Technical Report

Optimizing NetApp Storage for Microsoft System Center Data Protection Manager

Chris Lionetti, NetApp
February 2011 | TR-3900

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 MICROSOFT SYSTEM CENTER DATA PROTECTION MANAGER

The Microsoft® System Center Data Protection Manager (DPM) application is an integrated backup solution that can protect a back office–type installation or even the largest data centers. The DPM tool is designed to allow the continuing nightly (or even subhourly) backup of a collection of servers or client class machines. The DPM server backs up hosts via Ethernet, and can also back up SAN-based or DAS-based sources. Consequently, the storage that the DPM server uses to store its backups must be block-level storage and cannot be CIFS. This DPM storage should either be DAS- or SAN-based storage.

The scope of this document is rigidly defined to cover current OSs and applications (see Table 1).

Table 1) Scope definition.

Item	In Scope	Out of Scope
System Center DPM	DPM 2010	DPM 2007 and earlier
Operating systems	Windows 2008 R2	Windows 2008 or earlier
Windows® PowerShell	Version 2.0 and newer	Previous versions
Data ONTAP®	Version 8.0 and newer	Previous versions

1.1 HOW DPM READS A CLIENT DATASET

To set up DPM-based services on a host, you need to follow a two-step process to protect that server. The first step is to install the agent on the host with the proper credentials to allow the DPM server to connect. The second step is to allocate the storage space on the DPM server to store these backups. A wizard exists within the DPM tool to automate both of these tasks from the DPM console, which accelerates the ability to protect many servers.

NETWORK CONNECTIONS

Once the connection between the DPM server and the host to be protected is established, the default network settings on the host are used to determine which NIC from the host and which NIC on the DPM server will be used to transfer the backup. If multiple independent links exist between the host and the DPM server, network settings such as gateway metric or default network will determine the appropriate network to use. For this reason, a DPM server should have adequate network bandwidth available on all connected adaptors. If multiple Gigabit Ethernet (GbE) connections to a production network exist, care should be taken that individual links are not overloaded. The DPM Agent has no method to load balance traffic among multiple adaptors. This load balancing limitation and the ratio of many clients to a single DPM server may easily justify a 10GbE connection for the DPM server over multiple GbE connections.

PROCESSOR AND MEMORY USAGE OF THE DPM SERVER

The DPM server itself is commonly disk or network bound, and rarely CPU or memory bound. The DPM server has minimum requirements of a single CPU core and 3GB of RAM.

The recommendation for the best experience is two cores and 4GB of RAM. Because servers today commonly have no fewer than eight cores even in the least expensive 1 rack U server, it is a common approach for optimal usage to deploy the DPM application within a virtual machine instead of directly onto a physical machine. This movement from a physical to a virtual machine infrastructure also facilitates additional abilities such as live migration or site-to-site failover. For more information, see section 2: Optimal Storage Layout.

STORAGE CAPACITY REQUIREMENTS

DPM stores its data in a rather unique way to give you a reconstituted full backup that is up to date each night, while also maintaining historical backups for a specified time frame. To this end, the rule of thumb is to deploy three times the amount of storage that you need to protect. Depending on your servers this may allow you to maintain 30 days of daily change or even up to 90 days of change. As an example, suppose a server has a drive that is 1TB in size and is 65% full, that is, it contains 650GB of data. The percentage of the growth rate of data and the rate of change of data needs to be determined. The change rate that needs to be known is different than the growth rate, since modification of a file is different from simply adding a file.

When a host is added and that host has a 1TB drive that needs to be protected, the DPM server will create a partition to protect the drive. The size of this partition is the used space plus 10% of the drive size, which in this case is 715GB of space. Additionally, the DPM server will create a recover point drive to maintain the change rate that is 10% of the full drive space, in this case an additional 100GB. As data is added to the protected 715GB drive, and the free space starts to decrease, the DPM server will add more space to that partition using the standard dynamic disk spanned volume method. As files are modified on the protected 715GB drive, the original data is first copied to the recover point drive before it is overwritten on the protected 715GB drive. If the recover point drive approaches a full threshold, it will use the same method to expand itself as a spanned volume does.

600-VOLUME LIMIT

Each protected resource requires two partitions, and a protected server will likely contain many protected resources. DPM is limited to supporting only 600 partitions; as such, between 100 and 128 servers should be considered a maximum number of supported servers (per DPM server). If your protected servers contain many possible protected resources, this maximum server limit should be reduced appropriately.

Since a protected Resource Partition might expand multiple times, consuming more than its fair share of that 600-volume limit, a PowerShell command exists that allows a spanned volume to be moved from its existing location across many volumes to a new space that is a single unspanned volume. This move process allows resources to be freed and prevents DPM from reaching the 600-volume limit. For information on the commands needed to combine a spanned volume back to a nonspanned volume, see the Appendix: Scripts.

80TB LIMIT OF PROTECTED STORAGE

DPM enables no more than 80TB to be protected by a single DPM server. This should be considered when attaching servers that over time may grow significantly. If more than 80TB of data needs to be protected, a horizontal scale strategy should be employed that allows multiple DPM servers. A large data center infrastructure may require a farm of DPM servers. Such farms are much easier to manage and optimize if deployed in a virtual infrastructure.

ADVANCED ADD-IN FEATURES FOR SQL, HYPER-V, SHAREPOINT, AND EXCHANGE SERVER

The DPM application also has inherent knowledge of how data is organized in these specialized applications. In the case of SQL®, for example, each large database gets its own protection volume and recover point volume, but an option exists to allow smaller databases from a single SQL Server to coexist in a single combined protection volume and recover point volume.

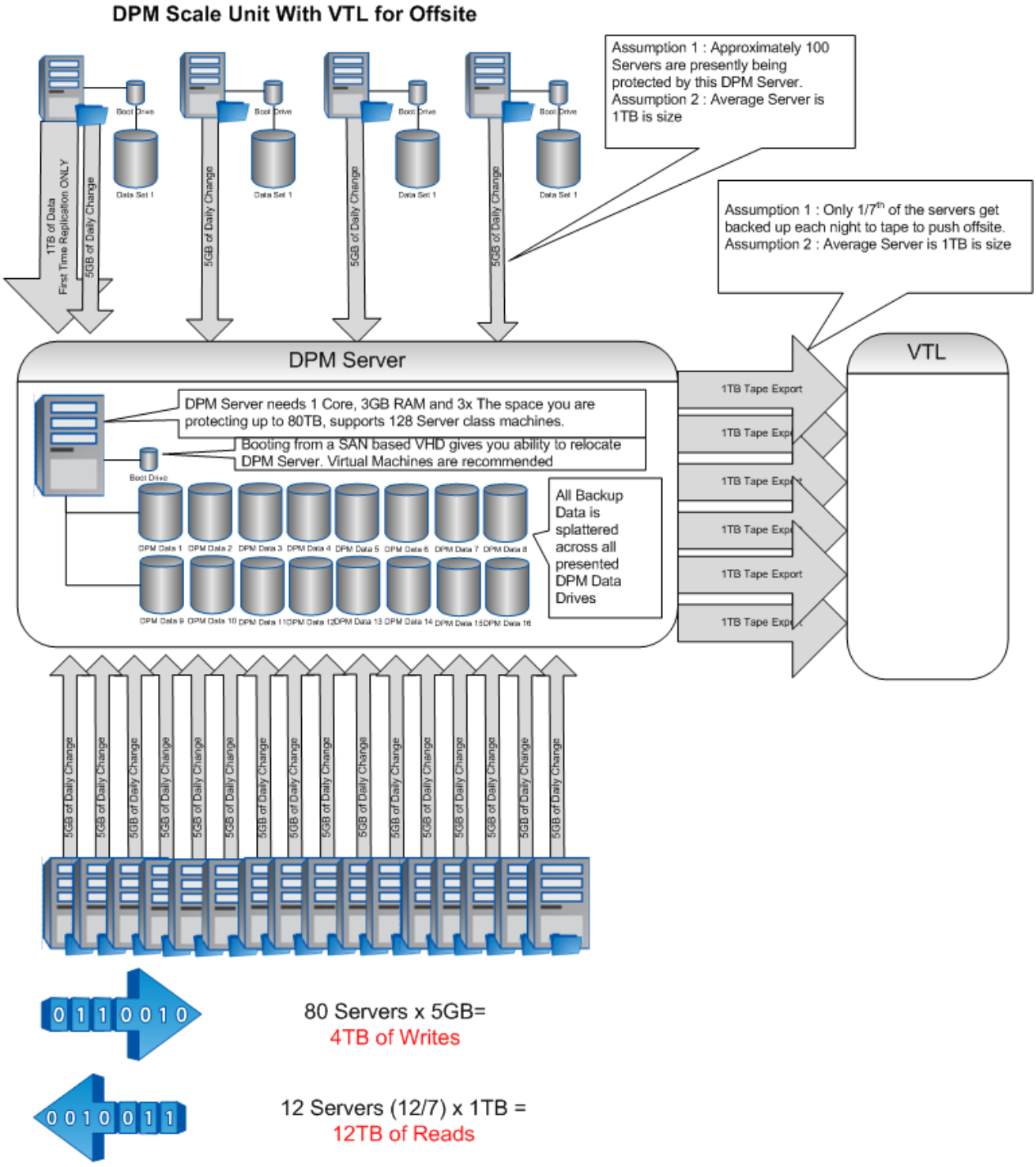
1.2 DISASTER RECOVERY METHODS IN DPM

INTEGRATION OF TAPE

DPM has a method to allow a protected server to be exported to tape. This method allows the contents of a single protected server to be exported to a tape or virtual tape. DPM also allows you to mount those tapes to an alternate DPM server and import the data. This method allows data to be transferred effectively between sites, but requires a full backup of the protected server's dataset from the DPM server each time a tape export is required. This method does, however, allow the exportation of a single dataset from the DPM database. In the case of 40 x 2TB LUNs presented to the DPM server, a single backup of a single server may exist over 12 of those drives in pieces. If those 40 x drives are exposed to a secondary site with a new DPM server, there exists NO method to import the data, and the recover points are lost.

Figure 1 illustrates an 80TB DPM server with an individual server size of 1TB and a daily change rate of 5GB per server. In this example, the normal DPM traffic would be only 80 x 5GB = 4TB. To export 1/7 of these servers each night, you would have to back up 12 servers a night, which constitutes 12TB of backup data.

Figure 1) DPM integration with tape.



DPM SERVERS BACKING UP DPM SERVERS

DPM permits a DPM server to back up another DPM server. This allows you to survive the loss of a single DPM server without losing the ability to restore from a given recover point. This method of DPM server chaining allows the secondary DPM server to either be local or be deployed remotely. In the case of deploying the second DPM server remotely, latency between sites may affect the ability for DPM to keep

up with the change rate. The DPM servers can, however, turn on network compression for in-flight compression, which can alleviate this problem. However, care should be taken to make sure that the site-to-site link can effectively be filled with the given latency.

As the latency between the chained DPM servers increases, the ability for the DPM server to keep the pipe filled diminishes.

Consider also the impact of the in-flight compression on the CPU of the individual DPM servers participating in the compressed data tunnel. This may require an extra CPU within a virtual machine, or may require proper sizing if physical machines are done from the onset of creating or building a DPM server.

1.3 SITE-TO-SITE FAILOVER OF A DPM SERVER

Another method to support disaster recovery is to allow a DPM server to be virtually relocated. In this method, the primary DPM server would exist within a virtual machine. The virtual machine could be protected via SnapManager® for Hyper-V™ and duplicated from site to site using SnapMirror® technology. In this scenario, the following process would be followed.

- A DPM server, configured with its data drives residing on NetApp® storage, would run on its regular schedule and back up its clients.
- The NetApp storage controller could be configured to then start the deduplication procedure if required to dedupe any added data from previous DPM operations.
- The Hyper-V host would then initiate a SnapManager for Hyper-V (SMHV) call to orchestrate a VSS-enabled backup on the VM running the DPM server.
- The SMHV would use a prescript to stop the DPM service, which would flush the data drives, and a postscript to restart the DPM service, which would return the DPM server to operation.
- SMHV would also execute the SnapMirror update to make sure that the recently created Snapshot® copies are available on the remote site.

This method allows two types of failovers to the remote site. Note that the virtual machine cannot be running in both locations at once, so if you promote the virtual machine on the DR site, you must first remove the VM from the primary site.

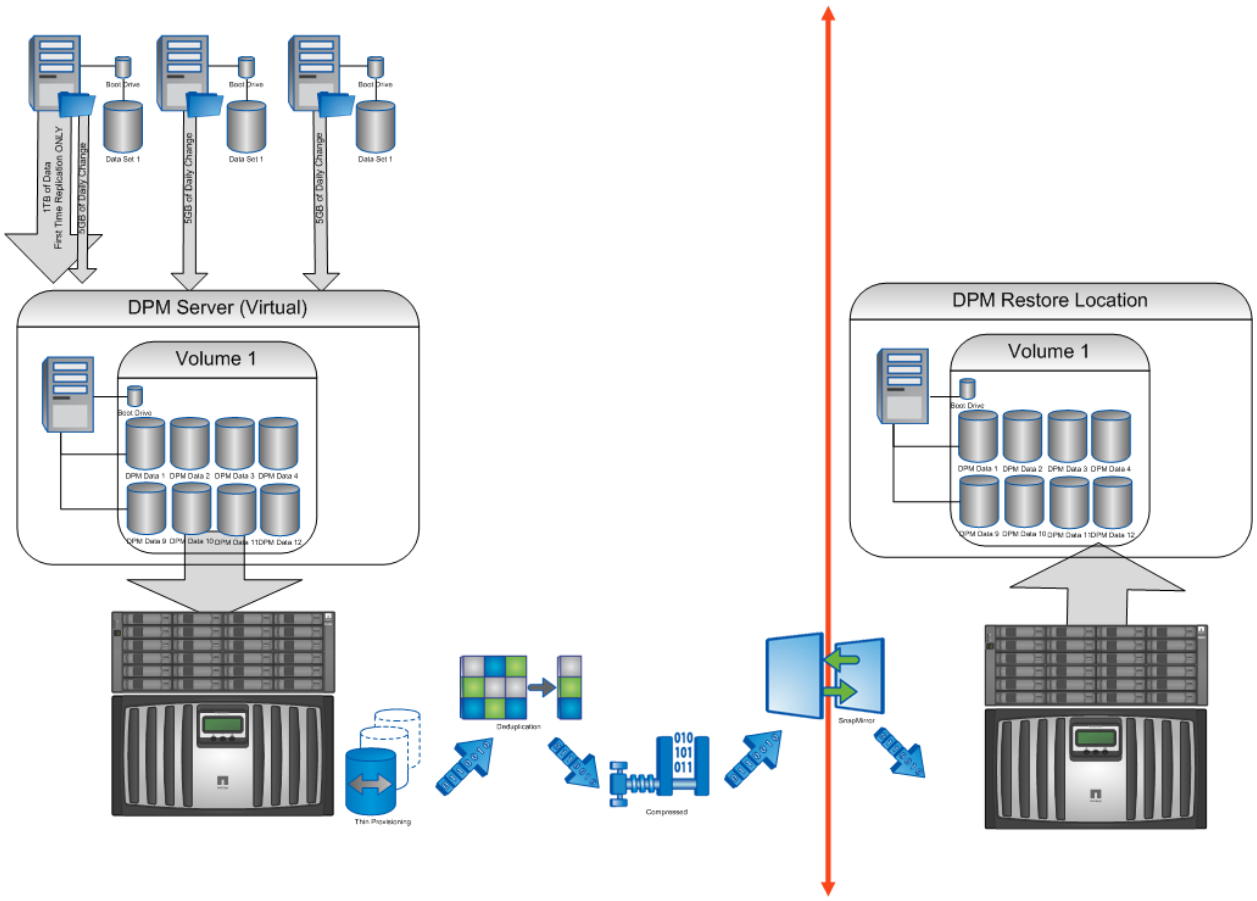
RESTORING VM FROM MOST RECENT DATA

You could choose during a disaster to simply bring up the most recent image of the boot and data drives for the DPM server, which have the advantage of only being a short time frame behind. This short time frame will depend on a number of factors, from the rate of change to the distance between sites to the network bandwidth limits. The SnapMirror settings allow this to be automated to minutes, hours, days, and so on.

RESTORING VM FROM MOST RECENT KNOWN GOOD SNAPSHOT COPY

With this method, from the remote site you would revert to that known good flushed copy of the data. This would prevent the DPM server from becoming corrupt, but you might lose recently backed up data. This method is more realistic if you have a large amount of data to transfer and a limited network link between sites. You could make sure that SnapMirror only runs after deduplication completes, for example, to conserve network traffic.

Figure 2) Restoring a VM.

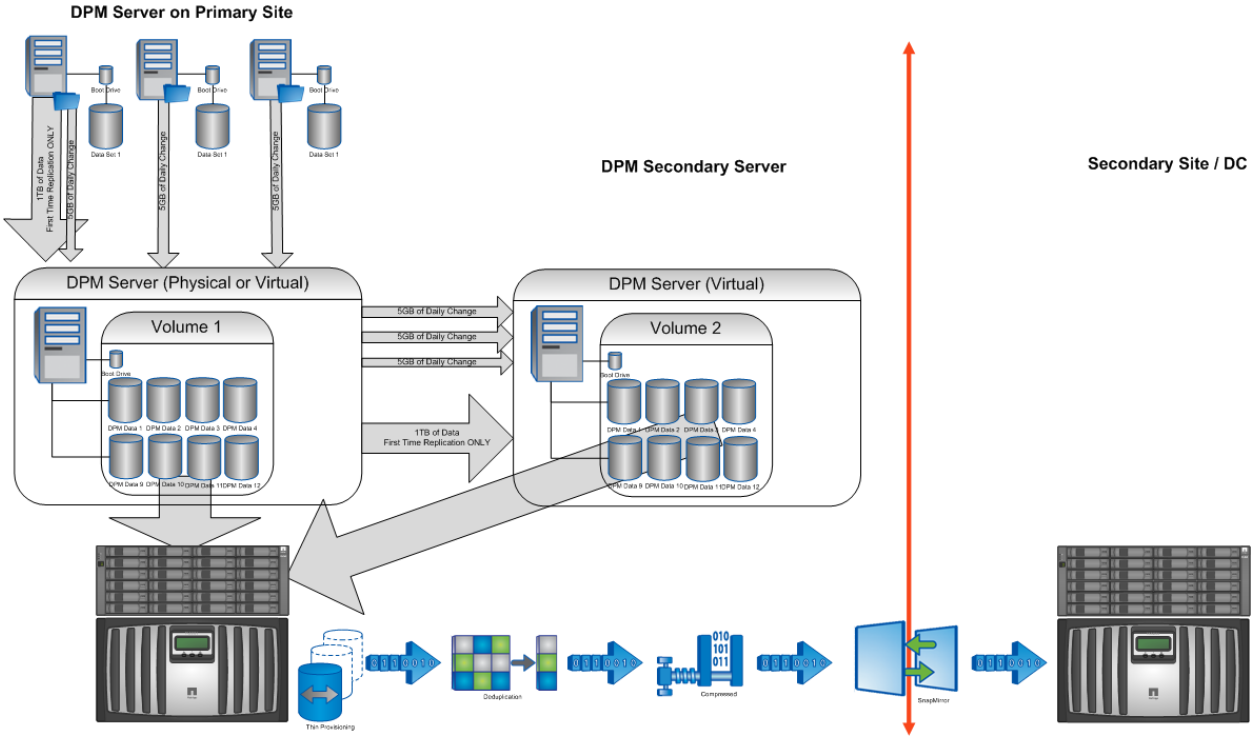


SITE-TO-SITE FAILOVER WITH CHAINED DPM SERVERS

Another alternative to failing over your primary DPM server is to create a chained DPM server that backs up the first DPM server. In this mode, DPM server 1 backs up your hosts, while DPM server 2 backs up DPM server 1. In this case, you can mirror DPM server 2 with SnapMirror and relocate DPM server 2 to the remote site. In this method, DPM server 2 will transfer all of its data from site to site without the aid of SnapMirror, deduplication, or NetApp compression. This has a few distinct advantages over a single DPM server.

- If the OS of the first DPM server is corrupted or the application fails, the second DPM server maintains its copy and can restore the first DPM server.
- DPM server 2 needs to restore a single application to the DR site, but you don't want to stop the backup process on the primary site.
- If a large amount of data needs to be transferred from DPM server 1 to DPM server 2, you can relocate DPM server 2 back to the primary data center to allow fast transfer.
- Once DPM server 2 is caught up to DPM server 1, deduplication can be started; once completed, SnapMirror can copy that new data to the remote site. DPM server 2 can be relocated to the DR site.

Figure 3) Site-to-site failover.



1.4 IN-DEPTH PREDICTION OF SPACE NEEDS

The process of prediction for a single DPM server is no different than prediction for multiple DPM servers in a server farm. The process for DPM server farms must be approached as a superset of the solution of the single DPM servers.

DETERMINING SERVER SPACE TRENDING

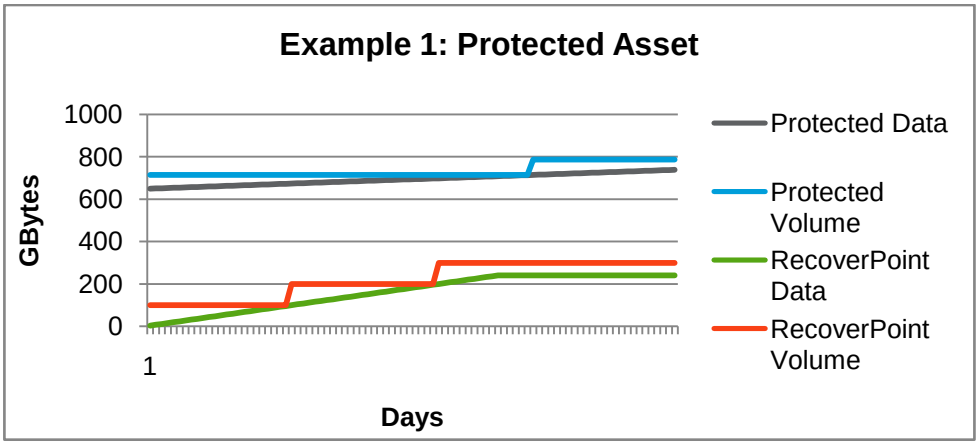
Gaining an historical perspective of the servers you are adding to the DPM server is a good process so that you don't overload individual DPM servers. If there are servers on a SAN-based controller, this trending information should be available via a SAN management application, or, in the case of NetApp, this information is available via AutoSupport™. The following information is required:

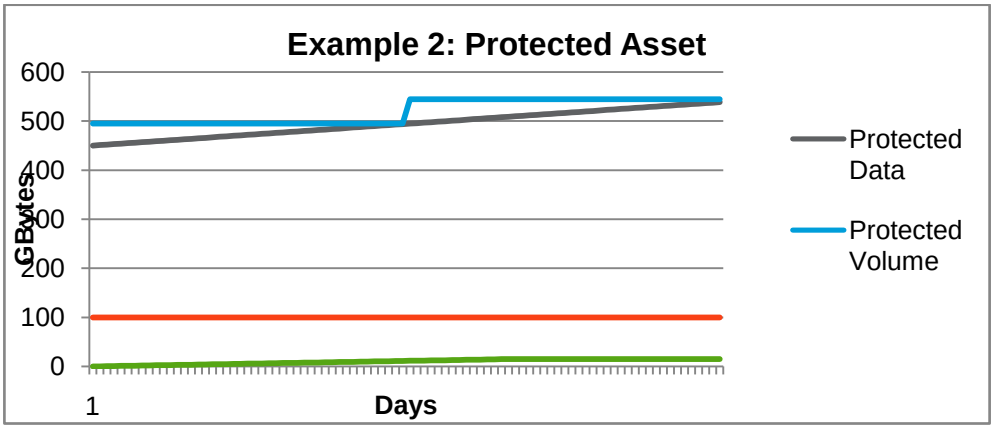
- Size of the partitions on the host to be backed up
- Current space used on those drives
- Estimate of the write traffic to that partition
- Growth rate of the current space used

As an example, suppose a server has 1TB of space to protect. In this 1TB partition there exists 650GB of data. Through trending you find that you write 5GB of data to that drive each day, but that the space used only grows by 1GB each day. This results in an overwrite of 4GB of data, and an addition of 1GB of new data daily.

Another example would be a server with a 1TB partition with 450GB of space used. Through your trending analysis you see that you write 3GB per day, and that the space used also grows by 2.75GB daily. When these two servers are protected in DPM 2010, the DPM server creates a 715GB (650GB + 10%) protection volume, and another 100GB recover point volume (10% of 1TB). You can predict that the 715GB protection group will start out at 650GB full and on average will grow towards 715GB by 1GB per day. By day 65, you can expect that the 715GB volume will create another 72GB expansion (10% of

715GB) expansion. You can expect that the 100GB recover point partition will start empty but grow by 4GB per day. If the retention period is 60 days, you can expect the recover point partitions to grow to 240GB. The default 100GB partition will grow to roughly 200GB on day 25, and, on day 50, will again expand to 300GB. Once the stable state space of 240GB is reached, it is likely that old data falling out of retention on the recover point volume will match the new data being added.





In the above examples, Protected Asset 1 will eventually start at close to a 13% thin provisioning benefit, but will eventually stabilize at approximately a 6% thin provisioning benefit. Protected Asset 2 will start at approximately an 18% benefit due to thin provisioning, but will eventually stabilize at around a 13% benefit of thin provisioning.

HOW NTFS WRITES FILES TO WINDISKS

NTFS has a peculiar method to write files to a file system that complicates both thin provisioning and deduplication efforts and limits their effectiveness. In this case let's take a 10GB NTFS volume that has been quick formatted and add 4 x 1GB files to that file system. The file system will report that it has 6GB free, and the array controller will report that the LUN that represents the WinDisk has 6GB unallocated. If a single 1GB LUN is deleted, the file system will report back that 7GB are now free, but the blocks that occupy that 1GB file are not deallocated. The array will still claim that 6GB are free instead of the 7GB that the file system thinks are free.

To complicate factors, the next file added to the file system will not overwrite the recently deleted files, but will instead be placed on a new unoccupied space. This allows the file system to preserve the ability to later attempt an undelete process on files that were wrongly deleted.

This behavior means that a file system that maintains a free space will see diminished returns for features such as thin provisioning and Zero Space Reclaim.

EFFECTS OF LACK OF DEALLOCATING ON THIN PROVISIONING AND REMEDY

Since DPM continually adds and deletes files from the recover point volumes over time, and makes no effort to zero or deallocate previously used space, this volume is a prime candidate to suffer from this effect. In the above examples, Dataset 1 would see at most a 20% thin provisioning benefit on the Recover Point dataset, while Dataset 2 could see an 85% thin provisioning benefit on the Recover Point database. In either case, use a tool from the Microsoft SysInternals Toolkit called SDelete to write zeros to all of the unallocated space in each recover point volume. In each case listed above, you would need to write approximately 60GB and 85GB of data, respectively. Once the SDelete Tool had completed its run, execute the deduplication process on the appropriate volumes to reclaim that space back to the appropriate volumes.

Note that since DPM uses dynamic-type disks instead of basic-type disks, the SnapDrive® space reclaimed application cannot reclaim space from the data drives in the data pool.

PROCESS TO RUN SDELETE ON RECOVER POINT VOLUMES

The SDelete application is a free download from the Microsoft TechNet Web site: (http://technet.microsoft.com/en-us/sysinternals/bb897443.aspx. It is a tool for securely deleting all of the empty space or deleted files on a file system.

You then need to create a map from the array volumes to the associated LUNs on the server. If the DPM server is within virtual machines, this will require first discovering the mapping from the array to the Hyper-V host, then from the host to the guest.

The array contains 4 x LUNs within two volumes:

```
/vol/Volume1/DPMData1    - Maps to Windisk1 on DPM
/vol/Volume1/DPMData2
/vol/Volume2/DPMData4
/vol/Volume2/DPMData5
```

In this example, I would map that the DPMData1 LUN eventually maps to the DPM1 server as WinDisk1, that DPMData2 maps to the DPM1 server as WinDisk2, and so on. In a real-world environment, you may have 40 LUNs per DPM server, and the mapping will likely be more random. You can gather all of this data from the NetApp MPIO DSM server manager MMC.

There is a script to create an Explorer contextual menu item to run SDelete against a drive letter.

The full details of this method can be found here http://forum.sysinternals.com/topic6065.html.

The full script can be found in the Appendix: Scripts.

The process is as follows.

- Identify all of the partitions that exist on a single DPM WinDisk.
- Trace that WinDisk back to the NetApp volume.
- Count the free space from the NTFS that would be zeroed. For example, if you have 2TB of drive space and all of the free space of all of those partitions equals 400GB, then record 400GB.
- Make sure that the volume that the LUN exists in has enough free space to handle an influx of 400GB of new data, that is, that volume should have these expected daily change rates plus 400GB of free space.
- Walk through the following procedure on each LUN in the WinDisk:
  - Add a drive letter to a mount point as an alternate method to connect.
  - Start the zero free space process with a single pass.
  - Remove the drive letter to the mount point.
- Once the complete WinDisk has been zeroed you can choose to continue to other WinDisks in the same volume.

- Once either the volume is approaching a full space or the SDelete process on the volume has been completed, initiate the deduplication process on that volume.

1.5 SDELETE SCRIPT HOW-TO

The following script completes all the above steps with a few additions. The script has two additional safety features and makes a few assumptions.

SDELETE SCRIPT 80% THRESHOLD

The two safety features are that the script has a defined threshold and that it will not fill a volume to exceed that threshold. As an example, if the threshold is 80%, then the SDelete process will run only when the addition of a new volume will not exceed the 80% barrier. If the volume is 78% full and by SDeleting the next volume the used space would change to 82%, the tool will skip the mount point and move to the next mount point. This threshold within the script can be quickly changed to match your acceptable threshold. A site that experiences only moderate load/additions to the DPM server during a nightly backup and has a massive disk store might find that this could easily be changed to 90%.

WEEKLY TIMEOUT OF THE SDELETE TIMER

The SDelete script in the Appendix also assumes that you want to run the SDelete process against a specific drive only once per week. As the SDelete process runs, it leaves a notice on the drive that the program has been run. The timestamp of the notice is compared to the current time to see if a week has elapsed. This allows the script to be run partially and to be aborted; if restarted it will pick up roughly where it left off. This one-week timeout can also be easily adjusted in the script to meet your current needs. If you want to prevent SDelete from running on a particular replica or recover point volume, all you need to do is update the timestamp of the SDelete complete file within. This allows you to alter behavior to meet your needs.

ASSUMPTIONS ABOUT S: DRIVE

The script also uses a Drive Letter S:\ to access all of the individual mount points used by DPM. If the script is aborted in the middle of a run, it is likely that an S:\ drive will still be connected to a mount point. This S:\ drive can be manually removed via the Server Manager Disk Management tool. Alternately, if you are not concerned with this extra drive letter, the next time the script runs it will first unmap the S Drive from its current location so as to not prevent the script from operating.

CAVEAT OF USING SDELETE TO RECLAIM THIN SPACE

The caveat of using zero blocks to reclaim data is that over time it will appear that our Thin Provision Space will appear to be disappearing, while your deduplication rates will climb by exactly that amount.

In this case, the Thin Provision Space will only be indicative of space that the DPM server has not partitioned out instead of space that DPM has simply not written to. The new deduplication rate will represent both the savings from actual deduplication efforts as well as space that has been recovered via forced zero overwrites.

ELIMINATING OVERPROVISIONING UNTIL ALL VOLUMES HAVE BEEN EQUALLY ALLOCATED

One of the challenges of DPM is that it gives you limited control over where protected data is actually stored after you add a selection of disk to DPM for it to add to the available pool. In a case in which you are using multiple volumes to store a single DPM server, you will likely want to make sure that these volumes are not oversubscribed until they are all equally filled. To do this you can create partitions on each of these volumes after they have been added to DPM, but before DPM allocates the full space of each drive. As an example, assume that a 50% savings rate will be found from the combination of deduplication and thin provisioning. Until the 50% full mark is exceeded, this particular DPM server can run in a nonoversubscribed mode.

Consider the following scenario:

- Assume that you want to allocate 80TB to the DPM server.
- 50% savings is required from the DPM server.
- You need to deploy 40TB of usable storage, and the deduplication limit is 3TB on this controller.
- Assume also that a FAS2040 controller is used.
- Three 14 x 3TB volumes are created and split among 3 x 16TB aggregates.
- Each 3TB volume will contain 3 x 2TB LUNs. (Each of these LUNs is oversubscribed 2 to 1.)
- In this case, you will want to fill each LUN only halfway before moving to the next LUN.

To facilitate this, create dummy partitions of 1TB on each 2TB LUN. DPM will avoid user-created partitions on disks in the disk pool. By creating these dummy partitions and not formatting them, DPM will be able to spread data more equally across all drives until that nonoversubscribed space is completely used. Once the last drive in the set is filled up, you can revisit these partitions.

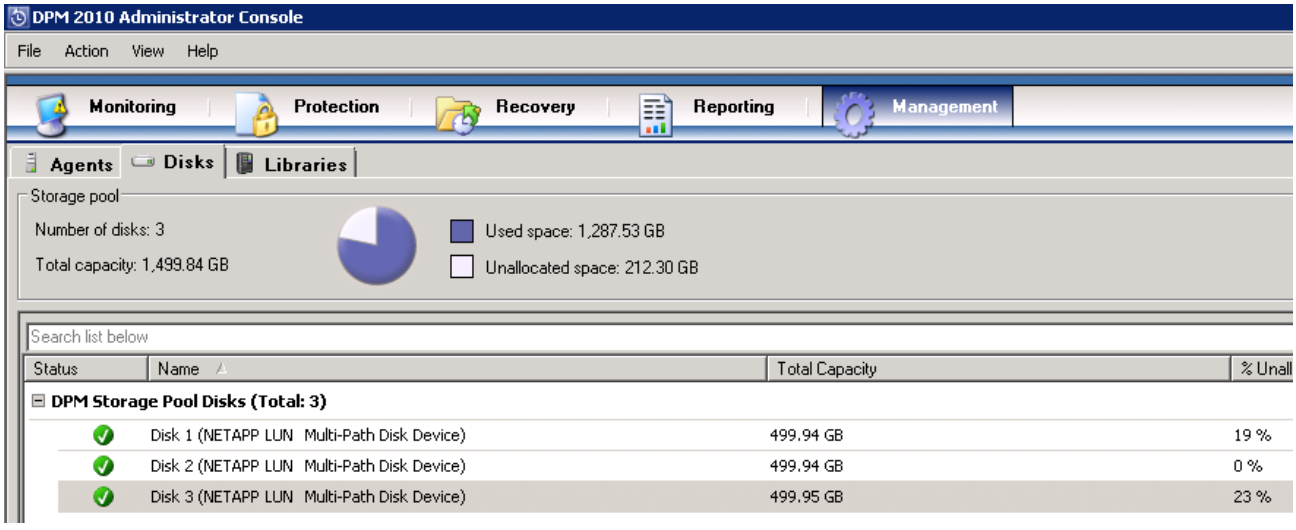
To prevent the oversubscription rate of space on one volume from immediately going to 50% while other partitions stay at 0%, selectively remove these partitions, freeing space for DPM to continue operating. As an example, instead of removing the dummy partitions from all three LUNs in the first volume, only remove the dummy partition from one LUN in the first volume and the dummy partition from one LUN in each subsequent volume. Once that space has been consumed, you can continue to equally remove dummy partitions from LUNs from each volume.

Space-saving technologies such as deduplication do not need to occur aggressively (or at all) when these dummy partitions exist. Once they are removed and DPM is allowed to start using this oversubscribed space, more care must be taken so that volumes are not overloaded.

WHEN AND HOW TO DELETE DEAD SPACE DUMMY PARTITIONS

You can monitor the space that is used from within DPM via the DPM drive. From the DPM 2010 administrator console, you can determine how much space DPM has available without tapping into that dead space that has been reserved. From Figure 4, you can see that 213GB of space is free according to DPM.

Figure 4) DMP administrator console.



When deleting dead space, you should delete dead space equally from each volume within the DPM server. When the DPM server has filled out the space that is not oversubscribed you should begin to assign more space slowly to the DPM server one unit at a time. In this example, assume three volumes,

each of which is 16TB in size and each of which has 13 x 2TB LUNs defined. In other words, 78TB of space is presented to the DPM server, but only 48TB of actual space remains; that is, you are oversubscribed by 40%. This means that you will have to create an 800GB dead space partition on each of the 39 WinDisks. After reviewing the deduplication rates of the dataset, you determine that there is a benefit of close to 33% from deduplication and thin provisioning. Instead of immediately freeing up the entire reserved space, you can organically grow towards the exact rate by selectively removing dead space partitions. If you randomly remove half of those partitions, one volume might see its safety buffer removed while another volume would never approach its actual capacity. Care should be taken to remove equal numbers of these partitions from each of the underlying volumes.

2 OPTIMAL STORAGE LAYOUT

As seen from the previous section, the optimal storage solution will change depending on a number of factors. These factors include the controller models used to host the DPM datasets. For example, the maximum dedupe volume size can vary for different controller models and the versions of Data ONTAP running on them. For information about these limits, refer to TR-3505: NetApp Deduplication for FAS and V-Series. As an example, the FAS2050 running Data ONTAP 7.2.5 can only dedupe 1TB per volume, while it can dedupe 2TB with Data ONTAP 7.3.4; on Data ONTAP 8.0 that limit is raised to 16TB.

2.1 DATA ONTAP CHOICE AND CONTROLLER CHOICE

Obviously, with 16TB volumes, the LUN layout and points of management decrease, which reduces complexity, but the time to complete and the impact of running a dedupe scan increase.

Another factor to consider is the 16TB aggregate size limit of Data ONTAP 7.3.x versus the 100TB aggregate size limit of Data ONTAP 8.x.

Best Practice
NetApp therefore highly recommends running Data ONTAP 8.x.

Another factor is the limit of the number of hosts and servers that a single DPM server can protect, making it highly likely that you will need to deploy a farm of DPM servers to protect a whole site. This farm of DPM servers has very limited needs in terms of CPU and memory and would benefit from encapsulation within a virtual environment. Care must be taken when designing the network layout for a farm, because bottlenecks can easily form when using Gigabit Ethernet, while a 10Gb/s Ethernet connection will likely be overkill for a single DPM server.

Best Practices
• Run DPM within virtual machines. • Colocate multiple DPM virtual machines as long as networking is not overloaded.

If deploying a farm of DPM servers, a number of factors come into play concerning scale. Each of the NetApp controllers has very distinct capabilities when it comes to the number of drives it supports, the size of deduplication volumes, as well as the number of concurrent deduplication sessions and SnapMirror sessions that can run.

Best Practices
• Scale your controller by the concurrent deduplication sessions you expect to run. To determine how many deduplication sessions can be run simultaneously on each of the different platforms and on different versions of Data ONTAP, see TR-3505: NetApp Deduplication for FAS and V-Series. • Scale your controller by the concurrent SnapMirror sessions you expect to be operational. Please refer to the TR on SnapMirror to properly size this. TR 3446: SnapMirror Async Overview and Best Practices Guide • Scale your controller by the total number of drives required to support your farm. Once your deduplication and SnapMirror requirements are satisfied, you may still need to enlarge your controllers to support a significantly large drive count.

Any savings gained by deploying a FAS2040 over a FAS3170 can be quickly lost when you have to purchase multiple FAS2040 controllers to service a large drive count or the extra SnapMirror licenses you may need to protect those FAS2040s, not to mention the added points of management.

2.2 AGGREGATE DESIGN

The maximum aggregate size for your platform should be selected. There is a distinct advantage for using Data ONTAP 8.0 or later with DPM because aggregate limits are greatly increased. The aggregates chosen for DPM should reflect the need for capacity and ease of administration over raw performance or small granular control, since that control is simply lost within DPM. Another consideration is the need to support compression, which requires 64-bit aggregates.

Best Practices
• Choose 64-bit aggregates. • Choose the largest aggregates that optimize for capacity. • Choose the largest SATA-type drives.

2.3 VOLUME DESIGN

DPM can use up to 80TB of storage per DPM instance. While this space may see the benefit of 66% savings due to thin provisioning/deduplication/compression, this amount still exceeds the volume limits imposed by the Data ONTAP deduplication engine. Consider the maximum size per volume to match the maximum deduplication pool allowed.

Best Practice
Do **not** allow volumes to grow.

It should also be noted that a DPM server has a custom database and cannot be rolled back should a problem occur. Typical Snapshot copies will not afford added protection to the DPM database because DPM has no method to import those older databases and the dataset will simply become corrupt. As such, the only method to back up a DPM server is to back up the OS volume along with the database.

Best Practices
• Do **not** set a fractional reserve. Set the fractional reserve to 0%. • If not using virtual machines, set the Snapshot space to 0%.

DPM will benefit from thin provisioning, but this can be accomplished at the LUN level without affecting the reliability of other DPM servers. Use thin provisioning at the LUN level only.

Best Practice
Thickly provision the volumes.

Additional consideration must be made for the deduplication process. This process may incur as much as 4% of the aggregate. This means that if you use a 100TB aggregate and you wish to create 10TB volumes that are thickly provisioned, you need to make sure that 4TB of space is left in the aggregate. In this case, it might be beneficial to consider that only 96TB of space is usable, and to create 8 x 12TB volumes that are thickly provisioned.

Best Practice
Leave 4% free space in each aggregate.

The volume name should be self-descriptive of the contents of that volume.

Best Practice
Volume names should contain the NetBIOS name of the server; for example, DPM3_Vol4.

2.4 LUN DESIGN

Each of the volumes will contain a large number of LUNs, but those LUNs should be limited to a single DPM server. A single DPM server will consume more space than any single volume can supply, and little benefit can be gained by sharing a volume amongst multiple DPM servers.

Best Practice
All LUNs in a volume should be owned by a single DPM server.

A common approach is to assign either two or three 16TB volumes to a single DPM server. The DPM server will need a 100GB volume for a boot OS if that DPM server is virtual (highly recommended), and between 8 and 20 x 2TB LUNs depending on your expected rate of savings.

Care should also be taken to label the LUNs appropriately so that they can be given to the correct DPM server should a failover need to occur. A good method to keep track of LUN ownership is to place the name of the VM the LUNs belong to in the comment field for each LUN, or incorporate them into the name of the volume that is being mirrored from site to site.

Best Practices
• The DPM NetBIOS name should be Subset of LUN Name; DPM3_Data13. • All DPM data LUNs should be defined as thin and relatively larger (2 to 4TB each). • The DPM Boot VHD should be thickly provisioned, but relatively small (50 to 100GB).

FAILOVER SITE CONSIDERATIONS

If the failover site is running dissimilar hardware, make sure that if operations need to resume from the failover site that the failover array is capable of continuing the deduplication process. If the failover site is configured with a FAS2040 while the main site is configured with a FAS6080, the secondary site has a volume limit much lower to run deduplication.

Best Practice
Adhere to the volume limits imposed on the failover site when designing the primary site.

Another consideration for the failover site is the number of volumes that are being constantly mirrored. For example, if a DPM server is using 80TB of space split across three volumes, those three volumes should be mirrored and be on the same schedule.

SNAPDRIVE LIMITATIONS ON LUN CREATION

When giving a drive (LUN) to a DPM server, that drive must be a raw drive without any existing partitions defined. In fact, the drive you give to the DPM server must have no partition defined; otherwise, DPM does not think it is available for use. The SnapDrive tool by default will create a LUN, mount that LUN, and create a partition on that LUN either as a drive letter, a mount point, or a CSV (if in a cluster). There is no option to simply mount a LUN without creating one of these optional settings. If SnapDrive is used, you will need to go back into the Disk Management tool and destroy any partitions that are made using SnapDrive.

LUNS TO BE USED FOR VIRTUAL MACHINES

To function as a virtual machine, the DPM server must boot from a VHD. Once the DPM server has been built and DPM has been installed, you can install the DPM data drives. The DPM data drives can either be directly mounted inside the VM via iSCSI, or they can be Fibre Channel or iSCSI drives mounted to the VM as pass-through disks via iSCSI or FC.

Alternately, you could create VHDs within each of the DPM data disks and assign them each to the DPM server, but this would require an extended amount of time since the VHDs must be fixed VHDs to allow deduplication. Since the DPM server will very likely be oversubscribed, take care to only format a subset of VHDs at a time and to allow deduplication to run between format sessions. This process is cumbersome and easily avoided via pass-through disks. Even though dynamic VHDs solve the quick format problem presented by fixed VHDs, due to missalignment problems these disks lose all ability to dedupe.

Best Practices
• Do not use fixed VHDs for data drives. • Do not use dynamic/difference VHDs for data drives. • The boot drive should be a 100GB fixed VHD.

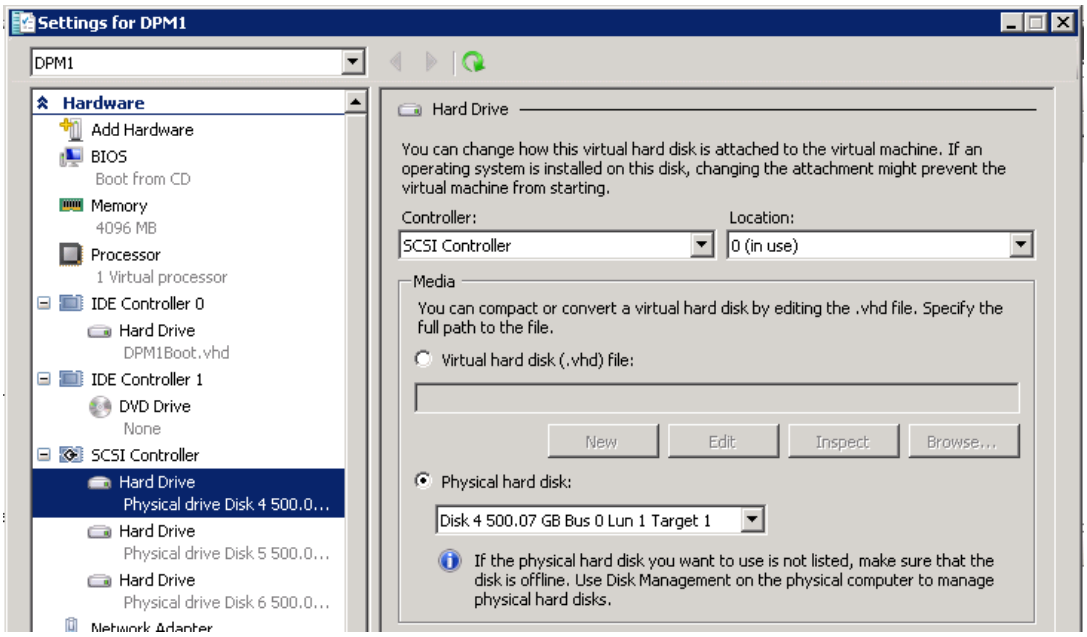
You may wish to install SnapDrive to facilitate a regularly scheduled backup process using SnapManager for Hyper-V (SMHV). If using SMHV, the backup process can be automated and the deduplication SnapMirror resync can be kicked off as part of the process.

Adding a physical pass-through disk to Hyper-V

To install a disk as a pass-through disk directly to the DPM Hyper-V node, choose settings on the VM and, under SCSI controller, select a physical hard drive.

Note: The physical hard drive must be offline from the host OS before Hyper-V will let you choose it as a possible drive for the guest OS.

Figure 5) DPMI settings.



Do not be alarmed when the disk disappears from the Disk Management Pane of the Host OS Disk Management; this is expected behavior.

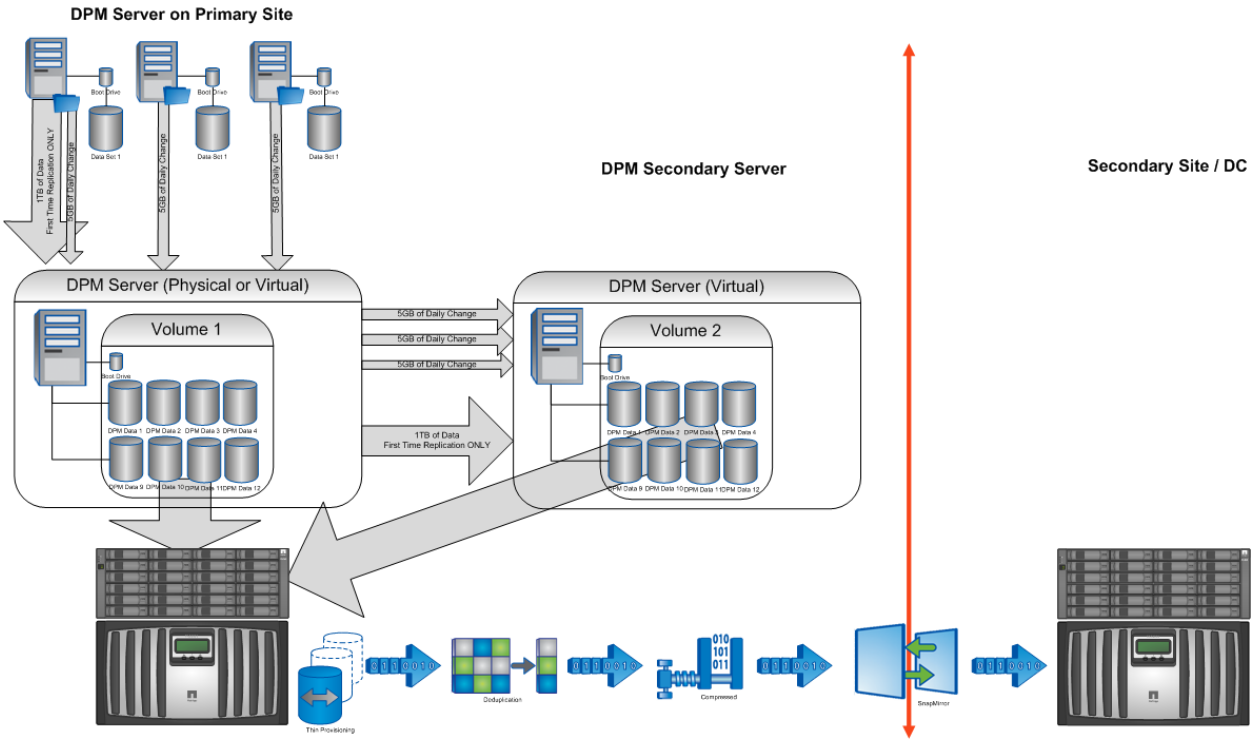
3 DPM SITE-TO-SITE FAILOVER USING SNAPMIRROR

There are a number of ways to allow a DPM instance to be failed from site to site. The first and most obvious way that requires no storage optimization is to allow DPM to mirror its data from site to site. This method is the simplest way to allow a dataset to be recovered from a remote site. This method is covered thoroughly on the Microsoft site and functions equally well in all DAS and SAN environments.

DPM INSTANCE FAILOVER FROM SITE TO SITE

The next step up from the basic site-to-site failover is to allow a DPM server to fail from site to site. In this case the DPM server on the primary site would live within a virtual machine. The virtual machine would be protected with either the SnapManager for Hyper-V without any interruption in operation or, if a slight interruption of service is allowed, via the script listed in the Appendix: Scripts. The need to run either an SMHV or a scripted VM Save will likely be determined by both your recovery point objective (RPO) and recovery time objective (RTO). Figure 6 illustrates the relationship and logical view of the failover from site to site.

Figure 6) Site-to-site failover.



CASE 1—SHORT RPO

In this case, suppose you require an RPO of three hours and that your DPM server is active during this time. You may opt to solve this by using SMHV to create Snapshot copies throughout the day. In this case you may find that the deduplication window prevents you from waiting until the deduplication process is complete before mirroring the data to the remote site. While this still allows the benefit of SnapMirror compression for newly added data, it prevents the added advantage of deduplication. In this case, you can choose to have SnapMirror keep a constant mirror and set a recurring schedule with SnapDrive.

CASE 2—SHORT RTO, MODERATE RPO

In this case, suppose you need to be able to fail over to an alternate site within a short time frame, but the data on the remote site may be behind the primary site by many hours (like 10 or 12) or days (1 or 2). This longer window allows the DPM server to be paused (VM Saved) for short periods, and also allows the deduplication process to be performed on new data before mirroring it to the remote site; in this case, using either SMHV or the scripts defined in the Appendix suffices. The deduplication process could be kicked off immediately preceding large backup sets being added and the SnapMirror process could be set to only occur when triggered. The trigger would occur once the dataset has completed its deduplication and might be triggered a couple of times a day. In this case the more relaxed RTO/RPO allows maximized use of the network bandwidth between sites since the data is both deduplicated as well as compressed in flight.

PROCESS—DPM SETUP

The following process shows how to set up the DPM server for everyday operation. It should be noted that site-to-site failover requires that a VM be previously exported using the config only export tool listed in the Appendix. It is important to note that a config only export option existed in Windows 2008 Hyper-V Manager, but that a GUI option **does not** exist in Windows 2008 R2 Hyper-V Manager. The functionality for config only exports still exists, but the GUI button to start the process is missing. This step must be done via the command line.

- Install the virtual machine that is the DPM server. This machine should consist of at least two CPUs (cores) and 6GB of RAM, 100GB boot drive (Fixed VHD), and up to 40 x 2TB data volumes. Once the DPM server is installed, install DPM and assign all of the data drives to DPM for use as DPM data drives.
- Create the dead space partitions on each WinDisk once DPM has started to protect the oversubscribed space until the data has been equally distributed among all of the drives.
- Create the SnapMirror relationship between sites to mirror volumes that belong to this DPM server. Depending on your expected RTO/RPO and your network bandwidth, set your SnapMirror update schedule appropriately. If you only want to mirror the data from site to site once deduplication has occurred, set this to manual and allow your scripts to initiate these updates from site to site.
- Modify the scripts to include volumes that this server uses and the array credentials. Copy scripts to both the primary and secondary sites.
- Install and configure the server on the secondary site that can run the failed-over DPM instance. It should be configured to have either iSCSI or Fibre Channel connections to the secondary site array controllers.
- If SMHV is being used to mirror the virtual machines between sites, then install and configure SMHV on both the primary site server and the secondary site server.
- As a last stage of the virtual machine configuration, you need to create a virtual machine config only export. This config only export will allow you to import the VM on the remote site. For the config only export, see Appendix: Scripts. The data files for this config only export should exist in the same volume as the VM Boot VHD.
- Use either the SMHV or the scripts listed in the Appendix to create a Snapshot copy of the DPM server.
- Make sure that the SnapMirror process has completed mirroring this data between sites.

DPM FAILOVER PROCESS—CONTROLLED FAILOVER

This is the first of two types of failovers. In this type of failover the process is planned and steps can be taken so that no data is lost between sites.

- Bring the DPM VM down via a normal shutdown operation or saved state operation.
- Institute a final config only export to make sure that any recent hardware changes have been accounted for in the configuration.
- Use either SMHV or the scripts to create final Snapshot copies of all of the volumes that the DPM server is using.
- Kick off a final SnapMirror resync between sites so that all final changes are transferred between sites.
- Institute a secondary site promotion of the SnapMirror destination (via a SnapMirror Break).
- Reverse the direction of the SnapMirror process using the `resync` command and reverse the source and destination.
- Expose the LUNs on the secondary site to the secondary server.
- Import the VM from the config only export directory. Determine that all of the volumes are mapped to this VM.
- Start the VM at the secondary site.

This process can be repeated in reverse to move from the secondary site back to the primary site when the time is correct.

DPM FAILOVER PROCESS—OUTAGE PROCESS

This is the second of two types of failovers. In this type of failover the primary site is down and the decision has been made to move operations to the secondary site.

- Bring the DPM VM down via a normal shutdown operation or saved state operation.
- Institute a secondary site promotion of the SnapMirror destination (via a SnapMirror Break).
- Expose the LUNs on the secondary site to the secondary server.
- Import the VM from the config only export directory. Determine that all of the volumes are mapped to this VM.
- Choose if you want to start from the most recent state of the DPM server or from the most recent application-consistent Snapshot copy. The benefit of starting from the Snapshot copy is that the image is consistent but may have an RPO that is farther into the past.
- Start the VM in the secondary site.
- When the primary site returns, reverse the direction of the SnapMirror action by swapping the source and the destination.

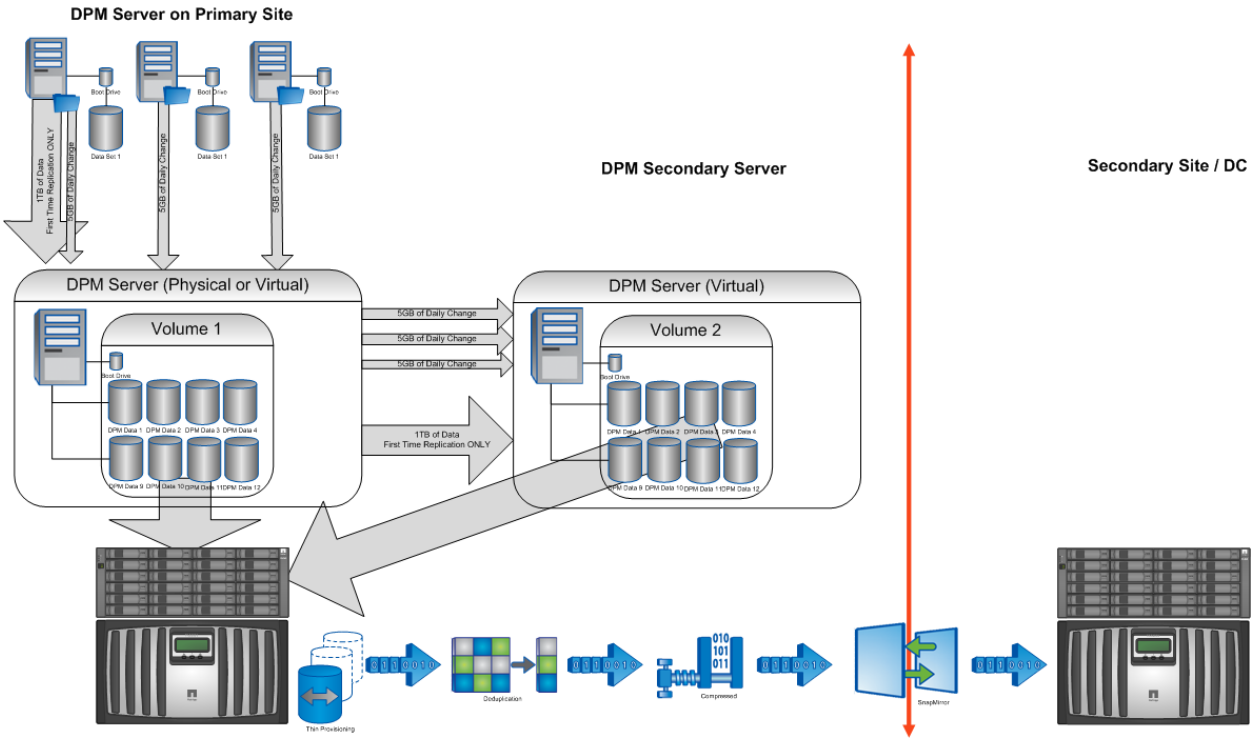
This process is only to be used to recover from an actual outage and should not be used to fail back to the primary site, since some data may be lost during the failover process. The proper failback scenario is to use the controlled failback procedure.

3.1 DPM PAIR UTILIZING FAILOVER

Another method to enable site-to-site failover both utilizes the DPM failover mechanism built into DPM natively and takes advantage of the optimized storage. In this method, the first DPM server is deployed on the primary site as normal but without setting up site-to-site mirroring. We then configure a second DPM server for the primary site to act as a backup for the primary DPM server. In this method, the latency between the DPM servers is kept to an absolute minimum, allowing DPM to operate as efficiently as possible. The second DPM server is mirrored from site to site using the site-to-site DPM protection mechanism. The distinct advantage of this type of configuration is that the second DPM server can be pivoted from site to site at will without interrupting the backup cycle on the primary DPM server.

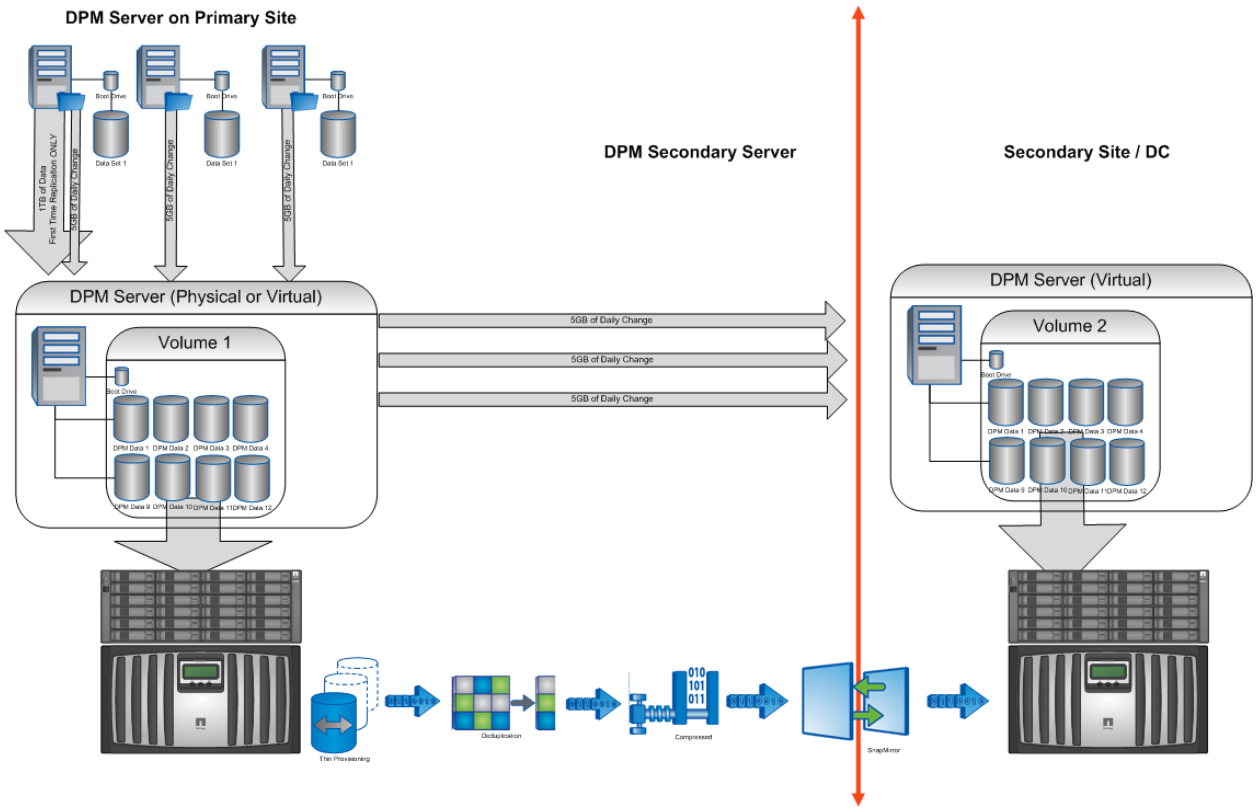
Figure 7 shows the logical location of the DPM servers when both exist on the main site.

Figure 7) Site-to-site failover.



This allows a restore to be done from the second site of a server while a full backup is occurring on the primary site, without causing any high latency on either DPM server. The second DPM server in effect can be relocated from site to site at will, depending on where it's needed most. If a large amount of new data needs to be added to the DPM server pair, then the secondary DPM server can relocate itself to the main site and gain all the benefits of being local. Once the deduplication and other space-saving mechanisms have occurred, and once SnapMirror has resynced the sites, the DPM server can be relocated back to the secondary site. Figure 8 illustrates the same environment with the secondary DPM server relocated to the secondary site.

Figure 8) DPM pair failover.



Relocation of the second DPM server can be done in less than two minutes and is scriptable and automatable using the same process as a controlled site-to-site failover.

Another advantage of this design is that the primary DPM server may either be a virtual machine or a physical machine; only the secondary DPM server needs to be a virtual machine.

4 APPENDIX: SCRIPTS

4.1 SCRIPT TO ALLOW SDELETE TO BE RUN FROM THE EXPLORER CONTEXT

Copy this script to a file called `sdelete.inf` and double-click to launch the installation process. Once done this script will allow you to right click on a drive letter and select the SDelete operation directly.

```
; context_sdelete.inf
; Adds Zap Free Space to the right click context menu in Windows XP
[version]

signature="$CHICAGO$"

[DefaultInstall]
AddReg=AddMe

[AddMe]
HKCR,"Drive\Shell\Zap Free Space\command",,,"sdelete -p 1 -z %1"
```

4.2 SCRIPT TO AUTOMATE SDELETE AGAINST ALL LUNS IN A DPM SERVER

The following script:

- Unmaps all existing mount points to the S Drive
- Walks through each mount point, mounting them and processing them on all drives
- Checks for a last-completed SDelete so that the SDelete process doesn't run against a drive that does not have a sufficient amount of reclaimable space
- Skips SDelete on volumes that would exceed a safe threshold on each volume
- Initiates deduplication on a volume once a complete LUN has been processed to recover space

```
# Script for DPM
#
# Script will map the DPM Mountpoints to the NetApp Filer
$filer = "10.58.99.254"

# Script will only let a volume fill to this percentage before it will stop running SDelete on Member LUNs
$threshold=80                                   # 80% threshold set
$DaysBetween=7                                  # will prevent SDelete from being run on a volume except every 2 days.
$user="administrator"
$password = ConvertTo-SecureString "Netapp1" -AsPlainText -Force
$cred = New-Object -TypeName System.Management.Automation.PSCredential -ArgumentList $user,$password
$dump = connect-nacontroller $filer -cred $cred #suppress the connect screen for Powershell Kit

$drives=gwmi Win32_diskdrive
cls
Write-host "SDELETE Script. "
Write-host "----------------"

function Exists-Drive{ #Returns True if S: drive is currently Mapped
        param($driveletter)
        (new-object system.io.driveinfo($driveletter)).DriveType -ne 'NoRootDirectory'
}

if (Exists-Drive 'S:')
{   $removedS=$false
    write-host ' S exists. Going down process of removing access to Drive S:'
    Foreach($disk in $Drives)
    {   # Walk through the Drives one at a time
        $wind=$disk.name -replace '\\\\.\\PHYSICALDRIVE',''
#-----------------------------
$command = @"
select disk "$wind"
detail disk
"@
#---------------------------------
        $temp1 = $command | diskpart
        $count=0
        while ($count -lt $temp1.length)
        {   # Walk through the Output of the Details Disk for each drive line at a time
            $t=$temp1[$count]
            $tvol=$t[1..10]
            if ($t[15] -eq 'S')
            {   # This goes back into Diskpart and removes the drive
                $volnum=$t[9]+$t[10]+$t[11]
$driveit = @"
select volume"$volnum"
```

```
remove letter=s
exit
"@
                $dump = $driveit | diskpart
                Write-host "     Removed Drive Letter S from Volume "$volnum
                $removedS=$true
                break
            }
            $count=$count+1
            if ($removeds) { break }
        }
        if ($removeds) { break }
     }
}

foreach ($disk in $drives)
{   # Major Loop, Walks Through each WinDisk
    $wind=$disk.name -replace '\\\\.\\PHYSICALDRIVE',''
    Write-host "WinDisk"$wind "=" $disk.SerialNumber"Serial Number"
    $tpath=''
    foreach($lun in get-nalun)
    {    # Checking for the Serial Numbers against the Array.
         if ((get-nalunserialnumber $lun.path) -eq $disk.SerialNumber)
         {    $tpath=$lun.path
              write-host '    LUN Found = '$filer':'$tpath
         }
    }
#----------------------------
$command = @"
select disk "$wind"
detail disk
"@
#--------------------------------
    $temp1 = $command | diskpart
    $count=0
    $WriteIt=0
    $tvv=''
    if ($tpath -ne '')
    {   # Lun is NetApp, trim off the LUN portion of the Path to get the Volume. Also Tell Free Space on VOL for the Windisk
        $tvv=$tpath
        while ($tvv[$tvv.count-1] -ne "/")
        {   # Cut letter at a time until I hit tha last / mark.
            $tvv=$tvv.trim($tvv[$tvv.count-1])
        }
        $tvv="/"+$tvv.trim($tvv[$tvv.count-1])
        # Tell me the Free Space that Exists on the LUN in that Windisk
        $tvolobj=get-navol $tvv
        $tvolsize=$tvolobj.sizeavailable
        $x2=[system.math]::round($tvolsize/1024/1024/1024*10)/10
        $tvoltotal=$tvolobj.sizetotal
        $volperc=[system.math]::round(1000-$tvolsize/$tvoltotal*1000)/10
        $x1=[system.math]::round($tvoltotal/1024/1024/1024*10)/10
        Write-host "     NetApp Volume Containing this WinDisk = "$x1"GB, FreeSpace in This Volume = "$x2"GB, Used = "$volperc"%"
    }
    while ($count -lt $temp1.length)
    {   if ($tvv -ne '')
        {   $FreeSpaceInVol=$tvolobj.sizeavailable
            $TotalSpaceInVol=$tvolobj.sizetotal
        }
        else
        {   $FreeSpaceInVol=0
            $TotalSpaceInVol=0
        }
        $t=$temp1[$count]
        $tnext=$temp1[$count+1]
        $tvol=$t[1..10]
        if ($t[0] -eq ' ')
        {   # skip all lines that dont start with a space. diskpart starts volume lists with spaces.
            if ($t.contains('Volume '))
            {   # does the line contain the word volume
                if ($t.contains('###'))
                {   # Must be a Title Block. Ignore it.
                }
                else
                {   if ($t[15] -ne ' ')
                    {   # its a Drive Letter
                        write-host '     '$tvol '    Drive Letter '$t[15]':\'
                    }
                    if ($tnext.contains(':'))
                    {   # Then it is a mount point
                        write-host $tvol' '$tnext
                        if ($tvv -ne '')
                        {   #discover Free Space
                            $freespace=0
                            foreach($vvv in gwmi win32_volume)
                            {  # Find the Volume in the List of Volumes from WMI
                               $x=$vvv.name
                               if ($x.length -gt 5)
                               # have to remove the simple drive letters like C:\ from the match pool
                               {   if ($tnext.contains($vvv.name))
                                   {   $freespace=$vvv.freespace
                                       $roundfree=[system.math]::round($freespace/1024/1024/1024*10)/10
                                       write-host "     Freespace = "$roundfree"GB"
                                       $perc=[system.math]::round($freespace/$tvolsize*1000)/10
                                       write-host "     Freespace percentage of Netapp Volume = "$perc"%"
```

```
                                    $tvolobj=get-navol $tvv
                                    $tvolsize=$tvolobj.sizeavailable
                                    $tvoltotal=$tvolobj.sizetotal
                                    $volperc=[system.math]::round(1000-$tvolsize/$tvoltotal*1000)/10
                                    $newperc=$perc+$volperc
                                    write-host "     NetApp Volume percentage full if I SDelete = "$newperc"%"
                                    if ($newperc -gt $threshold)
                                    { # Not Enough space free to run Sdelete
                                      write-host "    Volume is too full to continue with SDelete on it."
                                     }
                                     else
                                     { # THIS IS THE Dispart to assign letters and Run SDELETE
                                       $volnum=$t[9]+$t[10]+$t[11]
                                       write-host "     Adding Drive Letter to the Mount Point for Volume#"$volnum
$driveit = @"
select volume"$volnum"
assign letter=s
exit
"@

                                          $dump = $driveit | diskpart
                                          Write-host "     Given Drive Letter S to Volume "$volnum
                                          # Write a File to the Drive to show that an SDelete is Attempted
                                          if (test-path s:\SdeleteStart.txt)
                                          {   # then the file exists, so Sdelete has been previously run
                                              $tx = dir s:\
                                              $txdate1=''
                                              foreach ($file in (dir s:) )
                                              {    # walk through the files looking for Sdeletestart.txt
                                                   if ($file.name -eq 'Sdeletestart.txt')
                                                   {    $txdate1 = $file.LastWriteTime
                                                        break
                                                   }
                                               }
                                               Write-host "     Last SDelete Start Run was "$txdate1
                                               if ((test-path S:\SdeleteFinish.txt))
                                               {   # then the file exists, so Sdelete has been previously run
                                                   $txdate2=''
                                                   foreach ($file in (dir s:))
                                                   {    # walk through the files looking for Sdeletestart.txt
                                                        if ($file.name -eq 'SdeleteFinish.txt')
                                                        {   $txdate2 = $file.LastWriteTime
                                                            break
                                                        }
                                                    }
                                                    Write-host "     Last SDelete Finished was "$txdate2
                                                    if ($txdate1 -gt $txdate2)
                                                    {   # the Sdelete Start is Newer than Sdelete complete, must have been aborted
                                                        Write-host "     Sdelete complete is older than SDelete Start, last run must have Aborted"
                                                    }
                                               }
                                               else
                                               {   # Write that Sdelete never completed
                                                   Write-host "     Sdelete Never Completed"
                                               }
                                          }
                                          $txdate3=get-date
                                          if (test-path S:\SdeleteFinish.txt)
                                          {   if ($txdate2.adddays($daysbetween) -lt $txdate3)
                                              {   #if the timeout period of 2 days isnt expired dont run it.
                                                  Write-host "     Writing Sdelete Log to show it started"
                                                  out-file -filepath s:\SDeleteStart.txt
                                                  Write-host "     Simulated SDELETE on Drive"
                                                    start-sleep(60)
                                                  Write-host "     Writing Sdelete Log to show it Finished"
                                                  out-file -filepath s:\SDeleteFinish.txt
                                                  c:\software\sdelete -z s:
                                              }
                                              else
                                              {
                                                  Write-host "     SDelete has been completed within the Timeframe of "$daysbetween" Days"
                                              }
                                          }
                                          Else
                                          {
                                              Write-host "     SDelete has never completed"
                                              Write-host "     Writing Sdelete Log to show it started"
                                              out-file -filepath s:\SDeleteStart.txt
                                              Write-host "     Simulated SDELETE on Drive"
                                              start-sleep(60)
                                              Write-host "     Writing Sdelete Log to show it Finished"
                                              out-file -filepath s:\SDeleteFinish.txt
                                              c:\software\sdelete -z s:
                                            }
$driveit = @"
select volume"$volnum"
remove letter=s
exit
"@

                                            $dump = $driveit | diskpart
                                            Write-host "     Removed Drive Letter S from Volume "$volnum
                                          }
                                           Break   # done searching, might as well break out of the loop
                                     }
                                }
                           }
```

```
                        }
                    }
                }
            }
        }
        $count=$count+1
    }
    if ($tvv -ne '')
    {   write-host "Finished SDeleting entire WinDisk. Kicking off DeDupe on Volume"
        $dump = start-nasis /vol/DPM2Target
        Write-host "     "$tvv"     Volume Dedupe Started, Sleeping 5 Seconds"
        start-sleep -s 5
        $dds = get-nasis $tvv
        while ($dds.status -ne 'idle')
        {   write-host "    "$dds.progress
            $dds=get-nasis $tvv
            start-sleep -s 5
        }
        Write-host "      DeDuple Complete on Volume "$tvv
    }
}
```

4.3 SCRIPT TO AUTOMATE STOPPING AND STARTING THE DPM INSTANCE ON A GUEST OS

The following script moves the DPM server from an active state to a saved state. Once the DPM server is in a saved state, all of the DPM server drives may be safely snapped.

Once the volumes are snapped, the DPM server is moved back from a saved state to an active state.

```
$user = 'administrator'
$filer = '10.58.99.254'
$password = convertto-securestring "Netapp1" -asplaintext -force
$cred = new-object -typename System.Management.PSCredential -ArgumentList $user,$password
$dump = connect-nacontroller $filer -cred $cred

$vmname = "WindowsUpdateServer"
$vollist=@{'DPM1Vol1','DPM1Vol2','DPM1Vol3'}
stop-vm $vmname -force
Write-host "Saving VM State....Checking every 5 Seconds"
$notdown=$true
While ($notdown)
{   foreach ($vm in get-vm)
    {   if ($vm.VMElementName -eq $vmname)
        {   $itshere=$true
            if($vm.state -eq 'Suspended')
            {   $notdown = $false
            }
        }
    }
    if ($itshere -eq $false)
    {   write-host "Cant Find VM"
        break
    }
    Write-host '    VM Not down yet'
    start-sleep(5)
}
Write-host "    VM in Saved State"
Write-host "    Issuing Command to snapshot the Volumes"
Foreach ($vol in $vollist)
{   $t=get-date -format %y%m%d%h%m%s
    new-nasnapshot $vol $t
    Write-host "   Creating Snapshot for "$vol" called "$t
}
Write-host "Restarting VM"
start-vm $vmname
```

4.4 VB SCRIPT TO RUN A CONFIG ONLY EXPORT OF A VM

The following script is a VB script and not PowerShell. This script should be saved with a '.vbs' suffix instead of the '.ps1' suffix that PowerShell scripts use. The only difference between this script and the script that was posted on Taylor Brown's (Microsoft Hyper-V Team) blog is the removal of the interactive prompts to allow a higher level of automation. To execute this config only import, use the command:

`ConfigOnlyImport T:\DPM1\Exports DPM1.vhd DPM1`

Assuming that your VM was stored at T:\DPM1, it is named DPM1, the VHD file is called DPM1.vhd, and you are exporting your VM to a directory called T:\DPM1\Exports.

```
option explicit

dim objWMIService
dim managementService
dim switchService
dim fileSystem
const JobStarting = 3
const JobRunning = 4
const JobCompleted = 7
const wmiStarted = 4096
const wmiSuccessful = 0

Main()
'---------------------------------------------------------------------
' Main
'---------------------------------------------------------------------
Sub Main()
   dim computer, objArgs, importDirectory, generateNewID,VHDName, VMName, Suppress
   set fileSystem = Wscript.CreateObject("Scripting.FileSystemObject")
   computer = "."
   set objWMIService = GetObject("winmgmts:\\" & computer & "\root\virtualization")
   set managementService = objWMIService.ExecQuery("select * from Msvm_VirtualSystemManagementService").ItemIndex(0)
   set switchService = objWMIService.ExecQuery("select * from Msvm_VirtualSwitchManagementService").ItemIndex(0)
   set objArgs = WScript.Arguments
   if WScript.Arguments.Count = 3 then
```

```
      importDirectory = objArgs.Unnamed.Item(0)
      VHDName = objArgs.Unnamed.Item(1)
      VMName = objArgs.Unnamed.item(2)
   else
      WScript.Echo "usage: cscript ImportVirtualSystemEx-ConfigOnly.vbs importDirectoryName VHDNAME.vhd VMName"
      WScript.Quit(1)
   end if
   if (ImportVirtualSystemEx(importDirectory,VHDName,VMName)) then
      WScript.Quit(0)
   end if
      wscript.quit(0)
End Sub
'---------------------------------------------------------------------
' GetVirtualSystemImportSettingData from a directory
'---------------------------------------------------------------------
Function GetVirtualSystemImportSettingData(importDirectory)
                dim objInParam, objOutParams
                set objInParam = managementService.Methods_("GetVirtualSystemImportSettingData").InParameters.SpawnInstance_()
                objInParam.ImportDirectory = importDirectory
                set objOutParams = managementService.ExecMethod_("GetVirtualSystemImportSettingData", objInParam)
                if objOutParams.ReturnValue = wmiStarted then
                                if (WMIJobCompleted(objOutParams)) then
                                                set GetVirtualSystemImportSettingData = objOutParams.ImportSettingData
                                end if
                elseif objOutParams.ReturnValue = wmiSuccessful then
                                set GetVirtualSystemImportSettingData = objOutParams.ImportSettingData
                else
                                WriteLog Format1("GetVirtualSystemImportSettingData failed with ReturnValue {0}", objOutParams.ReturnValue)
                end if
End Function
'---------------------------------------------------------------------
' ImportVirtualSystem from a directory
'---------------------------------------------------------------------
Function ImportVirtualSystemEx(importDirectory,VHDName, VMName)

                dim objInParam, objOutParams
                dim newDataRoot
                dim importSettingData
                dim newSourceResourcePaths, newTargetNetworkConnections, newSwitch

                'Resources in newSourceResourcePaths below should be existing. Fill this with the resources corresponding to those in CurrentResourcePaths
                newSourceResourcePaths = Array(1)
                newSourceResourcePaths(0) = importDirectory & "\" & VHDName
                ImportVirtualSystemEx = false
                set objInParam = managementService.Methods_("ImportVirtualSystemEx").InParameters.SpawnInstance_()
                objInParam.ImportDirectory = importDirectory
                set importSettingData = GetVirtualSystemImportSettingData(importDirectory)
                importSettingData.GenerateNewId = true
                importSettingData.CreateCopy = false
                importSettingData.Name = VMName
                importSettingData.SourceResourcePaths = newSourceResourcePaths
                importSettingData.Put_
                objInParam.ImportSettingData = importSettingData.GetText_(1)
                set objOutParams = managementService.ExecMethod_("ImportVirtualSystemEx", objInParam)

                if objOutParams.ReturnValue = wmiStarted then
                                if (WMIJobCompleted(objOutParams)) then
                                                ImportVirtualSystemEx = true
                                end if
                elseif objOutParams.ReturnValue = wmiSuccessful then
                                ImportVirtualSystemEx = true
                end if
End Function
'---------------------------------------------------------------------
' Handle wmi Job object
'---------------------------------------------------------------------
Function WMIJobCompleted(outParam)
                dim WMIJob, jobState
                set WMIJob = objWMIService.Get(outParam.Job)
                WMIJobCompleted = true
                jobState = WMIJob.JobState

                while jobState = JobRunning or jobState = JobStarting
                                WScript.Sleep(1000)
                                set WMIJob = objWMIService.Get(outParam.Job)
                                jobState = WMIJob.JobState
                wend
End Function
'---------------------------------------------------------------------
' Create the console log files.
'---------------------------------------------------------------------
Sub WriteLog(line)
                dim fileStream
                set fileStream = fileSystem.OpenTextFile(".\ImportVirtualSystemEx-ConfigOnly.log", 8, true)
                WScript.Echo line
                fileStream.WriteLine line
                fileStream.Close
End Sub
'---------------------------------------------------------------------------
' The string formating functions to avoid string concatenation.
'---------------------------------------------------------------------------
Function Format1(myString, arg0)
    Format1 = Replace(myString, "{0}", arg0)
End Function
```

4.5 VB SCRIPT TO RUN A CONFIG ONLY IMPORT OF A VM

The following script allows the failover from site to site of a DPM server. This will automatically fail the DPM instance to the Hyper-V host from which the script was executed. After the functions are defined you will encounter a variable block that can be edited to reflect the specifications of your deployment. The following information will be needed to complete the variable block:

- Name of the host you are failing from (variable named $FromHost)
- Name of the controller being failed from (variable named $FromFiler)
- Name of the controller being failed to (variable named $ToFiler)
- Name of the SnapMirror session source (variable named $FromSnap)
- Name of the SnapMirror session target (variable named $ToSnap)
- Collection of the volumes to be relocated (variable named $VolName)
- Boot drive letter for the VM with its VHD (variable named $BootDriveLetter)
- Admin/Password for the controllers ($admin and $password)
- VHD path to the boot drive of the DPM server ($VMVHD)
- Virtual machine path ($VMPATH)

```
Function AddPassThroughs($WinDiskNum)

{               # Function Adds Passthroughs of the Windisk NUM passed to it to the VM Named $VMNAME
                $VMManagementService = Get-WmiObject -class "Msvm_VirtualSystemManagementService" -namespace "root\virtualization"
                $VM = Get-WmiObject -Namespace "root\virtualization" -Query "Select * From Msvm_ComputerSystem Where ElementName='$VMName'"
                $VMSettingData = Get-WmiObject -Namespace "root\virtualization" -Query "Associators of {$VM} Where ResultClass=Msvm_VirtualSystemSettingData
AssocClass=Msvm_SettingsDefineState"
                $VmScsiController = (Get-WmiObject -Namespace "root\virtualization" -Query "Associators of {$VMSettingData} Where ResultClass=Msvm_ResourceAllocationSettingData
AssocClass=Msvm_VirtualSystemSettingDataComponent" | where-object {$_.ElementName -eq "SCSI Controller"})
                $DiskAllocationSetting = Get-WmiObject -Namespace "root\virtualization" -Query "SELECT * FROM Msvm_AllocationCapabilities WHERE ResourceSubType = 'Microsoft Physical
Disk Drive'"
                $DefaultHardDisk = (Get-WmiObject -Namespace "root\virtualization" -Query "Associators of {$DiskAllocationSetting} Where ResultClass=Msvm_ResourceAllocationSettingData
AssocClass=Msvm_SettingsDefineCapabilities" | where-object {$_.InstanceID -like "*Default"})
                $Disk = Get-WmiObject -Namespace "root\virtualization" -Query "select * from Msvm_DiskDrive Where DriveNumber=$WindiskNum"
                $DefaultHardDisk.Parent = $VmScsiController.__Path
                $DefaultHardDisk.Address = $WinDiskNum
                $DefaultHardDisk.HostResource = $Disk.__PATH
                $VMManagementService.AddVirtualSystemResources($VM, $DefaultHardDisk.PSBase.GetText(1))
}

Function ControllerConnect($filer)
{               # connect to a controller and handel error checking
                if (Connect-naController $filer -credential $cred)
                {
                                Write-host "- ---Connected to Array"$Filer" Successfully"
                }
                else
                {

                                Write-host "- ---Failed to connect to Array"$Filer", script aborting"
                                break
                }
}

function FindHBA($Server)
{               # Given a Server Name, returns its IGroup on the Array
                $IGrouplist=get-naigroup
                Foreach ($ig in get-naigroup)
                {               # makes HUGE Assumption that the DPM server name is NOT a subset of any other servername on the array
                                # makes HUGE Assumption that the DPM server IGroup is NAMED after the server name.
                                $GroupName=$ig.InitiatorGroupName
                                if ($GroupName.contains($FromHost))
                                {               if ($Server -eq $Fromhost)
                                                {
                                                                Write-host "- ---Found Host "$Server" in IGroup "$GroupName
                                                                $supress=$FromHBA.Add($GroupName)
                                                }
                                }
                                if ($GroupName.contains($ToHost))
                                {
                                                if ($server -eq $ToHost)
                                                {
                                                                Write-host "- ---Found Host "$Server" in IGroup "$GroupName
                                                                $supress=$ToHBA.Add($Groupname)
                                                }
                                }
                }
}

function IsAlreadyMapped ($IAMLUNPATH, $IAMHBA)
{   # returns a True or False. If the LUN is current Mapped to that IGroup
                $IAMX = get-nalunmap $IAMLUNPATH
```

```
                $IAMResult=$FALSE
                Foreach ($IAMY in $IAMX)
                {
                        if ($IAMY.InitiatorGroupName -eq $IAMHBA)
                                {
                                        $IAMResult=$True
                                }
                }
                Return $IAMResult
}

function CheckSnapMirrorBusy
{               # Will take a Snapmirror name and cycle until it returns as IDLE
                Param ($Snapname)
                $x1=get-nasnapmirror $Snapname
                $x2=$x1.state
                if ($x2 -eq "broken-off")
                {
                        write-host "- ---SnapMirror Session Already Broken Off"
                        return $false
                }
                $x2=$x1.status
                while ($x2 -ne "idle")
                {
                        write-host "- ---Waiting until SnapMirror Session is Idle."
                        start-sleep(5)
                        $x2=get-nasnapmirror $Snapname
                        $t2=$x2.status
                }
                write-host "- ---SnapMirror Session is Idle"
                return $true
}

function FindWinDisks
{               $FWDdrives                      =               gwmi Win32_diskdrive
                $FWDWinDisk     =               New-Object System.Collections.ArrayList
                #find the Boot Drive and make sure it doesnt get added.

                foreach($FWDdisk in $FWDDrives)
                {               Foreach ($FWDLun in get-naLun)
                                {   # Check if LUN belongs to Volume owned
                                        $FWDLUNSerNum = get-nalunserialnumber $FWDLun.path
                                        if ($FWDdisk.SerialNumber -eq $FWDLUNSerNum)
                                                {
                                                                foreach ($FWDvol in $VolName)
                                                                {
                                                                                $FWDLunPathName=$FWDlun.path
                                                                                if ($FWDLunPathName.contains($FWDvol))
                                                                                                {               $FWDdiskname=$FWDdisk.name -replace
'\\\\.\\PHYSICALDRIVE',''

                                                                                                                if
($FWDLunpathName.Contains('Boot'))
                                                                                                                                {

                write-host "- ---   Found WinDisk#"$FWDdiskname" LUNPath ="$FWDLunPathName", Vol Name ="$FWDVol" Contains Boot Vol. SKIP"
                                                                                                                                }
                                                                                                                                else
                                                                                                                                {


                $FWDWindisk.Add($FWDdiskname)

                write-host "- ---   Found WinDisk#"$FWDdiskname" LUNPath ="$FWDLunPathName", Vol Name ="$FWDVol
                                                                                                                                }
                                                                                                                # this Windisk Lives inside one of
the VOLs that is being moved
                                                                                                }
                                                                }
                                                }
                                }
                }
                return $FWDWindisk
}

$ToHost                                                 = gc env:computername
$VMExportPath                           = "N:\Export\DPM2"
$VMName                                                 = "DPM2"
$VMPath                                                 = "N:\DPM2"
$VMVHD                                                  = "DPM2.vhd"
$VolName                                                = @("DPM2")
$BootDriveLetter                        = "N"
$admin                                  = "administrator"
$password                               = ConvertTo-SecureString "Netapp1" -AsPlainText -Force
$cred                                                   = New-Object -TypeName System.Management.Automation.PSCredential -ArgumentList
$admin,$password
$FromHBA                                                = New-Object System.Collections.ArrayList
$ToHBA                                                  = New-Object System.Collections.ArrayList
                if ($ToHost -eq 'HVVMDC0-04')
                {
                        $FromHost                                                       = "HVVMDC0-03"
                        $FromFiler                                                      = "fas2040bot"
                        $ToFiler                                                        = "2040top"
                        $FromSnap                                                       = "fas2040bot:DPM2"
                        $ToSnap                                                                 = "2040top:DPM2"
                }
                If ($ToHost -eq 'HVVMDC0-03')
```

```
                {
                                $FromHost                                                       = "HVVMDC0-04"
                                $ToFiler                                                        = "2040top"
                                $FromFiler                                                      = "fas2040bot"
                                $ToSnap                                                                 = "2040top:DPM2"
                                $FromSnap                                                       = "fas2040bot:DPM2"
                }


#Banners
cls
write-host "-------------------------------------------------"
write-host "- Starting Failover from DR Site to Main Site    -"
write-host "-------------------------------------------------"
Write-host "- --- To Hostname   = "$ToHost
Write-host "- --- From Hostname = "$FromHost
Write-host "- --- To Array      = "$ToFiler
Write-host "- --- To Snap       = "$ToSnap
Write-host "- --- From Array    = "$FromFiler
Write-host "- --- From Snap     = "$FromSnap
write-host "- ---Stoping VM "$VMName"on Site Migrating From : "$FromHost
$suppress = stop-vm $VMName -server $FromHost -force
Write-host "- ---Saving VM State....Checking every 5 Seconds"
Write-host "- ---VM in Saved State"
write-host "- ---Removing Actual VM from Hyper V on Main Site : "$VM
remove-vm $VMName -server $FromHost -force
ControllerConnect($FromFiler)
FindHBA($FromHost)
Write-host "- ---Found "$FromHBA.count" Igroups in Array for Server"
Write-host "- ---Removing Mapping from server on from site"
Foreach ($Lunname in get-naLun)
{   # Remove the old From Site LUN Mapping
                $LunNamePath=$Lunname.path
                foreach ($vol in $VolName)
                {               if ($LunNamePath.contains($vol))
                                {               #Filters the LUN List by only the Volume I care about
                                                # this LUN Lives inside one of the VOLs that is being moved
                                                foreach ($HBA in $FromHBA)
                                                {               if (IsAlreadyMapped $Lunnamepath $HBA )
                                                                {
                                                                                $Supress=remove-nalunmap $LunNamePath $HBA
                                                                                Write-host "- ---   Removing Access to "$LunPathName" via iGroup "$HBA
                                                                }
                                                                Else
                                                                {
                                                                                Write-host "- ---   "$HBA " Host Currently not Connected to LUN "$LunNamePath
                                                                }
                                                }
                                }
                }
}
write-host "- ---Updating Mirror Relationship"
ControllerConnect($ToFiler)
Write-host "- ---   Checking SnapMirror for Idle from "$FromSnap
#if (CheckSnapMirrorBusy($FromSnap))
#{              $suppress = invoke-NaSnapMirrorUpdate -source $FromSnap -destination $ToSnap
#}
Write-host "- ---   Checking SnapMirror for Idle from "$FromSnap
if (CheckSnapMirrorBusy($FromSnap))
{               Write-host "- ---Breaking Mirror to Expose the To site"
                $Suppress = invoke-NaSnapMirrorBreak -destination $ToSnap
                start-sleep(1)
}
write-host "- ---Mapping LUN from Secondary Array to Secondary Cluster"

ControllerConnect($ToFiler)
FindHBA($ToHost)
Write-host "- ---Found "$findHBA.count" Igroups in Array for Server"
Write-host "- ---Removing Mapping from server on from site"
Foreach ($Lun in get-naLun)
{   # Add to new site the LUN Mapping
                foreach ($vol in $VolName)
                {
                                $LunPathName=$lun.path
                                if ($LunPathName.contains($vol))
                                {
                                                # this LUN Lives inside one of the VOLs that is being moved
                                                ForEach ($HBA in $ToHBA)
                                                {               if (IsAlreadyMapped $LunPathName $HBA )
                                                                {
                                                                                write-host "- ---   Already Mapped to "$HBA
                                                                }
                                                                Else
                                                                {
                                                                                $supress=add-nalunmap $LunPathName $HBA
                                                                                Write-host "- ---   Adding Access to "$LunPathName" via iGroup "$HBA
                                                                }
                                                }
                                }
    }
}
write-host "- ---Initiating DiskPart Rescan"
start-sleep(5)
$suppress = "rescan" | Diskpart
write-host "- ---Updating the Reversed Mirror. Secondary Site and Primary Site roles now reversed"
```

```
ControllerConnect($FromFiler)
$supress = Invoke-NaSnapMirrorResync -Source $ToSnap $FromSnap
# Check to make sure the VM isnt already running
if (get-vm $VMName)
                { Write-host "- ---   VM is currently exists"
                }
Else
{               # Import the Config only VM
                xcopy $VMExportPath $VMPath /s /y
                write-host "- ---Import The VM via a Config Only VM Import. Specify existing in place VHD"
                C:\Software\ConfigOnlyImportv3 $VMPath $VMVHD $VMName
                start-sleep(5)
}
Write-host "- ---Finding WinDisks that Belong to this VM"
ControllerConnect($ToFiler)
$windisk=findwindisks
foreach ($WD in $WinDisk)
                {
                                $Suppress=AddPassThroughs $WD
                }
Write-host "- ---Starting VM"
                # Start-VM $VMname

write-host "--------------------------------------"
Write-host "- --- Failback to Main Site Complete -"
write-host "--------------------------------------"
```


www.netapp.com