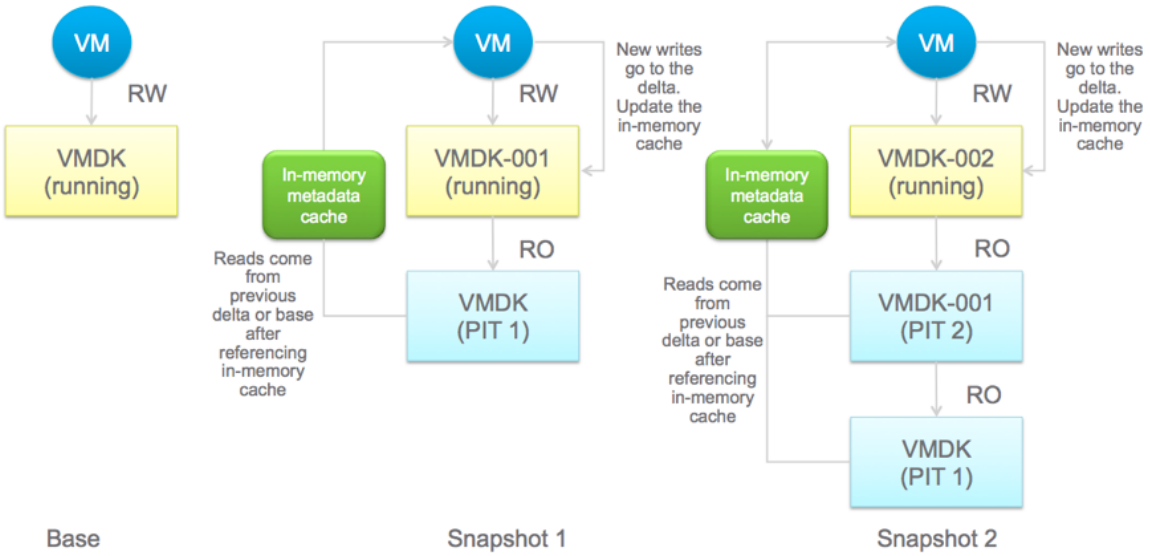
The new vsanSparse snapshot format and the underlying VirstoFS filesystem both use a 512-byte allocation unit size, which allows snapshots taken with the vsanSparse format to always remain sparse.



Consider the case when a snapshot has been taken of a virtual machine. When a guest OS sends a write to disk, the vsanSparse driver receives the write. Writes always goes to the top-most object in the snapshot chain. When the write is acknowledged, the vsanSparse driver updates its "in-memory" metadata cache, and confirms the write back to the guest OS. On subsequent reads, the vsanSparse driver can reference its metadata cache and on a cache hit, immediately locate the data block.



Reads are serviced from one or more of the vsanSparse deltas in the snapshot tree. The vsanSparse driver checks the "in-memory" metadata cache to determine which delta or deltas to read. This depends on what parts of the data were written in a particular snapshot level. Therefore to satisfy a read I/O request, the snapshot logic does not need to traverse through every delta of the snapshot tree, but can go directly to the necessary vsanSparse delta and retrieves the data requested. Reads are sent to all deltas that have the necessary data in parallel.



On a cache miss, however the vsanSparse driver must still traverse each layer to fetch the latest data. This is done in a similar way to read requests in so far as the requests are sent to ALL layers are send in parallel.

2.4 In-Memory Cache Considerations

The vsanSparse in-memory cache initially has “unknown” ranges. In other words, cache is cold. When there is a read request from an unknown range, a cache miss is generated. This range is then retrieved and cached for future requests. A cache miss increases the I/O latency.

Cache is in-memory only and is never committed to persistent storage. This raises the question about what happens when there is a host failure. A power failure, a reboot or simply a VM power off leads to cache erases. Therefore the next time the virtual disk is opened, the cache is cold (empty) and will be filled up as the VM generates I/O, with the first I/O paying ’cache fill’ latency penalty.

2.5 How Does an Administrator Utilize vsanSparse?

A vSphere administrator manages vsanSparse snapshots in exactly the same way that previous virtual machine snapshots were managed. There is no selection process to choose vsanSparse format. If the underlying is storage is vSAN, if the on-disk format is v2, and if there are no older vmfsSparse/redo log format snapshots on the virtual machine, vsanSparse format snapshots will be automatically used.

This can easily be verified by checking the snapshot descriptor file in the home namespace of the virtual machine. The type of snapshot will be shown as “VSANSPARSE”:

```
> cat ch-vsan-desktop-000001.vmdk
 # Disk DescriptorFile
 version=4
 encoding="UTF-8"
 CID=0c4e9289
 parentCID=34cbbf0c
 isNativeSnapshot="no"
 createType="vsanSparse"
 parentFileNameHint="ch-vsan-desktop.vmdk"
 # Extent description
 RW 209715200 VSANSPARSE "vsan://7c110055-06d0-fd51-28dc-001517a69c72"

 # The Disk Data Base
 #DDB

 ddb.longContentID = "7000cbcbd7b039455788be5b0c4e9289"
```

3. vsanSparse Considerations

There are a number of considerations for administrators who plan to use vsanSparse format snapshots. These are outlined here.

3.1 Homogeneous Snapshot Types

Administrators cannot mix vmfsSparse and vsanSparse snapshot format in a chain. All disks in a vsanSparse chain need to be vsanSparse (except the base disk). As an aside, administrators cannot create linked clones of a virtual machine with vsanSparse snapshots on datastores other than a (v2) VirstoFS on-disk format VSAN datastore, e.g. VMFS, NFS, (v1) VMFS-L on-disk format VSAN datastore.

If a virtual machine has existing vmfsSparse/redo log based snapshots, it will continue to get vmfsSparse/redo log based snapshots until the user consolidates and deletes all of the current snapshots.

3.2 Requirements and Limitations

To be able to use vsanSparse, the following requirements must be met:

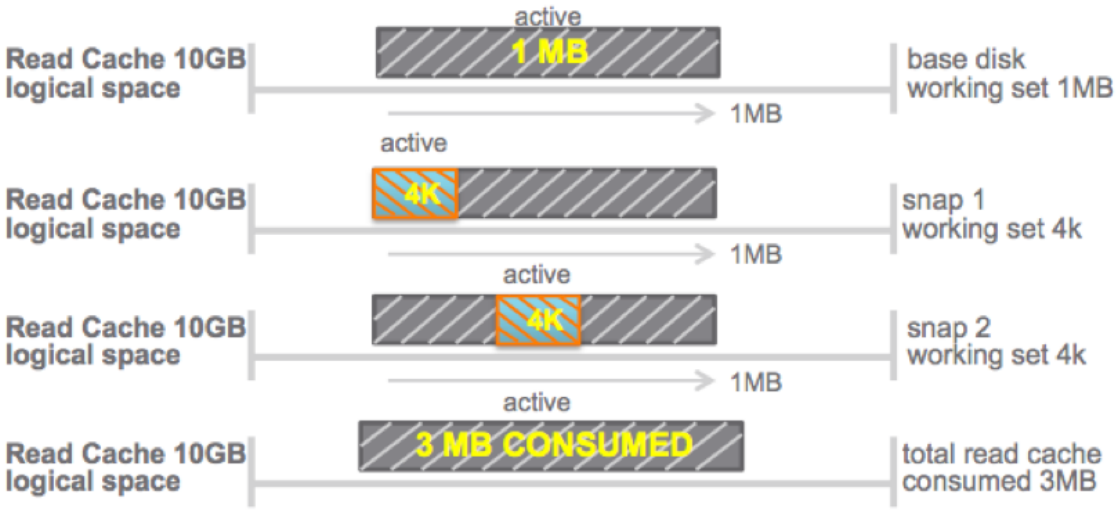
- VSAN 6.0 only
- On-disk format v2 only (VirstoFS)
- No existing vmfsSparse/redo log format snapshots

3.3 Read Cache Reservation Consumption Considerations

Intensive use of snapshots on hybrid vSAN 6.0 hybrid deployments may lead to higher than normal consumption of read cache resources. This may result in other workloads becoming temporarily cache starved, and/or snapshots themselves to perform poorly, depending on the scenario.

It is important to understand how read cache is allocated to fully explain this issue. Read cache on vSAN is allocated on a 1MB chunk basis. This is because we expect subsequent reads to be proximal to the previous read, and thus we ensure a read cache hit. Consider the diagram below. There is 1MB of read cache allocated to the virtual machine. A snapshot is taken of the virtual machine, “snap 1”, and a single new 4KB read is initiated by the guest OS. 1MB of read cache is allocated to “snap 1” to accommodate this read. Now consider a new snapshot, “snap 2”, taken against the same virtual machine. Another single, new 4KB read operation is initiated by the guest OS, but it is not proximal to the previous read. Therefore another 1MB of read cache is allocated. There is now a total of 3MB of read cache allocated for this VM and its two snapshots.

This is all fine if the subsequent reads are somewhat proximal in nature compared to the previous read which caused the read cache to be allocated. But if the reads are extremely random in nature, the current 1MB of read cache may not be utilized, leading to additional read cache usage.



Each disk snapshot creates a vSAN object, which in turn consumes read cache reservation resources. If the read cache is sized too small, or the I/O profile is not conducive to using the co-located blocks of allocated read cache, performance may drop as the snapshot chain gets above 16 or more.

This does not affect all-flash vSAN (AF-VSAN) as it does not use the caching layer for reads. The flash devices in the capacity layer satisfy reads.

3.4 Monitoring vSAN Read Cache

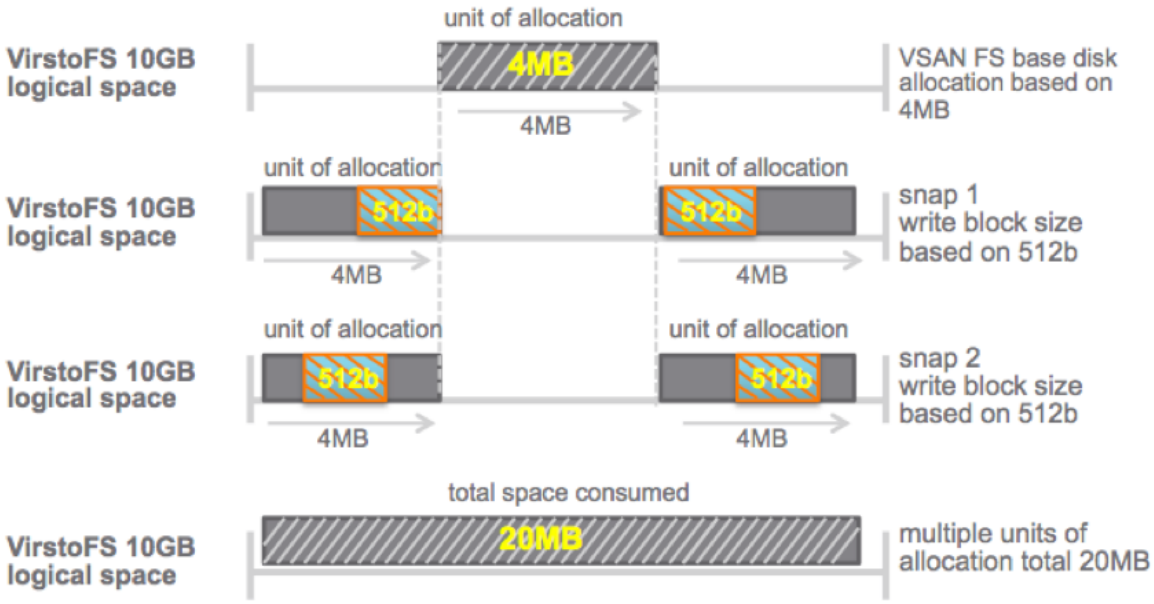
The Ruby vSphere Console (RVC) command vsan.whatif_host_failures can be used to monitor how much read cache is currently being consumed. If using snapshots regularly with larger snapshot chains, consider proactively checking read cache consumption with this command. Here is an example of such a command. The amount of free read cache in the "Usage right now" column of RC reservation is what should be monitored:

```
/localhost/IE-VSAN-DC/computers> ls
 0 VSAN60 (cluster): cpu 109 GHz, memory 328 GB
 /localhost/IE-VSAN-DC/computers> vsan.whatif_host_failures 0
 Simulating 1 host failures:

 +-----------------+-----------------------------+-----------------------------------+
 | Resource        | Usage right now             | Usage after failure/re-protection |
 +-----------------+-----------------------------+-----------------------------------+
 | HDD capacity    |  67% used (1074.02 GB free) |  95% used (118.93 GB free)        |
 | Components      |   1% used (35632 available) |   1% used (26632 available)       |
 | RC reservations |   0% used (521.57 GB free)  |   0% used (391.17 GB free)        |
 +-----------------+-----------------------------+-----------------------------------+
```

3.5 vSAN Datastore Capacity Consumption Consideration

As mentioned previously, creating snapshots on vSAN creates a vSAN object, which in turn consumes VSAN datastore capacity. These snapshot are sparse objects, and thinly provisioned. Assuming a worst case scenario when a guest OS is randomly writing to disk, and the VM's running point is a snapshot delta, unusual and rapid capacity consumption may result, in excess of what one would normally expect. Depending on the scenario, this may cause a surprise out-of-space condition in addition to high capacity consumption. This once again relates to the allocation mechanism on vSAN. The following diagram may help to explain the behavior.



Logical units of space on the vSAN (v2) on-disk format are allocated in 4MB chunks. However vSAN writes are done in 512-byte blocks. Take the above example where 4MB of disk space is allocated to the base disk to accommodate a write by the guest OS. Then a virtual machine snapshot ("snap 1") is taken. Next, with the running point of the virtual machine on "snap 1", there are two new 512-byte random writes made by the guest OS that requires two new 4MB chunks to be allocated from the VSAN datastore (VirstoFS). Then a new snapshot of the virtual machine is taken, "snap 2", and once again there are two new 512-byte random writes which requires another two new 4MB chunks to be allocated from the VSAN datastore. So with a total of 2KB of snapshot writes, 20MB of capacity has been allocated to his virtual machine and its snapshots.

Please keep in mind that what we are showing here is a worst-case scenario. We are assuming a workload that does 512 bytes writes, evenly spaced over every 4MB of disk space. Note that there is no proximal or sequential writes in this example. In this case, 4MB of disk space consumed for per every 512 bytes. This is not normal or common behavior for a workload. There is also a worse case assumption here that no additional writes hit the same 4MB chunks until the test ends. This is an even harsher (and less realistic) assumption.

Snapshots will never grow larger than the size of the original base disk. The size of the delta will be dependent on the number of changes made since the snapshot was taken.

This behavior applies to both all flash and hybrid versions of VSAN 6.0.

3.6 Monitoring vSAN Capacity

The VSAN datastore capacity can be checked from the vSphere Web Client. Alternatively, the RVC command vsan.disks_stats can be used to monitor VSAN datastore capacity. If using snapshots regularly with I/O intensive virtual machines, consider proactively checking datastore capacity with this command.

```
/localhost/IE-VSAN-DC/computers> vsan.disks_stats 0
2015-03-12 11:08:44 +0000: Fetching VSAN disk info from cs-ie-h04.ie.local (may take a moment) ...
2015-03-12 11:08:44 +0000: Fetching VSAN disk info from cs-ie-h01.ie.local (may take a moment) ...
2015-03-12 11:08:44 +0000: Fetching VSAN disk info from cs-ie-h02.ie.local (may take a moment) ...
2015-03-12 11:08:44 +0000: Fetching VSAN disk info from cs-ie-h03.ie.local (may take a moment) ...
2015-03-12 11:08:47 +0000: Done fetching VSAN disk infos
+--------------------------------------+--------------------+-------+------+-----------+------+----------+---------+
|                                      |                    |       | Num  | Capacity  |      |          | Status  |
| DisplayName                          | Host               | isSSD | Comp | Total     | Used | Reserved | Health  |
+--------------------------------------+--------------------+-------+------+-----------+------+----------+---------+
| naa.600508b1001c61cedd42b0c3fbf55132 | cs-ie-h01.ie.local | SSD   | 0    | 186.27 GB | 0 %  | 0 %      | OK (v2) |
| naa.600508b1001c3ea7838c0436dbe6d7a2 | cs-ie-h01.ie.local | MD    | 19   | 136.44 GB | 68 % | 67 %     | OK (v2) |
| naa.600508b1001ccd5d506e7ed19c40a64c | cs-ie-h01.ie.local | MD    | 15   | 136.44 GB | 76 % | 75 %     | OK (v2) |
| naa.600508b1001c388c92e817e43fcd5237 | cs-ie-h01.ie.local | MD    | 33   | 136.44 GB | 72 % | 71 %     | OK (v2) |
| naa.600508b1001c79748e8465571b6f4a46 | cs-ie-h01.ie.local | MD    | 5    | 136.44 GB | 37 % | 37 %     | OK (v2) |
| naa.600508b1001c16be6e256767284eaf88 | cs-ie-h01.ie.local | MD    | 13   | 136.44 GB | 68 % | 67 %     | OK (v2) |
| naa.600508b1001c2ee9a6446e708105054b | cs-ie-h01.ie.local | MD    | 12   | 136.44 GB | 67 % | 66 %     | OK (v2) |
| naa.600508b1001c64816271482a56a48c3c | cs-ie-h01.ie.local | MD    | 15   | 136.44 GB | 73 % | 73 %     | OK (v2) |
+--------------------------------------+--------------------+-------+------+-----------+------+----------+---------+
| naa.600508b1001c64b76c8ceb56e816a89d | cs-ie-h02.ie.local | SSD   | 0    | 186.27 GB | 0 %  | 0 %      | OK (v2) |
| naa.600508b1001c0cc0ba2a3866cf8e28be | cs-ie-h02.ie.local | MD    | 15   | 136.44 GB | 65 % | 65 %     | OK (v2) |
| naa.600508b1001c19335174d82278dee603 | cs-ie-h02.ie.local | MD    | 14   | 136.44 GB | 69 % | 68 %     | OK (v2) |
| naa.600508b1001cb2234d6ff4f7b1144f59 | cs-ie-h02.ie.local | MD    | 13   | 136.44 GB | 72 % | 13 %     | OK (v2) |
| naa.600508b1001c07d525259e83da9541bf | cs-ie-h02.ie.local | MD    | 12   | 136.44 GB | 66 % | 59 %     | OK (v2) |
| naa.600508b1001ca36381622ca880f3aacd | cs-ie-h02.ie.local | MD    | 23   | 136.44 GB | 62 % | 59 %     | OK (v2) |
+--------------------------------------+--------------------+-------+------+-----------+------+----------+---------+
| naa.600508b1001c9c8b5f6f0d7a2be44433 | cs-ie-h03.ie.local | SSD   | 0    | 186.27 GB | 0 %  | 0 %      | OK (v2) |
| naa.600508b1001ceefc4213ceb9b51c4be4 | cs-ie-h03.ie.local | MD    | 13   | 136.44 GB | 69 % | 54 %     | OK (v2) |
| naa.600508b1001c1a7f310269ccd51a4e83 | cs-ie-h03.ie.local | MD    | 16   | 136.44 GB | 76 % | 60 %     | OK (v2) |
| naa.600508b1001c2b7a3d39534ac6beb92d | cs-ie-h03.ie.local | MD    | 14   | 136.44 GB | 75 % | 74 %     | OK (v2) |
| naa.600508b1001cd259ab7ef213c87eaad7 | cs-ie-h03.ie.local | MD    | 18   | 136.44 GB | 60 % | 58 %     | OK (v2) |
| naa.600508b1001cb11f3292fe743a0fd2e7 | cs-ie-h03.ie.local | MD    | 12   | 136.44 GB | 61 % | 60 %     | OK (v2) |
| naa.600508b1001c9b93053e6dc3ea9bf3ef | cs-ie-h03.ie.local | MD    | 12   | 136.44 GB | 76 % | 76 %     | OK (v2) |
+--------------------------------------+--------------------+-------+------+-----------+------+----------+---------+
| naa.600508b1001c29d8145d6cc1925e9fb9 | cs-ie-h04.ie.local | SSD   | 0    | 186.27 GB | 0 %  | 0 %      | OK (v2) |
| naa.600508b1001c846c000c3d9114ed71b3 | cs-ie-h04.ie.local | MD    | 15   | 136.44 GB | 69 % | 60 %     | OK (v2) |
| naa.600508b1001c6a664d5d576299cec941 | cs-ie-h04.ie.local | MD    | 12   | 136.44 GB | 63 % | 55 %     | OK (v2) |
| naa.600508b1001c4b820b4d80f9f8acfa95 | cs-ie-h04.ie.local | MD    | 13   | 136.44 GB | 73 % | 66 %     | OK (v2) |
| naa.600508b1001cadff5d80ba7665b8f09a | cs-ie-h04.ie.local | MD    | 29   | 136.44 GB | 52 % | 50 %     | OK (v2) |
| naa.600508b1001c258181f0a088f6e40dab | cs-ie-h04.ie.local | MD    | 12   | 136.44 GB | 74 % | 74 %     | OK (v2) |
| naa.600508b1001c51f3a696fe0bbbcb5096 | cs-ie-h04.ie.local | MD    | 13   | 136.44 GB | 72 % | 72 %     | OK (v2) |
+--------------------------------------+--------------------+-------+------+-----------+------+----------+---------+
/localhost/IE-VSAN-DC/computers>
```

4. Conclusions

VMware's advice is to proactively monitor the VSAN Datastore capacity and read cache consumption on a regular basis when using snapshots intensively on vSAN.

4.1 Snapshot Retention Period

VMware does not make the same recommendations for vsanSparse snapshots retention period as we do for vmfsSparse/redo log format. In other words, you do not need to limit vsanSparse snapshot usage to short periods of time (24 - 72 hours). However we would recommend regularly checking the read cache usage and the VSAN Datastore capacity when using snapshots for long periods of time, and if either of these appear to be impact by the scenarios outlined earlier, it would be good practice to not use snapshots unnecessarily, freeing up resources for other uses.

4.2 Snapshots Per Chain

VMware supports the full maximum chain length of 32 snapshots when vsanSparse snapshots are used. It is advisable however to not exceed 16 snapshot deltas per chain when running very random workloads as performance degradation has been noticed. Once again, VMware recommends checking the read cache consumption with the command provided previously to see if the number of snapshots could be contributing to the performance issue. If it does appear to be a read cache depletion issue, consider consolidating the number of snapshots in the snapshot chain to replenish the read cache. Another option of course is to add additional cache resources to your vSAN hosts.

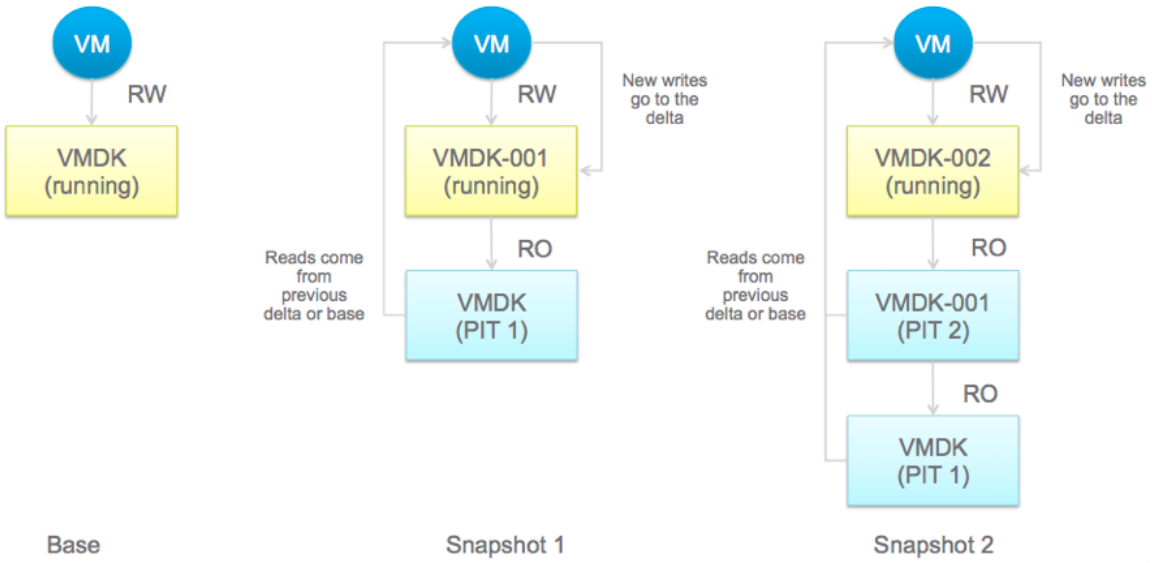
5. Appendix A: Other Snapshot Formats

.VMware supports multiple different snapshot formats. The type of snapshot format chosen is dependent on many factors, including underlying storage and VMDK characteristics.

5.1 vmfsSparse

vmfsSparse, commonly referred to as the redo log format, is the original snapshot format used by VMware. It is the format used on VMFS, NFS (without VAAI-NAS) and vSAN 5.5. The vmfsSparse snapshots use an allocation unit size of 512 bytes. As we shall see later, this small allocation unit size can be a cause for concern with certain storage array implementations.

When a snapshot is taken of a base disk using the **redo log** format, a child delta disk is created. The parent is then considered a point-in-time (PIT) copy. The running point of the virtual machine is now the delta. New writes by the virtual machine go to the delta, but the base disk or other snapshots in the chain satisfy reads.

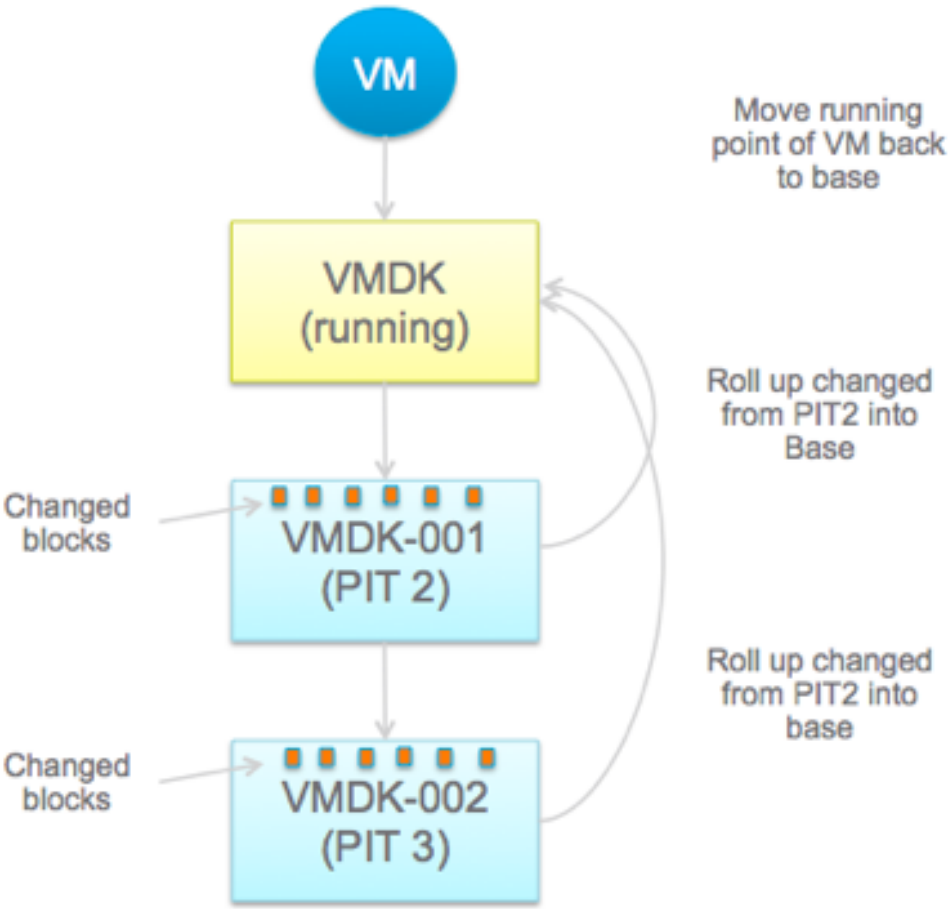


When a **consolidate** operation is needed, in other words there is a wish to roll up all of the changes from a point-in-time delta (PIT1) into (PIT2), we need to **redo** all the changes into the PIT1 delta and update the PIT2 delta, as well as change the running point of the VM back to the PIT2 VMDK.

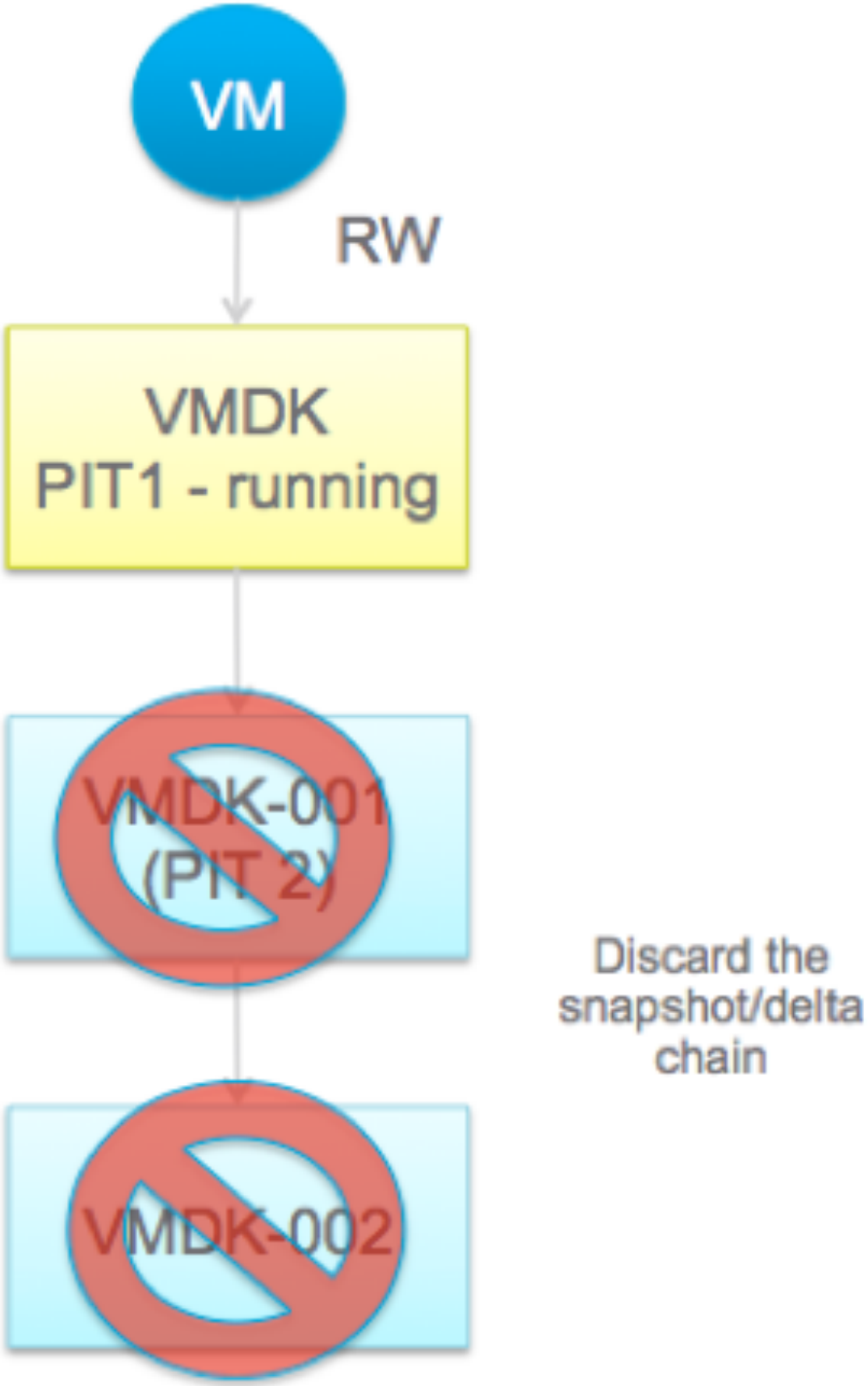
If you wanted to consolidate everything into the base disk, you could do this by consolidating each snapshot, one at a time like we mentioned. There is of course another way to do it by consolidating the whole chain (delete-all).

With a very long chain with many changes, it can take a considerable effort to redo all of the changes in each of the snapshots in the chain to the base disk. This is especially true when there are many snapshot deltas in the chain. Each delta's set of changes needs to be committed in turn.

Another consideration is when a whole chain is consolidated and the base disk is thinly provisioned, it may require additional space as snapshots changes are merged into the base disk, and this base disk may grow in size.



One final item to highlight with the redo log format is the **revert** mechanism. This is quite straightforward with redo logs as administrators can simply discard the chain of deltas and return to a particular delta or base disk. In this example, the virtual machine was reverted to the point-in-time held on the base disk by simply discarding the snapshot deltas holding the changes made since the snapshots were taken:



One major concern with vmfsSparse/redo log snapshots is that they can negatively affect the performance of a virtual machine. Performance degradation is based on how long the snapshot or snapshot tree is in place, the depth of the tree, and how much the virtual machine and its guest operating system have changed from the time the snapshot was taken. Also, you might see a delay in the amount of time it takes the virtual machine to power-on**. VMware does not recommend running production virtual machines from snapshots using redo log format on a permanent basis**.

VMware also makes the recommendation that only 2-3 snapshots should be used in a chain. There are more best practices guidance around redo log based snapshots in VMware KB article 1025279.

One final item to note is that the vmfsSparse snapshot block allocation unit size is 512 bytes. This will become an important consideration when additional formats are discussed. For more information on snapshots, please refer to VMware KB article 1015180.

5.2 Space Efficient (SE) Sparse

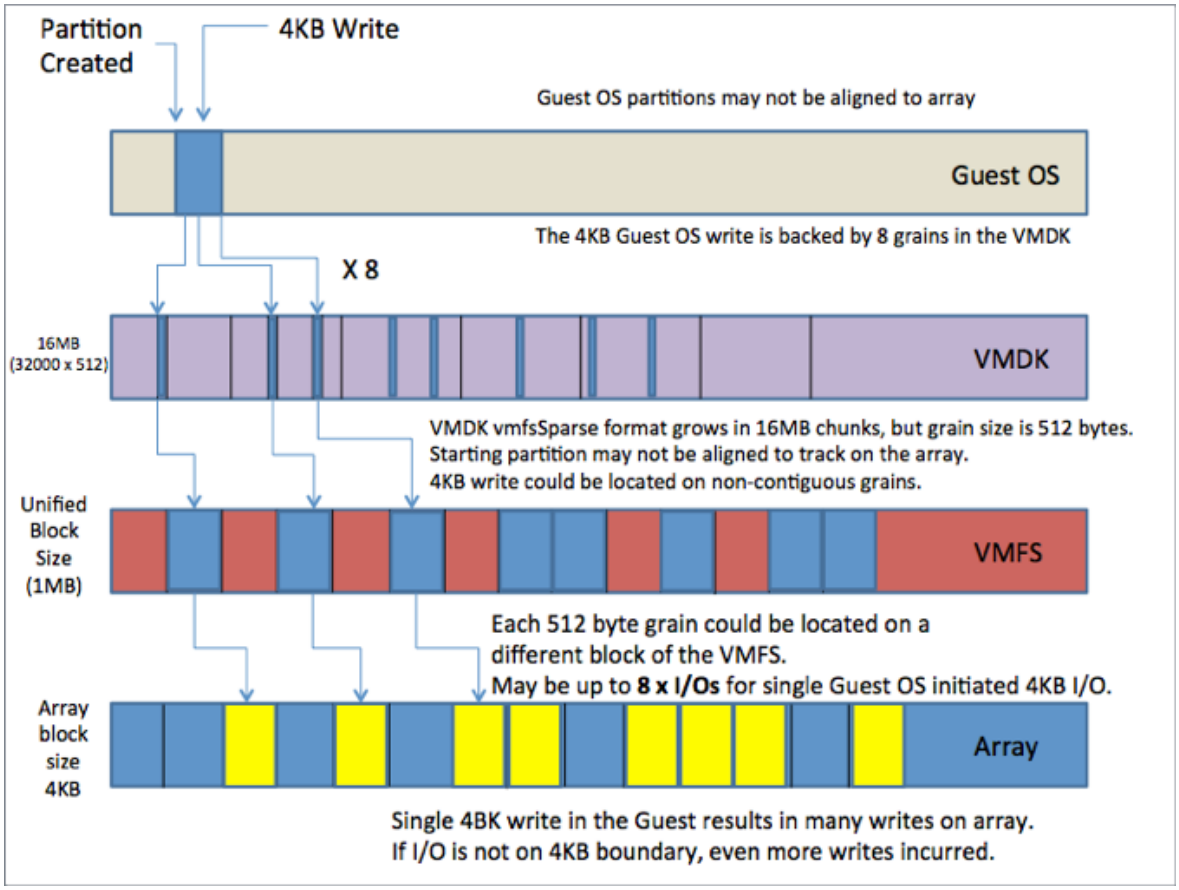
This snapshot format may be encountered on traditional storage, such as VMFS and NFS, when VMware Horizon View is deployed. Introduced in vSphere 5.1, the SE Sparse Disk format (SE standing for space efficient) introduces two new features to snapshots.

The first of these is the snapshot block size granularity in SE Sparse is now set to 4KB. This addresses a concern with the vmfsSparse format mentioned earlier. A single 4KB guest OS write, when running on a vmfsSparse snapshot, could result in significant *write amplification*. This single guest OS 4KB write could generate multiple a number of write operations on the back-end storage array. These diagrams should help to highlight this concern with the smaller 512-byte block size found in vmfsSparse/redo logs.

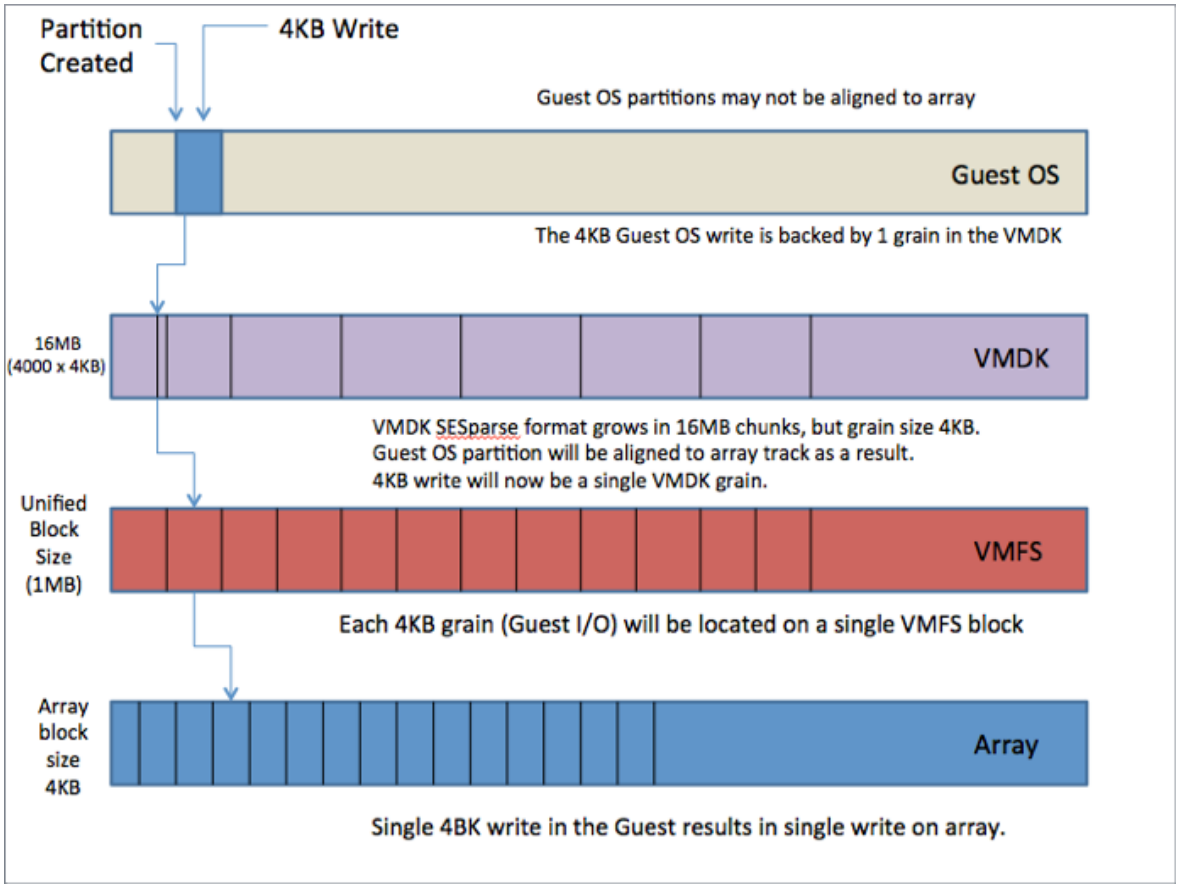
In the diagrams below, an example is shown where a single 4KB I/O is issued from the guest OS. The issue with vmfsSparse disks is that they are backed by block allocation unit sizes of 512 bytes. This means that in a worse case scenario, this 4KB I/O may involve 8 distinct 512-byte writes to different blocks on the underlying VMFS volume, and in turn lead to 8 I/Os issued to the array, what is termed in the industry as write amplification.

Although this is a worse case scenario, it is still plausible. In a case where a base disk has a database installed using raw disks in the guest OS, and then a vmfsSparse snapshot is taken of it, I/Os issued to update the database tables may lead to this write amplification behaviour described here**.**

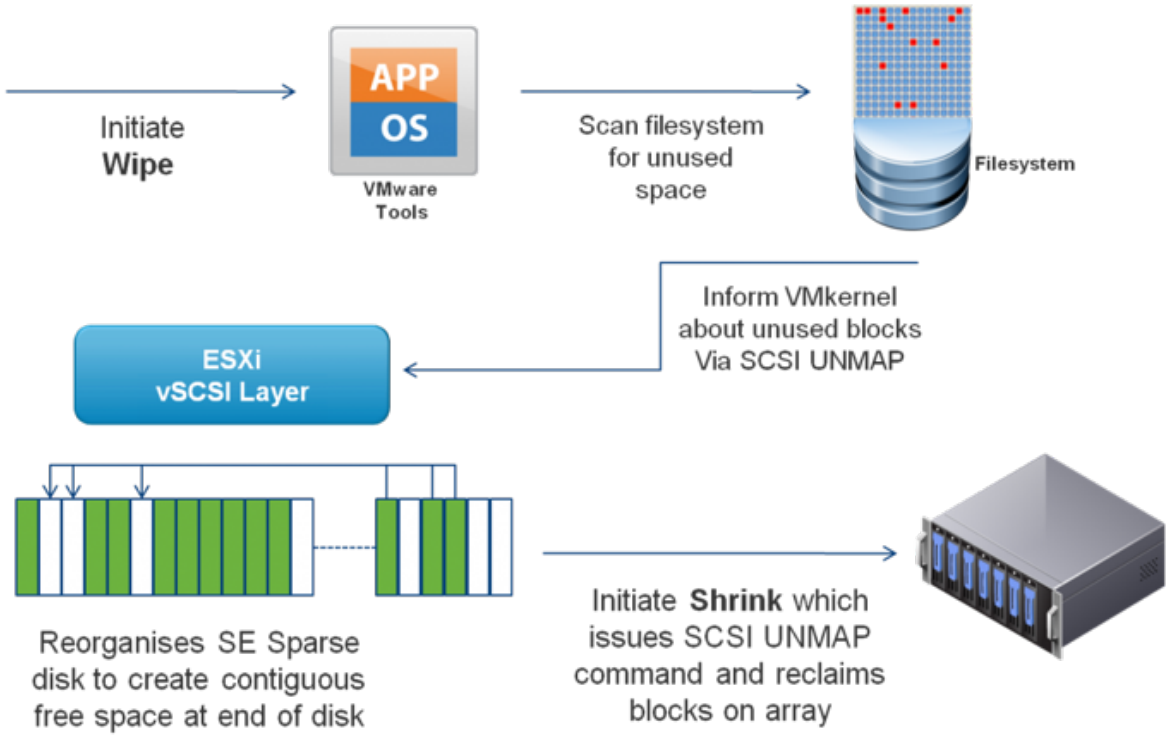
Let's look at the vmfsSparse behavior more closely:



Notice the write amplification shown above. This is where SE Sparse can help. Using SE Sparse with a new block allocation size of 4KB, the 4KB I/O from the Guest OS will use a single block in the VMDK, which in turn will use a single VMFS block, and generate a single I/O to the array.



The second enhancement to SE Sparse format is the ability to reclaim previously used space within the guest OS. This stale data is data that was previously written to, but is currently in unaddressed blocks in a file system/database. SE Sparse disks, through the introduction of a new shrink and wipe operation, allow disks based on these formats to shrink in size.



There are only two very specific use cases for SE Sparse Disks.

The scope was initially restricted in vSphere 5.1 to a VMware View use case when VMware View Composer uses “Linked Clones” for the rollout of desktops. VMware View desktops typically benefit from the new 4KB block allocation unit size as it addresses the partial write and alignment issues experienced by some storage arrays when the 512 bytes block allocation unit size found in the vmfsSparse format is used for linked clones.

SE Sparse Disks also gives far better space efficiency to desktops deployed on this virtual disk format since it has the ability to reclaim stranded space from within the Guest OS.

In vSphere 5.5, the scope of SE Sparse disk format was extended to include snapshots of VMDK disks that are larger than 2TB (the actual size is 2TB minus 512 bytes to be exact). vSphere 5.5 introduced support for VMDK size of up to 62TB, although vSAN 5.5 did not support this larger size. When a snapshot is taken of these large disks, the SE Sparse format is used for the snapshot.