Veritas Storage Foundation™ and Thin Provisioning

Rationale, Configuration and Benefits

March 2008

Scott Kaiser and Oscar Wahlberg

TABLE OF CONTENTS

Executive Summary

Data centers are increasingly turning to “thin provisioning” to simplify operational tasks and save capital budget. Thin provisioning, also known as dynamic provisioning, creates a virtual disk drive that appears, to applications and servers, much larger than the actual physical storage allotted to it. As applications write to this virtual disk, the storage array automatically allocates more physical storage. The process is transparent to applications and servers. This lets actual storage demand, rather than forecasts (which tend to overestimate), determine how much physical storage is used.

While thin provisioning is transparent to applications and servers, the reverse is not true – the behavior of the applications and the server does impact how ‘thin’ the storage can be. That behavior varies substantially across different vendors. Veritas Storage Foundation provides substantial benefits for companies using or planning to use thin provisioning.

A key advantage of Storage Foundation comes when migrating data from legacy storage to thin provisioning storage. Traditional migration tools copy both data and empty space, resulting in a “thin” provisioning system that uses just as much storage as the legacy system. In contrast, Storage Foundation strips out empty storage as it migrates data from traditional storage to thin provisioned storage. This avoids unnecessary ‘grow’ events on the thin provisioning array.

Due to its thin provisioning-friendly algorithms and architecture, Storage Foundation triggers fewer ‘grow’ events on the storage array than other alternatives. This results in better storage utilization, less allocated but unused storage, and lower costs. Storage Foundation accomplishes this with smart architectural choices, such as just-in-time metadata, compact addressing, and in-place overwrites. And, when files are deleted, Storage Foundation re-uses the space freed by those files before writing to new locations. These techniques reduce the amount of physical storage that the array allocates. Hewlett-Packard’s StorageWorks Division recently published a thin provisioning guide that gave high marks to Storage Foundation’s utilization efficiency when used with HP’s thin provisioning array.

Storage Foundation supports all thin provisioning architectures currently available, including 3PAR, Hitachi, Hewlett-Packard XP, Network Appliance, and Sun. Storage Foundation has the broadest hardware compatibility list on the market, which lets customers standardize on one toolset for their data center to reduce complexity.

When designing a thin provisioning architecture, consider whether your host-based storage tools add to, or subtract from, the benefits of thin provisioning. Choosing the right ones can be the difference between enjoying the savings that thin provisioning offers, and being asked to figure out why the actual savings were less than forecast.

Scope & Audience

This paper explains the value that Veritas Storage Foundation software can bring to a data center using, or planning to use, thin provisioning. It conceptually describes how Veritas software technically accomplishes these benefits. It does not attempt to place a specific financial value on those benefits. If you are interested in such an analysis, contact your Symantec Account Manager. It also provides detailed examples of the command line operations to perform on the host to migrate legacy data to thin provisioning. It does not explain how to set up thin provisioning itself, as those steps differ among different array vendors.

This paper is intended for engineering/architecture and operations groups for storage and for UNIX servers in large enterprise data centers who are deploying or who are evaluating thin provisioning.This paper also provides how-to steps and examples that an individual system administrator or an operations group can use as an implementation guide. Technical staff may also find this paper of use in describing the benefits of Storage Foundation to their management.

This paper assumes basic knowledge of data center operations and experience with host-based storage management tools, such as file systems, logical volumes, and LUNs. It also assumes basic understanding of thin provisioning, also known as dynamic provisioning.

Veritas Storage Foundation Overview

Veritas Storage Foundation is an integrated software suite providing core storage capabilities for servers. Major Storage Foundation components include the Veritas File System (VxFS), a physical file system; Veritas Volume Manager (VxVM) which creates and manages logical volumes comprised of LUNs or physical disks; Dynamic Multipathing (DMP) to provide load balancing and I/O error recovery on SANs; and Storage Foundation Manager, with central management and task workflow of the other components.

Thin Provisioning Overview

Thin provisioning relies on the observation that most organizations over allocate storage. They do so for various reasons. Running out of storage can be career threatening for the administrator of a mission critical application, whereas the cost of using too much storage may be on someone else's budget. Every developer anticipates their new application will be a killer app used by everyone, but most won't be. Getting storage early in a project when funds are allocated is easier than getting it later, when funds may be canceled. If the server operations and storage operations teams are split, the request for more storage may have a long lead time and may cause friction. For these and for other reasons, organizations over allocate storage.

Thin provisioning offers an alternative. It creates virtual disk drives – LUNs, for Logical Units – that appear to be one size, but whose actual physical storage only covers a fraction of their claimed size. If a LUN needs more storage, the storage array allocates more physical storage to it, without changing the presented size[1].

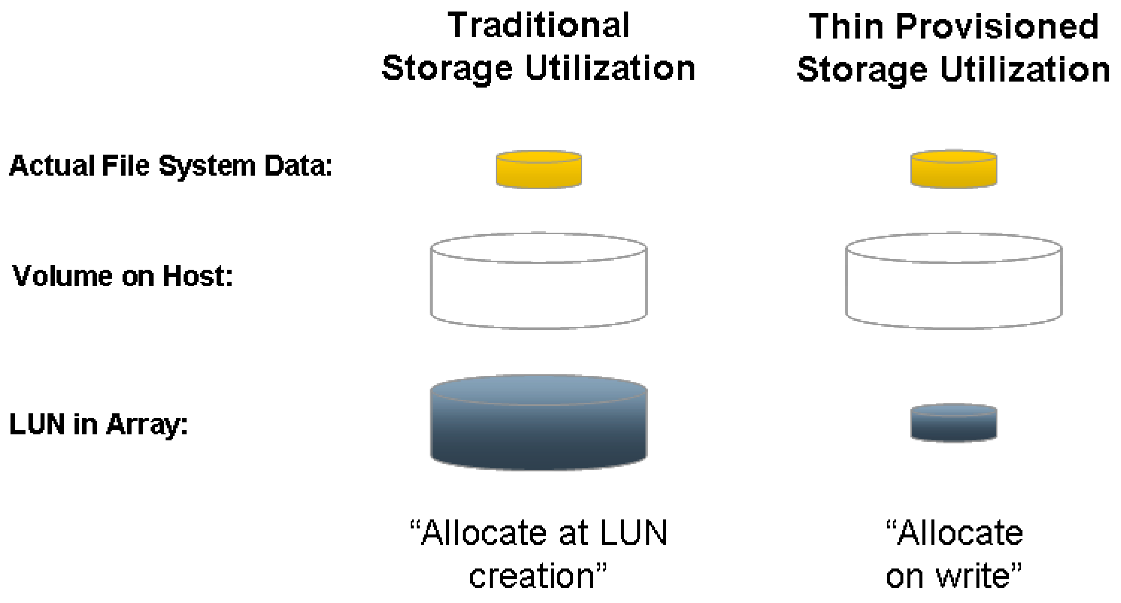

Traditional Storage Utilization
Thin Provisioned Storage Utilization
Actual File System Data:
Volume on Host:
LUN in Array:
"Allocate at LUN creation"
"Allocate on write"

Figure 1: Traditional and Thin provisioning. The storage used in the thin scenario is the same as the file system (best case); actual storage usage can be higher, as later sections explain.

[1] Some wags may prefer 'pretended size'.

Thin Provisioning Concepts & Related Concepts

Three concepts help explain thin provisioning. The first is 'first write', a technical concept. Block devices, such as disk drives and LUNs, are sequentially numbered, linear arrays of blocks. Blocks are fixed size, usually 512 bytes. Higher-level applications group blocks into files, which are easier to use. As a result, block devices have no idea whether a given block is used or is not used. So, they must assume all blocks are used. Thin provisioning's inventors realized that the block device could watch for the first write to a block. Before that first write, the block is not in use; forever after, the storage array considers the block in use.

The second concept is inventory efficiency. This operational concept directly impacts capital expense. After reading about the risk of running out of storage, readers may wonder if this is more likely with thin provisioning than with traditional storage. In fact, the opposite is true. From a business perspective, storage management is an inventory problem. You don't want to run out – it costs the enterprise and the staff - so data centers keep unused storage at the ready[2]. But keeping unused storage is also expensive. Administrators are loath to reduce the amount of spare storage they have, because that increases the risk of running out.

But cost can be lowered while maintaining or even reducing the risk. As spare storage – or any inventory – fragments into small pieces, more must be kept to avoid running out at any one location. By merging spare storage into larger groups, less storage is needed for the same risk of running out -- as long as it can be quickly moved where needed. This concept – keep fewer, larger buckets of inventory – reduces storage without increased risk[3].

Thin provisioning storage arrays use this principle to keep spare storage for all applications using that array. This makes more spare storage available to all applications. Thin provisioning skips administrative steps like making the new storage visible to the server and growing the volume and file system[4]. Couple this with the array automatically allocating storage, and spare storage goes rapidly where it is needed – a requirement when storage moves from small buckets close to the application to large buckets farther from the application.

Traditional storage is usually managed in one of two ways. The first is to fully allocate storage close to the application, creating many small pieces of free storage. For applications with highly predictable storage needs, like some scientific environments, this may work well. Everyone grows a proportional amount and keeps a proportional spare storage buffer. Note that thin provisioning is no worse than traditional storage in this scenario.

If there is any variability or unpredictability in storage usage across applications – typical in most commercial and government environments – the traditional storage method performs poorly. Applications that use more storage wind up with less spare storage; other applications have more than they need. Thin provisioning, on the other hand, can balance based on actual usage, keeping all spare storage available to any application that might need it. This results in less risk of running out of storage than with traditional storage.

The second way to manage traditional storage is to keep spare storage in large chunks, monitor usage closely, and manually move it as needed[5]. This is conceptually similar to thin provisioning, but it places a large burden on the data center staff. Thin provisioning requires fewer provisioning steps, giving it a time and effort advantage. Plus, the datacenter staff monitors fewer

[2] If you never want to run out of storage, the math is clear: keep an infinite amount of unused storage, which in turn requires infinite floor space, energy, cooling, and capital budget.
[3] The authors avoid the term "pool", because every storage vendor offers "pools" these days regardless of whether they provide an inventory benefit.
[4] More accurately, these steps are done up front.
[5] In practice, data centers operate between the "allocate everything" and "allocate nothing" extremes.

objects. They only monitor each thin provisioning array's spare storage, rather than storage usage for thousands of database tables and file systems. Thin provisioning has the advantage over traditional storage in this management paradigm, too.

The third concept is uncorrelated demand, an operational issue. Ultimately, a thin provisioning storage array can only provide as much physical storage as it has available. Imagine two servers, each using a single thin LUN. Each thin LUN appears to be 10GB; each LUN has 8GB of physical storage allocated. One storage array provides both LUNs, and that array has 2GB free to allocate, in addition to the 16GB provisioned to the two LUNs (Figure 2). Server 1 hosts a retail internet database, serving US customers. This application is busiest in the fourth quarter due to holidays. Server 2's application is used by the finance department for quarterly reporting. It is busiest the first two weeks of each quarter. Imagine the array allocates its last 2GB of available storage to Server 1's thin LUN in December. Because Server 2 is not likely to need more storage until January 1 – its demand is uncorrelated with Server 1 – the storage operations team has time to order more storage[6].

If Server 1 was a General Ledger application, the many accounting entries required to close the quarter would arrive – and use storage – at about the same time that Server 2's quarter-end financial reporting system would consume storage. Because both applications need more storage at the about same time, they are more likely to break the bank – using up their allotted 8GB only to discover that the other application's LUN has already been given those last 2GB of storage. In this counterexample, the applications' storage demand is correlated.

In practice, organizations do not group applications by uncorrelated storage demand. But the same principle is now being used to balance virtual machines' CPU loads, so the concept is becoming familiar. Since thin provisioning benefits more than traditional storage, the concept is more important to these environments. The Further Reading section has more information.

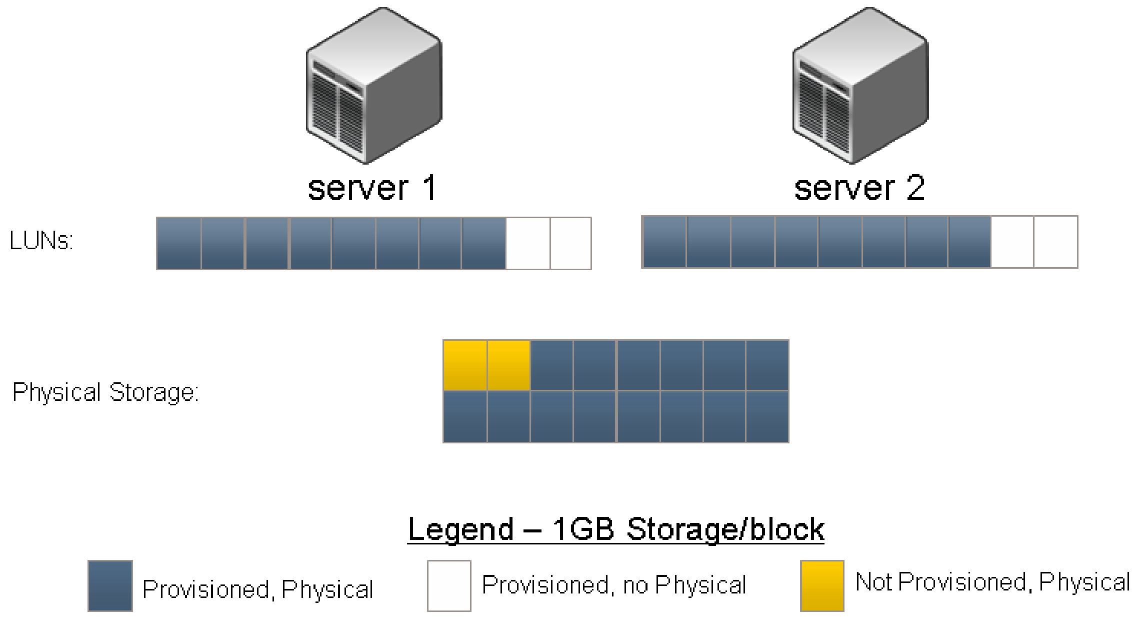


Figure 2: Illustration for Correlated and Uncorrelated Demand.

[6] This example illustrates the concept, not real life in a data center. Few administrators willingly run with margins this thin. But even with large safety margins, risk is further reduced by matching applications that consume storage at different times. The tendency for separate departments to want dedicated storage makes this harder to accomplish, since storage demand is often correlated within a department.

Migrating from Legacy Storage

"How long does getting thin take?"

– A. A. Milne

Legacy Migration – the Overlooked Challenge

Amid growing enthusiasm for thin provisioning and its benefits, the issue of migrating from legacy storage is often overlooked or glossed over. Recall how thin provisioning works: the storage array treats each block as empty until the first write. Legacy storage arrays do not track whether a block has been written to or not. So how can you migrate from a legacy array to a thin provisioning array without moving every block?

As shown below, migrations with traditional logical volume managers cannot; they must copy every block from the original LUN and write it to the new LUN, effectively filling up the new, thin LUN and defeating the whole point of thin provisioning.

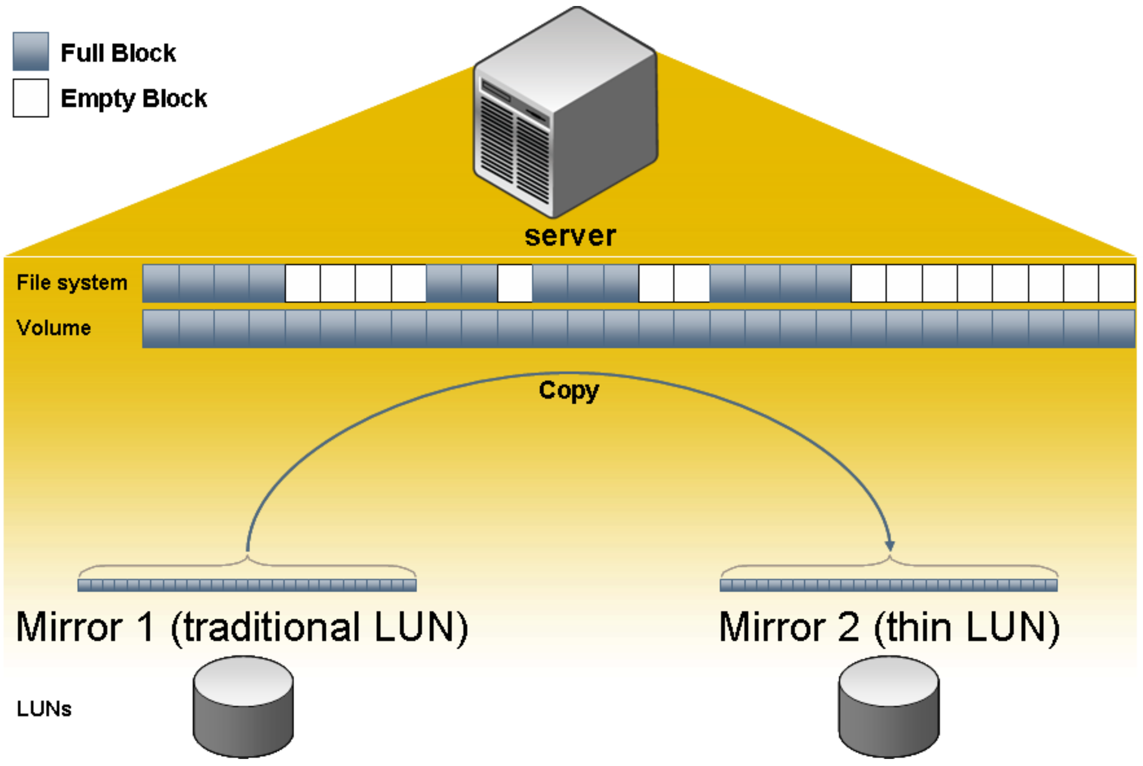


Figure 3: A traditional Logical Volume Manager Migration.
Even empty blocks are copied.

Some organizations are evaluating "virtual" LUNs. These vLUNs are presented by a switch, controller or appliance, which in turn mirrors or concatenates LUNs created by more traditional storage arrays. Unfortunately, these approaches must also copy all the blocks during migrations into thin LUNs, as shown below.

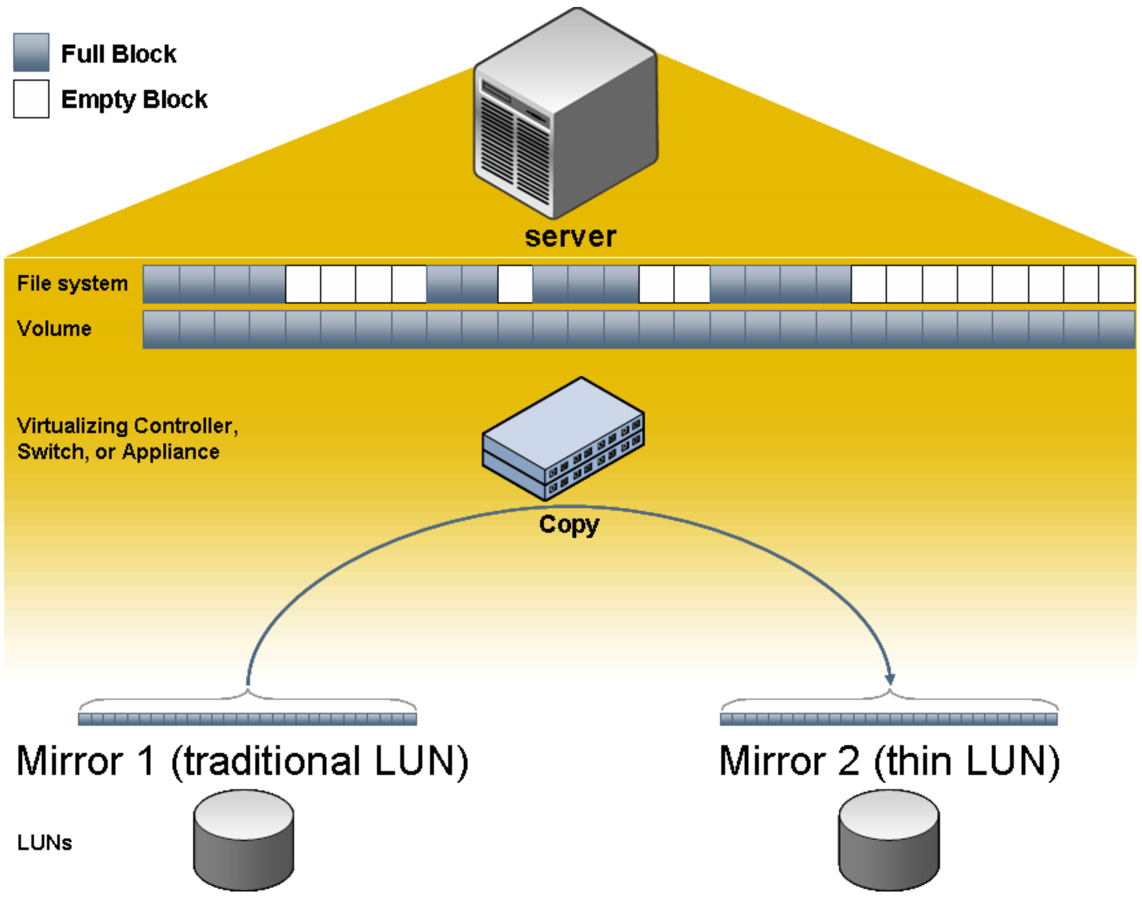

Full Block
Empty Block
server
File system
Volume
Virtualizing Controller, Switch, or Appliance
Copy
Mirror 1 (traditional LUN)
Mirror 2 (thin LUN)
LUNs

Figure 4: Switch-, Appliance-, and Controller-based Virtualization. Again, empty blocks are copied, defeating the value of thin provisioning.

The solution to this problem is to move up to a software level that knows if a block is occupied or not: the file system. File systems' basic job is to remember which blocks make up each file. They must also track which blocks are free, in order to make new files or grow existing ones. As a result, file systems know which blocks are free, and which are not.

Veritas Storage Foundation uses this knowledge to enable migrations from legacy storage to thin provisioning, thus reclaiming any free space. Three methods are available, described below. All three use Veritas Storage Foundation's basic storage migration capability, which has been in the product since its first release in 1990. This migration capability has been used in production by thousands, if not tens of thousands, of customers.

Storage Foundation's migration engine uses a fundamental capability called mirroring. Mirroring makes one block device (like a LUN) an identical copy of another block device. Each mirror can come from a different storage array, even if the arrays are made by different vendors or if one LUN is bigger[7]. Once the copy is complete, the mirrors are in perfect sync with each other; when a write request is received, it is not returned until all mirrors have acknowledged its completion. Mirrors can be added and removed while an application is online and performing I/O to the storage. Basic array migration adds a new mirror representing the new target array to which the data will be migrated. Once that new mirror is in sync with the original mirror, the original mirror, from the older source array, is removed.

In large configurations with hundreds or thousands of LUNs, the migration tasks of

[7] The source LUN must be smaller than the target LUN. Once the smaller LUN is removed, the volume can be grown to the size of the larger LUN, while the application is online and performing I/O. This is useful when migrating to a thin LUN that is much larger than the existing legacy LUN.

adding and removing mirrors can be simplified using Veritas Storage Foundation Manager. To learn more about Storage Foundation Manager, see the Further Reading section.

Solve Legacy Migrations with Veritas SmartMove™

Storage Foundation knows which blocks are free and which are not. It uses this knowledge during array migrations. When Veritas Volume Manager (VxVM) starts a migration, it requests the free space map from the Veritas File System (VxFS). This is transparent to users and administrators, and the migration steps are performed identically to traditional array migrations. This facility is known as Veritas SmartMove™. When creating the new mirror on the thin LUN, Storage Foundation only copies the occupied blocks. It does not copy the free blocks.

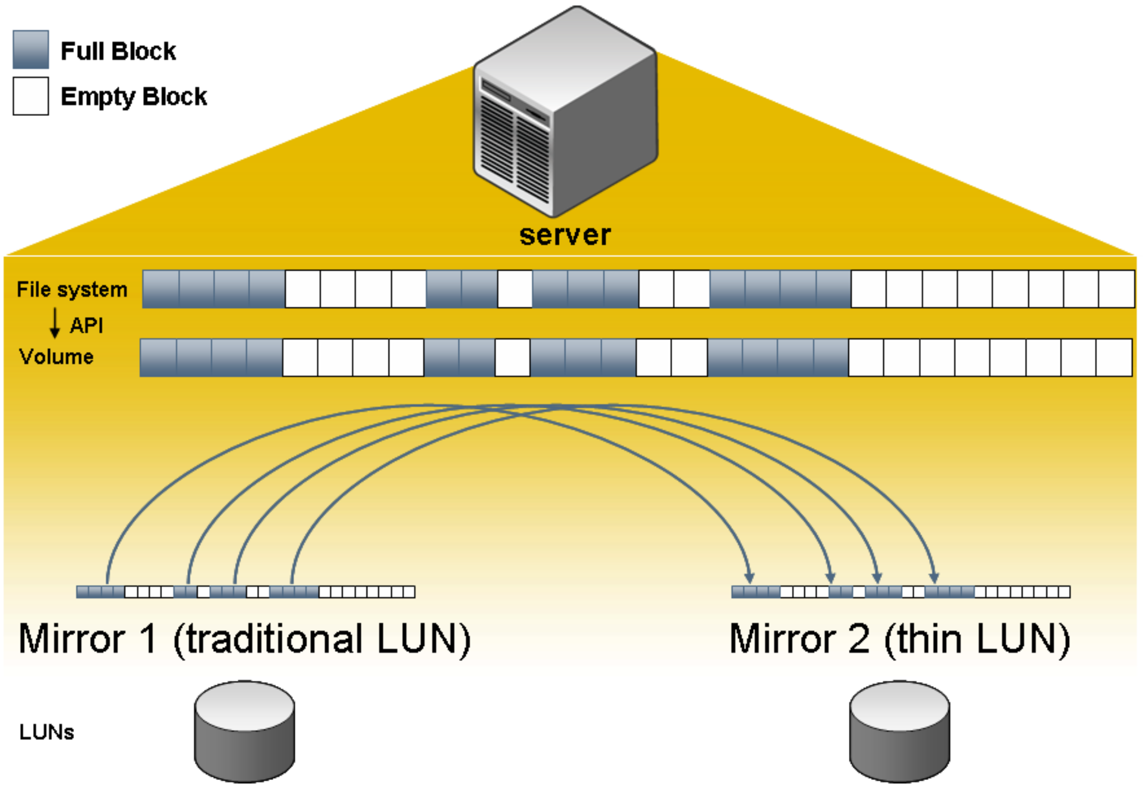


Figure 5: SmartMove™ only migrates occupied blocks

SmartMove™ is the simplest and fastest mechanism for migrating to thin provisioned environments. It requires Storage Foundation version 5.0, Maintenance Pack 3 or higher. Sample administrative commands for Veritas SmartMove™ are given in Appendix 1. The SmartMove™ method should be used whenever Storage Foundation 5.0MP3 is available, because it is fastest and simplest. When it is not available, use one of the other methods described below.

Solve Legacy Migrations with Reclaim-before-move

The legacy data migration challenge can also be solved using an approach we term 'reclaim-before-move'. This approach requires more administrative effort, but is available with all supported Storage Foundation versions, plus many older versions.

In this method, the copy operation does not distinguish between empty and full blocks. Rather, the file system and volume is shrunk just before the migration, so that there are fewer empty blocks to be copied. After migration, they grow back to their original size. Both the shrink and the grow happen online, while the application is running. Aside from the lack of SmartMove™ and the extra grow and shrink steps, this method uses the same array migration operation described previously. Veritas Storage Foundation mirrors the LUN on the legacy, traditional, source array to a LUN on the new, thin, target array.

Like SmartMove™, file system knowledge of whether a block is free or full is used; but it is used to reduce the number of empty blocks. All occupied blocks are kept during the shrink. The source LUN does not shrink; Storage Foundation divides the LUN into one used and one unused region, and then copies only the used region[8].

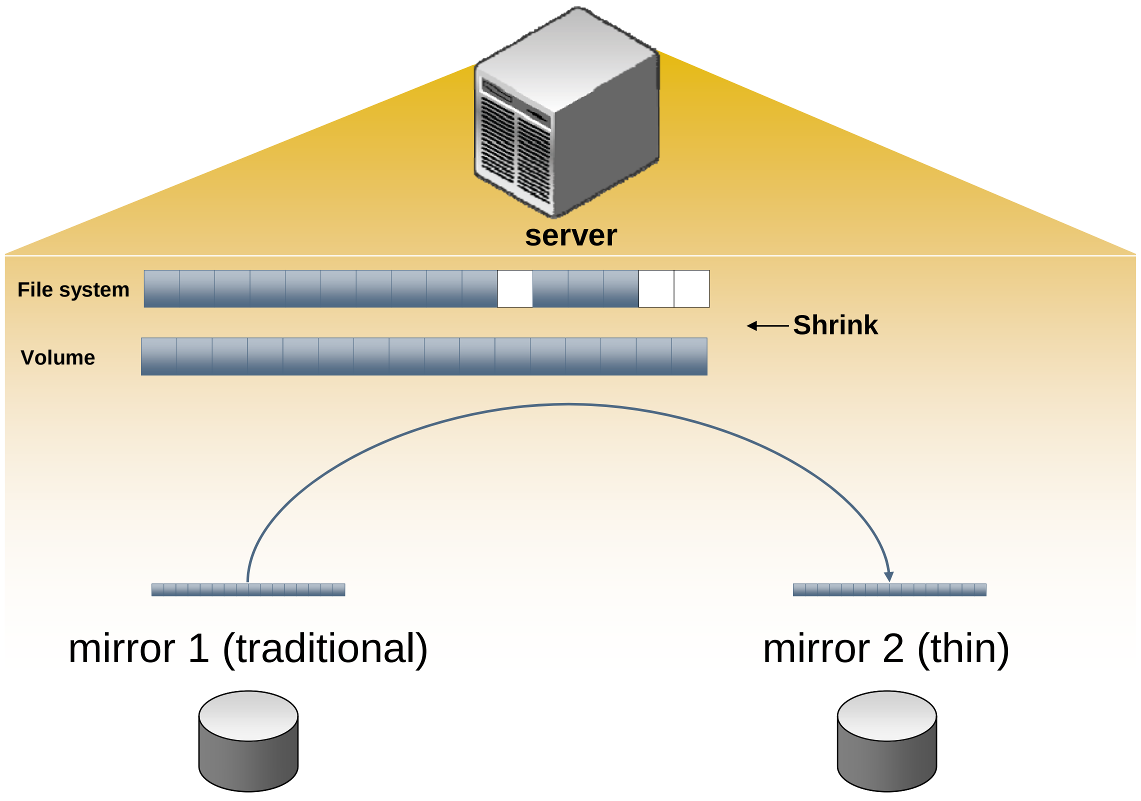


Figure 6: Reclaim, then migrate.
The next step is to grow the volume and file system back to their original size (not shown).

Solve Legacy Migrations with Multi-volume File Systems

A third method for performing migrations from traditional storage to thin storage is to use Storage Foundation's multi-volume file system capability. This method differs from the previous two approaches by requiring an application outage, as well as more administrative work. The authors offer one use case that is not handled better by the previous methods. Thoughtful readers may find others.

This use case is where the administrator plans to use a two-tier Dynamic Storage Tiering (DST) implementation with their thin provisioning. The Storage Tiering section of this paper explains DST in more detail. By using this multi-volume method for the legacy migration, the administrator combines the legacy storage migration with the migration to DST.

[8] At shrink, VxFS tell VxVM "this tail-end of the LUN is unused." This is a much less granular communication than SmartMove, essentially drawing a line at the volume's end and marking the rest free.

Storage Foundation offers multi-volume file systems where a single file system can store data in more than one volume[9]. By shifting data between two volumes in the same file system, a traditional-to-thin migration can occur. The procedure is covered in detail Appendix 1.

Enhance Legacy Migrations with Oracle

Some applications can improve legacy migrations. Databases are especially relevant because they often pre-allocate storage – often months or years worth. In these cases, less storage can be reclaimed during a legacy migration, even with Veritas' advanced technologies. This is because the database files are opaque to Veritas Storage Foundation; conservatively, Storage Foundation must assume they are completely full, and copy them accordingly, even though actual data is usually a small fraction of these files. In such an environment, the amount of space that can be reclaimed is approximately equal to the file system free space.

Oracle's relational database added a storage reclamation capability in its 10g release. Many IT professionals, including database administrators, are unaware that it exists. This feature allows additional unused storage to be reclaimed during a migration from legacy storage to thin provisioning.

Oracle Database 10g's feature is very similar to VxFS shrink, but at the application level rather than at the file system level. As a result, the procedure to reclaim storage during a migration is conceptually similar to the reclaim-before-move approach discussed above. Because Oracle 10g does not have a file system copy mechanism, Oracle's file shrink feature improves migrations from legacy storage to thin provisioning, but does not perform the migration itself.

The basic approach is as follows. Once the new, thin provisioning storage is created, but before the migration and copy operations start, the DBA shrinks those Oracle files with significant free space, leaving enough free space for normal operations. The system administrator then initiates the migration using any of the three previously described methods, such as SmartMove™. The legacy storage can then be removed[10].

For more information on Oracle 10g file shrink, see the Further Reading section.

What If I Don't Have Storage Foundation? The High Cost of "Free"

Other vendors have long advised customers to use their own tools, rather than Storage Foundation, often by claiming that their tools are free. This claim ignores a key business concept, known as Total Cost of Ownership, or TCO. TCO includes all relevant costs for a business process or function, not just acquisition costs. Since other vendors' tools are unable to help migrate from legacy environments into thin environments, and since many of these tools also use thin provisioning less efficiently than does Storage Foundation, storage costs are higher, and hence their TCO is higher. Since storage hardware costs are typically much larger than Storage Foundation license costs, the net result can be negative for the datacenter.

Organizations seeking a lower TCO often face political challenges. For example, if a server group has the host software budget, and a storage group has the storage hardware

[9] This can be an unusual concept, particularly for those with years of experience of having one file system built on one volume. It can be demystified by thinking about how a file is addressed. The Veritas file system stores file data in extents, an extent being a range of blocks. The addressing for an extent is straightforward: <first block, length in blocks>. With a multi-volume file system, the addressing becomes <volume, first block, length in blocks>.

[10] Once mirroring is finished, the DBA can grow the files to their original size or to a larger size if the thin LUN is substantially larger than the original. The DBA can also leave the files at their new, smaller size and use Oracle auto-extend to manage the space.

budget, then a solution which results in slightly higher costs to the server group but much lower costs to the storage group may require coordination between both groups and their management.

If the legacy storage is not managed by Storage Foundation, there are options. Storage Foundation offers tools to convert many vendors' file systems and volumes to Storage Foundation. These tools require administrative work and application downtime, but upon completion, the systems enjoy Storage Foundation's many benefits. For more conversion information, see the Further Reading section.

Conversion costs may be high because of the downtime and the administrative work; whether the savings from thin provisioning and from the other benefits of Storage Foundation will offset those costs will vary by data center. This paper argues it is better to deploy and use Storage Foundation from the beginning.

Legacy Migration Command Examples

"Theories are always very thin and insubstantial, experience only is tangible."

– Hosea Ballou

This section uses the following conventions:

Commands are in `courier`; names are in *`italicized courier`*.

The legacy disk group is named *`datadg`*

There is exactly one volume in the disk group, named *`datavol`*

The volume and filesystem are 20GB; the file system is mounted at mount point *`/data`*

The legacy LUN is 20GB and is named *`EMC_CLARiiON1_1`*
This is an enclosure-based name. If using classic naming, such as `hdisk#` on AIX, `c#t#d#` or `disk#` on HP-UX, `sd{letter}#` on Linux, or `c#t#d#` on Solaris, substitute the classic name instead. If you do not know your legacy disk's name, see the vxprint command examples below.

The new, thin LUN is 40GB and is named *`3PARDATA0_2`*

`EMC_CLARiiON1_1` and *`3PARDATA0_2`* are both the media name and the access name for the LUNs in these examples. The access name is the 'system name'; the media name is the name used by VxVM. VxVM lets users separate these names, which can be useful. In a real migration, you may find a legacy LUN whose media name differs from its access name. If so, substitute the media name for *`EMC_CLARiiON1_1`* everywhere *`EMC_CLARiiON1_1`* is used below[11]. To determine the media name of your LUNs, run:

```
# vxprint -g datadg –dt
```

Output 1: the legacy LUN's media name and access name differ. The first name is the media name, the second is the access name.

```
DM NAME         DEVICE         TYPE   PRIVLEN   PUBLEN       STATE
dm 3PARDATA0_2  3PARDATA0_2     auto   65536    83820544       -
dm datadg01     EMC_CLARiiON1_1 auto   65536    41877504       -
```

Output 2: the legacy LUN has the same media name and access name.

```
DM NAME             DEVICE          TYPE    PRIVLEN   PUBLEN    STATE
dm 3PARDATA0_2      3PARDATA0_2      auto    65536    83820544    -
dm EMC_CLARiiON1_1  EMC_CLARiiON1_1 auto    65536    41877504    -
```

Full path names are not given in the examples. From Storage Foundation version 4.0 on, all Veritas commands are in /opt/VRTS/bin except for vxresize, which was overlooked. For vxresize, use the full path name, /etc/vx/bin/vxresize. This will be fixed in a future release.

Setting up Dynamic Multipathing (DMP) to the new LUN is not described. For information on these steps, see the Further Reading section.

[11] This is unnecessary for the thin LUN, because the media name is set to the access name by the "vxdg adddisk" step. Similarly, in the real world you control of the thin LUN's media name, but the legacy LUN's media name is dealt to you.

LUN creation and provisioning steps are not shown, as these vary substantially by vendor. Similarly, the operating system-specific step of labeling/formatting the LUN is not shown.

These examples may not apply to Storage Foundation for Windows. A future paper on Windows is anticipated.

If the system on which the legacy migration is being performed is a Cluster File System (CFS) or a Storage Foundation for Oracle RAC cluster, the commands below must be executed on the Cluster Volume Manager master node. Replace the mount and umount commands (needed only in the multi-volume file system method) with the cfsmount and cfsumount commands; for more information, consult the cfsmount(1m) man page.

The file system flag for mount and for fsadm varies. AIX uses -V, HP-UX -F, Linux -t, and Solaris -F. All other commands are identical unless noted. The examples use Linux -t.

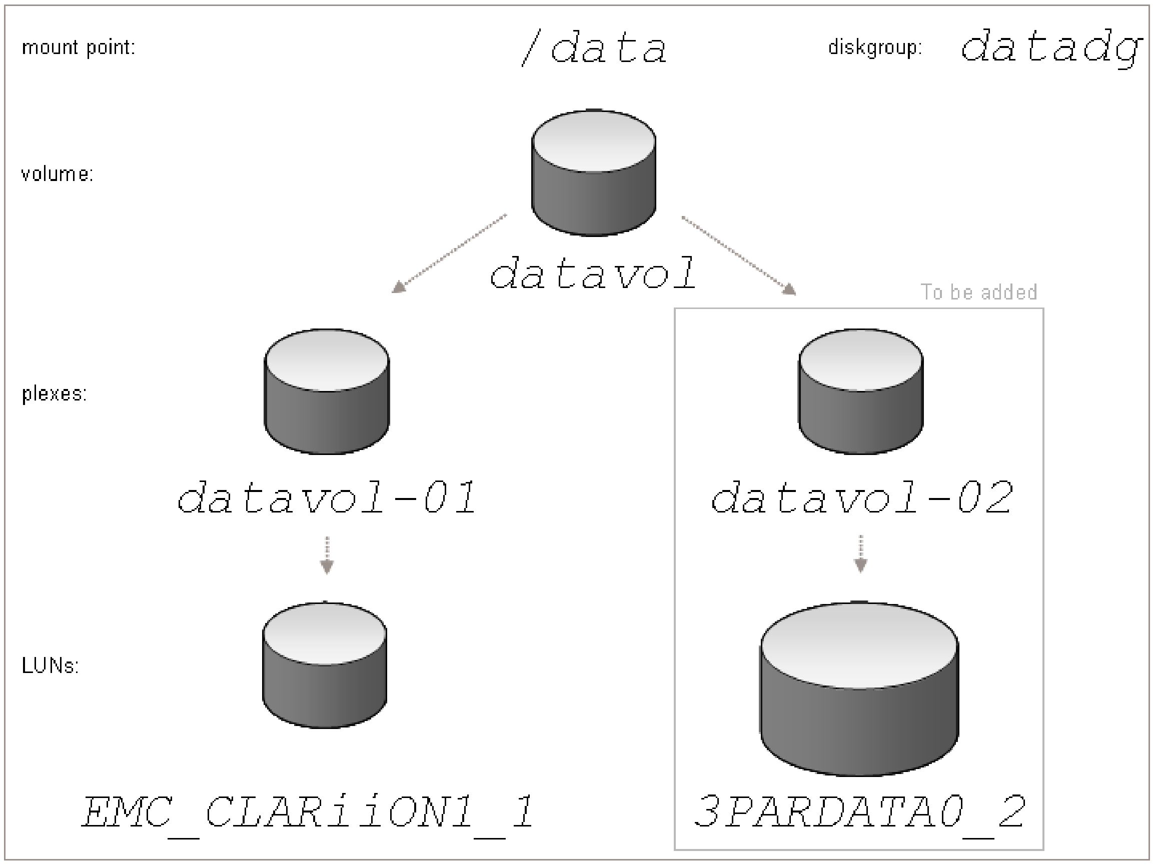


Figure 7: Object Relationships and Names.

Legacy Migration with Veritas SmartMove™

This approach requires VxFS and VxVM 5.0MP3 or higher.

1. Turn on SmartMove™. Edit the file /etc/default/vxsf so that the variable usefssmartmove=yes

   This tunable is system-wide and persistent, so it only needs to be set once per server. Note that 5.0MP3 sets usefssmartmove=no by default. In a future release, usefssmartmove will default to yes and step 1 (this step) will no longer be necessary.

2. (Optional). Defragment the file system. See Appendix for a discussion of pros and cons.

   2-i. (Option 1: run until complete)

   ```
   # fsadm –t vxfs –ed /data
   ```

   2-ii. (Option 2: cap run time at, in this example, 1 hour = 3600 seconds)[12]

   ```
   # fsadm –t vxfs –ed –T3600 /data
   ```

3. Add the new, thin LUN to the existing disk group.

   3a. `# vxdisksetup –i 3PARDATA0_2`

   3b. `# vxdisk init 3PARDATA0_2`

   3c. `# vxdg -g datadg adddisk 3PARDATA0_2`

4. Add the new, thin LUN as a new plex to the volume. Three options are given. The first is the default. The second finishes sooner, but with more I/O impact. The third has less impact, but takes longer.

NOTE: The file system must be mounted to get the benefits of SmartMove™.

   4a-i. (Option 1: default)

   ```
   # vxassist –g datadg mirror datavol 3PARDATA0_2
   ```

   4a-ii. (Option 2: fast completion)[13]

   ```
   # vxassist –b –oiosize=1m –t thinmig -g datadg mirror datavol
   3PARDATA0_2
   # vxtask monitor thinmig
   ```

   4a-iii. (Option 3: minimum impact)[14]

   ```
   # vxassist -oslow –g datadg mirror datavol 3PARDATA0_2
   ```

5. (Optional) Test the new LUN's performance before removing the old LUN[15].

   5a. Optional. Determine which plex corresponds to the thin LUN:

   ```
   # vxprint -g datadg
   ```

(output on next page)

[12] Note that the time option flag is -t on AIX, HP-UX, and Solaris, and -T on Linux (because Linux specifies -t for the file system flag).
[13] The -b option runs the command in the background, but also multi-threads I/O, synchronizing multiple mirror regions simultaneously. The -t option and the vxtask command simplify monitoring (since the command is in the background); if you run multiple migrations simultaneously, assign a different name to each. If using Storage Foundation 5.0 or higher, you do not need the "–oiosize=1m" option, as this is the default. Internal tests suggest little benefit for values >1m.
[14] Impact can be further reduced by specifying a number greater than 250, e.g., -oslow=500
[15] This optional test makes VxVM read only from the new LUN (assuming it is available). Writes still go to both LUNs to ensure both mirrors contain identical data.

```
TY NAME                 ASSOC          KSTATE   LENGTH   PLOFFS   STATE    TUTIL0 PUTIL0
dg datadg               datadg          -       -           -        -        -      -

dm 3PARDATA0_2          3PARDATA0_2     -      83886080    -        -        -      -
dm EMC_CLARiiON1_1      EMC_CLARiiON1_1 -      41943040    -        -        -      -

v  datavol             fsgen          ENABLED  41943040    -       ACTIVE   -       -
pl datavol-01          datavol        ENABLED  41943040    -       ACTIVE   -       -
sd EMC_CLARiiON1_1-01  datavol-01     ENABLED  41943040    0        -       -       -
pl datavol-02          datavol        ENABLED  41943040    -       ACTIVE   -       -
sd 3PARDATA0_2-01      datavol-02     ENABLED  41943040    0        -       -       -
```

5b. Optional. The thin LUN corresponds to plex *datavol-02*. Direct all reads to come from that LUN:

```
# vxvol -g datadg rdpol prefer datavol datavol-02
```

6. Remove the original LUN. Note that ! is an escape on some shells; the example given avoids bash escape.

   6a. `# vxassist -g datadg remove mirror datavol \!EMC_CLARiiON1_1`

   6b. `# vxdg -g datadg rmdisk EMC_CLARiiON1_1`

   6c. `# vxdisk rm EMC_CLARiiON1_1`

7. Grow the file system and volume to use all of the larger thin LUN:

   7a. `# vxresize -g datadg -x datavol 40g 3PARDATA0_2`

Legacy Migration with Reclaim-before-move

This approach requires VxFS and VxVM[16].

1. Add the new, thin LUN to the existing disk group. The operating system-specific step of labeling/formatting the LUN for OS recognition is not shown[17].

   1a. `# vxdisksetup –i 3PARDATA0_2 privlen=32m`

   1a. `# vxdisk init 3PARDATA0_2`

   1b. `# vxdg -g datadg adddisk 3PARDATA0_2`

2. Reclaim the storage. Note: the file system must be mounted for this operation[18]. Note that example reclaims to 10 GB; this is just illustrative. Choose the size so that the file

[16] All Storage Foundation versions since at least 1995 officially support this method. In some cases, the amount of storage that can be reclaimed will be limited. If the file system is unable to shrink by the amount requested, reduce the request by about 50% and retry. If successful, attempt to shrink again by 25% of the attempt; if not successful, repeat the prior step. Repeat. Limitations: (1) file systems converted from another vendor to VxFS prior to VxFS version 5.0 will be limited in their ability to shrink. Also, if an mmapped file is present in VxFS 4.0 or earlier, the amount of storage that can be reclaimed may be limited – the workaround is to close all applications during the shrink operation, but first attempt the operation with the application open.

[17] The privlen=32m argument in step 1a is not strictly necessary, but it is a recommended best practice. This is not needed with Storage Foundation version 5.0 or higher, since 32m is the default.

system will be close to full, but leave some space available in case the application needs new storage during the migration. For example, if the file system is 100gb total and 30% full as show by a df -k, 40g may be an appropriate new size.

2a. `# vxresize -g` *`datadg`* `-s` *`datavol`* `10g`

3. Add the new, thin LUN as a new plex to the volume. For throttling options, see the previous section, Legacy Migration with Veritas SmartMove™ step 4.

NOTE: The file system must be mounted to get the benefits of SmartMove™.

3a. `# vxassist –g` *`datadg`* `mirror` *`datavol 3PARDATA0_2`*

4. (Optional) Test the new LUN's performance before removing the old LUN[19]. See the previous section, Legacy Migration with Veritas SmartMove™ step 5.

5. Remove the original LUN as a plex. Note that ! is an escape on some shells; the example given avoids bash escape.

5a. `# vxassist -g` *`datadg`* `remove mirror` *`datavol`* `\`*`!EMC_CLARiiON1_1`*

6. Grow the file system and volume to use all of the larger thin LUN.

6a. `# vxresize -g` *`datadg`* `-x` *`datavol`* `40g` *`3PARDATA0_2`*

7. Remove the legacy LUN from the diskgroup:

7a. `# vxdg -g` *`datadg`* `rmdisk` *`EMC_CLARiiON1_1`*

7b. `# vxdisk rm` *`EMC_CLARiiON1_1`*

Legacy Migration with Storage Foundation Manager (SFM)

SFM is available to customers on a current Storage Foundation support contract at no additional charge. SFM works with Storage Foundation version 4.0 and higher, and can be downloaded at: www.symantec.com/sfm

SFM can simplify migrations, particularly when there are tens, hundreds, or thousands of volumes or LUNs to migrate; when the migration spans multiple servers or multiple arrays; or when administrators are not familiar with Veritas command line utilities

When given the source array and target array, SFM identifies all affected Storage Foundation volumes, and suggests a mapping from legacy LUNs to new LUNs. The administrator can modify the mapping if desired; the planned migration can either be run or be saved to execute later, during a change window. Screenshots from the migration process are below.

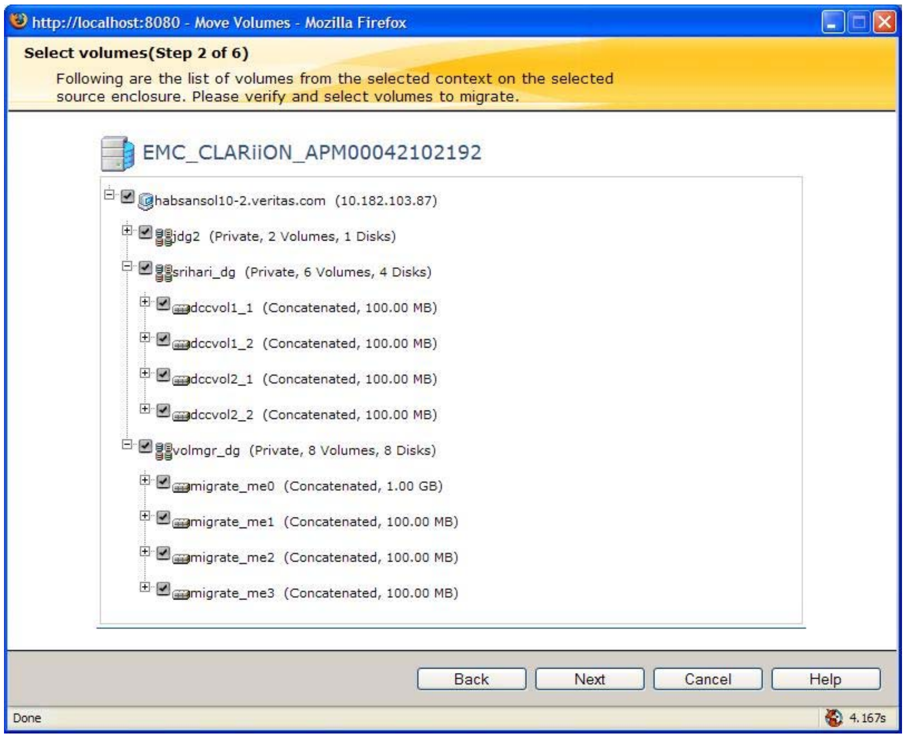
The steps in the SFM "volume migration add-on" correspond to steps 2-5 as described above in the SmartMove™ section; SmartMove™ step 6 can be performed via SFM as well, though not as part of the automated volume migration process. Similarly, when performing a "reclaim before move" migration with SFM, all steps can be performed with SFM but the order of operations

---

[18] For VxFS/VxVM versions prior to 4.1, do not attempt to run this operation with an unmounted file system. Version 4.1 and higher will block the operation if the file system is unmounted.
[19] This optional test makes VxVM read only from the new LUN (assuming it is available). Writes still go to both LUNs to ensure both mirrors contain identical data.

differs[20]. Step 2 is performed by the administrator in SFM. Steps 1, 3-5, and 7 are performed by SFM's volume migration process. Step 6 is then performed by the administrator in SFM.

Given SFM's intuitive GUI, the shots below showcase the process, but do not fully document it.



Select volumes(Step 2 of 6)

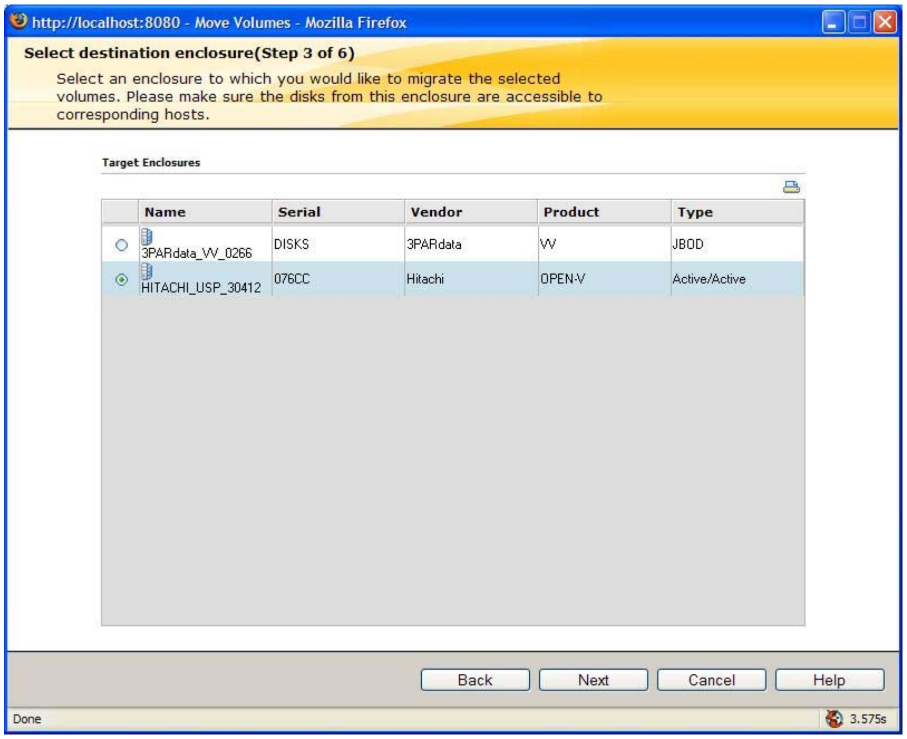
Following are the list of volumes from the selected context on the selected source enclosure. Please verify and select volumes to migrate.

EMC_CLARiiON_APM00042102192

```
habsansol10-2.veritas.com (10.182.103.87)
  jdg2 (Private, 2 Volumes, 1 Disks)
  srihari_dg (Private, 6 Volumes, 4 Disks)
    dccvol1_1 (Concatenated, 100.00 MB)
    dccvol1_2 (Concatenated, 100.00 MB)
    dccvol2_1 (Concatenated, 100.00 MB)
    dccvol2_2 (Concatenated, 100.00 MB)
  volmgr_dg (Private, 8 Volumes, 8 Disks)
    migrate_me0 (Concatenated, 1.00 GB)
    migrate_me1 (Concatenated, 100.00 MB)
    migrate_me2 (Concatenated, 100.00 MB)
    migrate_me3 (Concatenated, 100.00 MB)
```

Figure 8: SFM Volume Migration. Select Volumes from legacy (source) array. By default, all are migrated.

[20] Either order of operations works. The CLI order minimizes the time the file system spends at the temporary smaller size; the SFM order uses SFM's bulk processing of multiple LUNs and volumes.



http://localhost:8080 - Move Volumes - Mozilla Firefox

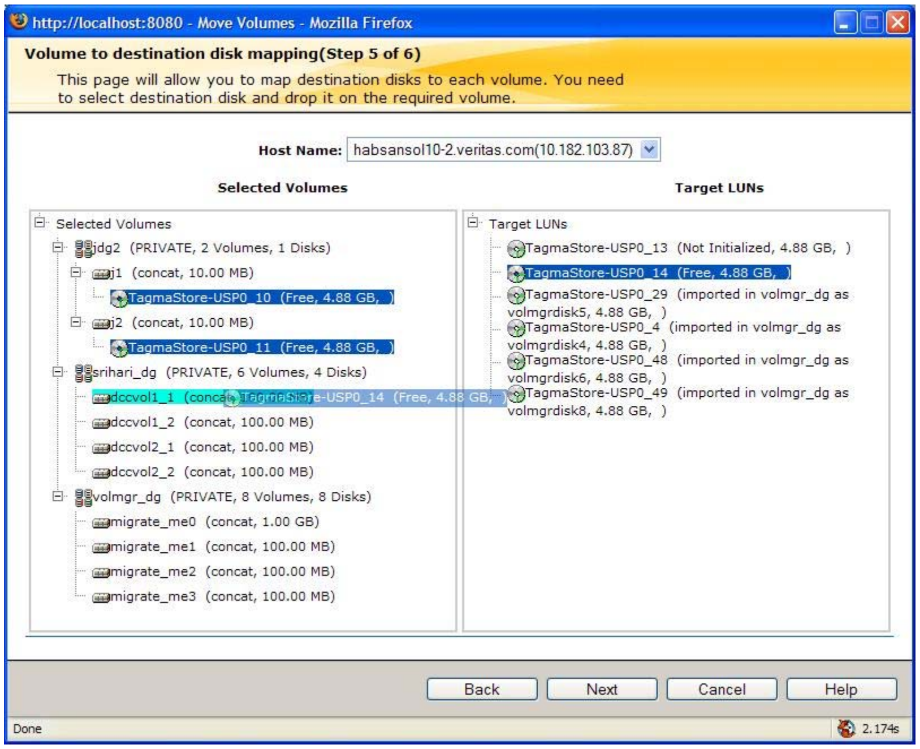
Select destination enclosure(Step 3 of 6)

Select an enclosure to which you would like to migrate the selected volumes. Please make sure the disks from this enclosure are accessible to corresponding hosts.

Target Enclosures

	Name	Serial	Vendor	Product	Type
○	3PARdata_VV_0266	DISKS	3PARdata	VV	JBOD
●	HITACHI_USP_30412	076CC	Hitachi	OPEN-V	Active/Active

Figure 9: SFM Volume Migration. Select the target (new) array.



http://localhost:8080 - Move Volumes - Mozilla Firefox

Volume to destination disk mapping(Step 5 of 6)

This page will allow you to map destination disks to each volume. You need to select destination disk and drop it on the required volume.

Host Name: habsansol10-2.veritas.com(10.182.103.87)

Selected Volumes

```
Selected Volumes
  jdg2  (PRIVATE, 2 Volumes, 1 Disks)
    j1  (concat, 10.00 MB)
      TagmaStore-USP0_10  (Free, 4.88 GB, )
    j2  (concat, 10.00 MB)
      TagmaStore-USP0_11  (Free, 4.88 GB, )
  srihari_dg  (PRIVATE, 6 Volumes, 4 Disks)
    dccvol1_1  (concat  TagmaStore-USP0_14  (Free, 4.88 GB, )
    dccvol1_2  (concat, 100.00 MB)
    dccvol2_1  (concat, 100.00 MB)
    dccvol2_2  (concat, 100.00 MB)
  volmgr_dg  (PRIVATE, 8 Volumes, 8 Disks)
    migrate_me0  (concat, 1.00 GB)
    migrate_me1  (concat, 100.00 MB)
    migrate_me2  (concat, 100.00 MB)
    migrate_me3  (concat, 100.00 MB)
```

Target LUNs

```
Target LUNs
  TagmaStore-USP0_13  (Not Initialized, 4.88 GB, )
  TagmaStore-USP0_14  (Free, 4.88 GB, )
  TagmaStore-USP0_29  (imported in volmgr_dg as volmgrdisk5, 4.88 GB, )
  TagmaStore-USP0_4  (imported in volmgr_dg as volmgrdisk4, 4.88 GB, )
  TagmaStore-USP0_48  (imported in volmgr_dg as volmgrdisk6, 4.88 GB, )
  TagmaStore-USP0_49  (imported in volmgr_dg as volmgrdisk8, 4.88 GB, )
```

Figure 10: SFM Volume Migration. Mapping target LUNs to volumes. Drag and Drop interface.

Legacy Migration with Multi-volume File Systems

Command examples are provided in the Appendix.

Use Thin File Systems

VxFS is an extent-based file system, which uses thin provisioning more efficiently than traditional block-based file systems. Block-based file systems typically use more metadata to store the same data. Block-based file systems also tend to allocate their metadata, such as inodes, when the file system is created. By contrast, VxFS creates metadata only as needed. As a result, VxFS uses thin provisioning more efficiently.

Hewlett-Packard's StorageWorks Division recently published a guide to using thin provisioning. It has several good operational recommendations for thin provisioning; it agrees with this paper's recommendations on defragmentation, for example. As part of this work, the paper analyzed whether different file systems are "capacity efficient". The results, in Table 9 on page 15, show VxFS receiving high marks[21,22].

http://h71028.www7.hp.com/ERC/downloads/4AA1-3937ENW.pdf

Deletions and Shuffling Reduce Thin Provisioning's Benefit

As users or applications delete files or records, whether at the file system or at the database layer, logical space is freed, but physical storage in the thin LUN stays allocated. Consider a thin LUN that claims to be 10GB, but whose actual physical storage is 2GB. That 2GB represents the "high water mark" for data used. Imagine this system had two files, each 1GB in size as in Figure 11a (below). A user deletes one file, reducing the actual logical usage at the file system layer at 1GB. The thin LUN remains 2GB, however (Figure 11b).

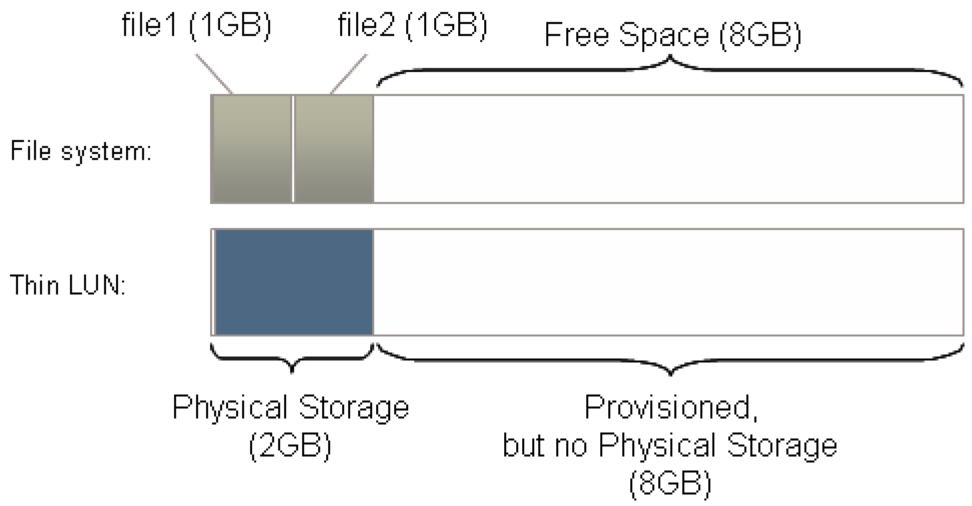


Figure 11a. The system before the deletion (above).

[21] VxFS was not tested on Linux in this comparison, however VxFS' format and allocation strategy are common across platforms, so the authors expect the same results on Linux as on every other platform.
[22] HP StorageWorks XP24000 Thin Provisioning Software implementation white paper. Accessed December 4, 2007.

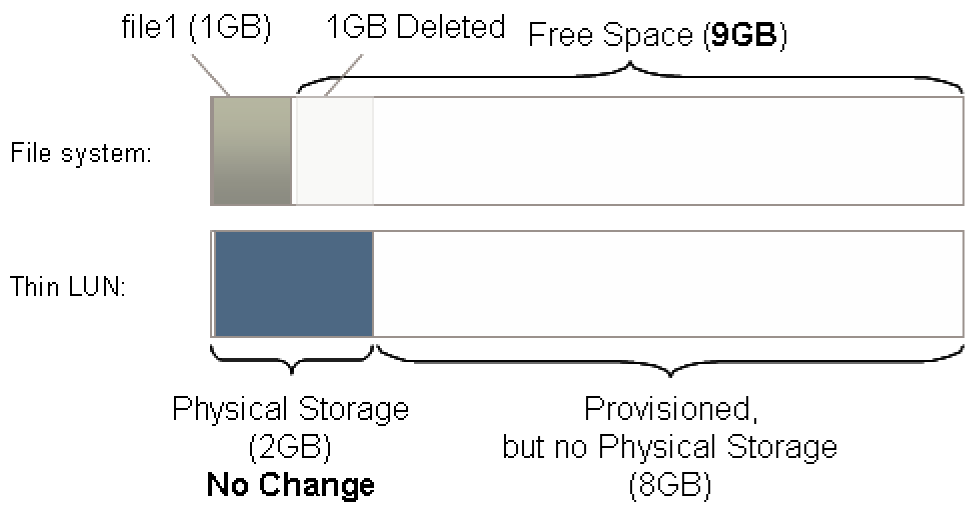

file1 (1GB)
1GB Deleted
Free Space (**9GB**)
File system:
Thin LUN:
Physical Storage (2GB) **No Change**
Provisioned, but no Physical Storage (8GB)

Figure 11b. Deleting files does not reclaim thin provisioned storage.

A similar situation can arise through a different mechanism, shuffling. A file system or an application may rearrange its data so as to improve performance, typically by moving related data that was far away closer together. Also called defragmenting, Microsoft's NTFS and Veritas Storage Foundation provide this service[23], as do some databases. However, defragmenting defeats thin provisioning's relatively simple logic, which only tracks the first write to a given block. At this time, Symantec recommends not defragmenting thin provisioned systems[24].

A third, situation is when data moves within a file or a file system before shrinking[25]. The amount of thin provisioned storage wasted will vary, but the basic operating rule is simple: once a Veritas file system has been migrated to thin LUN, don't shrink it[26].

VxFS Algorithms Reduce the Deletion Problem

When any file system receives a request for more storage – when an application or user creates a new file or appends data to an existing file – that file system uses its internal algorithms to choose the free blocks it will assign to that request. The Veritas File System's algorithm allocation is thin-provisioning friendly, and specifically deals with the deleted files issue.

For scalability reasons beyond this paper's scope, VxFS divides the file system into regions called allocation units (AUs). When allocating space for a new file or for an appending write, VxFS uses space from any partially used AUs units <u>before it uses space from empty AUs</u>.

[23] Vendors who give users' transparency into their products' level of fragmentation, and the means to fix it are attacked as "prone to fragmentation" by other vendors who refuse to disclose data on their own products' level of fragmentation. This is an example of the folk wisdom "no good deed goes unpunished."
[24] Commercial database applications rarely cause significant file system fragmentation, although they may fragment at the database level. Fragmentation is seen in some UNIX mailservers, where deletions are common, a large number of files are involved, and files' sizes are widely distributed.
[25] Data is typically moved from the back of the file (or file system) to the front, since the shrink happens at the back, or tail, of the file (or file system).
[26] This is not an issue of capability or safety, but is a best practice relating to keeping storage efficiency high. If there is a need to shrink the Veritas file system – the authors cannot think of an example other than migrating off of a big thin LUN back to a smaller traditional LUN – it is fully supported and safe to do so, the only cost will be some thin provisioning efficiency.

This works well with thin provisioning. To see why, consider an example.

This example file system has 10 allocation units, each 256MB in size[27], and is built on a 3PAR thinly provisioned LUN. One AU is completely filled up with several files. Another AU is half full and half free. The other eight are empty; they hold no files and all blocks are free. For simplicity, assume the thin array also allocates storage in 256MB chunks. Since two AUs are fully or partially used, the array has allocated two chunks. As applications request more storage from VxFS, VxFS always chooses free space from the partially filled AU to meet those requests. That means the array need not allocate a third chunk. Now an application deletes a file; it happens to reside on the partially full AU. Next, an application requests more storage from VxFS. Again, VxFS chooses storage from the partially filled AU. Until VxFS completely fills the second AU, the array will not need to allocate more chunks.

SmartMove™ Helps in Some Cases

Symantec is working with thin provisioning vendors to explore methods to reclaim deleted files and clean up after defragmentation, thus giving data centers even better storage utilization from thin provisioning. In the meantime, data center staff can use Veritas SmartMove™ as a work around should they find themselves in a situation where the file system is mostly empty, but the thin LUN is largely or completely backed by physical storage. The process is similar to the legacy migration described earlier. This time, migration is from a thin LUN to a thinner LUN.

Recall our thin LUN – it represents itself as 10GB, it is backed by 2GB, but only 1GB of data is actually stored on it. A new 10GB thin LUN can be created on the new array that is backed not by the size of the current array's actual storage (2GB), but by the size of the current file system (1GB). By using Veritas SmartMove™ for the migration, only 1GB of data will be copied to the new LUN, reclaiming 1GB of storage as part of an already scheduled process.

Scheduled migrations are convenient times for such reclamations, because the staff time and planning for the migration provides essentially a free ride for the reclamation. However, one need not wait for an array migration to perform reclamation. A new, smaller thin LUN can be provisioned from the current array, and Veritas Storage Foundation will copy only the used data from the old to the new LUN.

Doing such 'inter-array migration' on an entire data center would cost more in staff time that it would save in storage. Again, this method is presented only as a stop-gap for operators who need an immediate option, and who cannot wait until Symantec and the thin provisioning vendors deliver more automated solutions[28].

Volume Mirroring with Thin LUNs

An important consideration when designing a storage architecture is volume mirroring via a host-based volume manager. Host-based mirroring protects against array-level failures[29]. If the arrays are located in different failure domains – as small as adjacent rooms with separate cooling systems, or as large as geographic regions served by different electrical utilities – the system's resilience improves. Many enterprises use host-based mirroring when uptime is truly critical.

[27] VxFS AUs are 256MB for an 8k file system block size, 128MB for a 4k block size, 64MB for a 2k block size, and 32MB for a 1k (default) block size. The last AU in a file system is sized to the space available.
[28] A third option is backup the original LUN and restore it to a newly created thin LUN. Like the SmartMove™ approach, a new LUN is needed. Unlike the SmartMove™ approach, it requires downtime.
[29] Host-level mirroring holds a key advantage over array-to-array mirroring: host-level mirroring can lose either array without causing application downtime. By contrast, array-to-array mirroring sets up a source-target dependency chain; if the source array is lost, the application must be restarted using the target array.

Host-based volume managers must behave differently when mirroring thin provisioned arrays as compared to traditional arrays. One configuration and two failure scenarios must be handled differently: volume creation, mirror synchronization after transient array failure, and mirror synchronization after server failure. Each scenario is explained and discussed below.

First, volume creation. By default, when mirroring two LUNs, Storage Foundation ensures data consistency by synchronizing the contents of the LUNs before the volume can be used[30]. With thin provisioning, this synchronization is unnecessary. When fulfilling a read request of a block that has never been written to, a thin provisioning LUN returns a standard value, typically zero. This means that two thin provisioned LUNs are already effectively synchronized. Furthermore, the synchronization step defeats the purpose of thin provisioning, since it triggers unnecessary grow events on the LUN. Fortunately, Storage Foundation provides a method to create mirrored volumes that are not synchronized before use.

Second, array failures. After an array failure – typically transient and due to power or network connectivity – Storage Foundation normally copies the surviving array's LUN to the failed array's LUN once the failed array recovers. This again defeats the purpose of thin provisioning. There are two ways to avoid this issue. For versions prior to 5.0MP3, a full block copy can be avoided using Storage Foundation's FlashSnap technology. FlashSnap tracks changed blocks to speed up mirror synchronization. In thin provisioning environments this has the added benefit of avoiding a full copy triggered by an array failure. With 5.0MP3 and higher, the authors recommend enabling both SmartMove™ and FlashSnap. Both SmartMove™ and FlashSnap limit the copying of unused blocks when resyncing LUNs. But they capture slightly different usage data at different granularity levels[31]. Using both features keeps LUNs as thin as possible post-resync.

Third, server failures. When a crashed server boots, Storage Foundation resyncs mirrored volumes to ensure consistency between them, in case an in-flight write has been applied to one half of a mirror but not the other. Storage Foundation does this by copying the contents of one mirror to the other. But, if the mirrored LUNs are thin provisioned, this recovery process defeats the thin provisioning logic. SmartMove™ cannot help in this case, since the file system cannot be brought online until the volume is consistent. Helpfully, Storage Foundation can log the locations of recent writes, limiting the resync to only the areas that had active writes. This log is called a Dirty Region Log, or DRL[32].

In summary, Storage Foundation handles these scenarios that arise when mirroring thin provisioned LUNs. Command Examples showing how to mirror thin provisioned LUNs with Storage Foundation are given in the Appendix.

---

[30] Why must two new LUNs be in sync? In Symantec's 15+ years of volume manager experience, we have encountered applications that read blocks they have not written to. Because VxVM splits reads across mirrors to increase IO bandwidth, successive reads of the same block could return different values if the lower level LUNs had not been synchronized. VxVM's default behavior protects both against this scenario, and against the resulting consequences to the application.

[31] Specific examples of the rational for using both FlashSnap and SmartMove are given in the Appendix.

[32] When using Oracle 9i or higher, Storage Foundation's ODM feature can improve performance. This API between Oracle and Storage Foundation boosts performance of write-intensive databases. Storage Foundation 5.0 further improves performance by automatically adjusting DRL behavior when Oracle sends IO through ODM.

Efficiency Measurements

Quantifying Thin Storage Efficiency

The efficiency of various file systems while running on top of thin provisioned storage can be measured and the results compared. The authors devised, conducted, and recorded a series of initial tests and test results from host-based storage tools. Three tools were studied: Veritas Storage Foundation; Linux' ext3 and LVM; and Sun Microsystem's ZFS.

Although we tested fewer tools than did HP in their recent paper[33], our findings reinforce their findings on Table 9, page 15, namely that Veritas Storage Foundation makes efficient use of thinly provisioned storage. This paper extends that research by providing test measurements.

Three test series were conducted, each with five or more test runs to ensure repeatability of the results. The first test series used the UNIX dd utility to create files and write data to them. dd's widespread availability makes this a popular testing tool among UNIX administrators. The second and third tests used Symantec's vxbench utility. This utility gives additional control of IO, which can better match actual application IO. The second test series used vxbench to send asynchronous IO (where write flushing can be delayed to improve performance), and the third test series used vxbench to send synchronous IO (where writes must be flushed to disk before returning).

Each test run consisted of creating an empty file system, then alternating creations and deletion of between one and ten files, each between 1GB and 10GB in size. As a final step, all files were deleted. After each step (e.g., mkfs, file creation, file deletion) the amount of physical storage used in the thin provisioning array was recorded.

The results in all three test series were broadly similar. When empty, ext3/LVM used substantially more physical storage than did Storage Foundation or ZFS (whose results were comparable). This difference is likely due to the initial, static allocation of inodes across different regions of the LUN by ext3. Once a substantial amount of data was created, Storage Foundation typically used 12% less physical storage than did ZFS. Storage Foundation used between 4% and 26% less physical storage than did ext3/LVM, depending on the test run. None of the three tools tested reclaimed any storage upon final deletion of all files.

Charts

In each chart, the dashed line represents the amount of user data that the test generated. It starts at zero when the file system is created (the metadata of the file system is not considered user data). It increases as files are created, and decreases as files are deleted. The results of the three vendor tools are represented by solid lines. Essentially, the lower (closer) a solid line lies to the dashed line, the better the performance of that tool. Note that because thinly provisioned storage uses an "allocate forever after first write", when all files are deleted, the amount of user data drops to zero, but the amount of physical storage the array has allocated does not change.

[33] http://h71028.www7.hp.com/ERC/downloads/4AA1-3937ENW.pdf Accessed December 4, 2007.

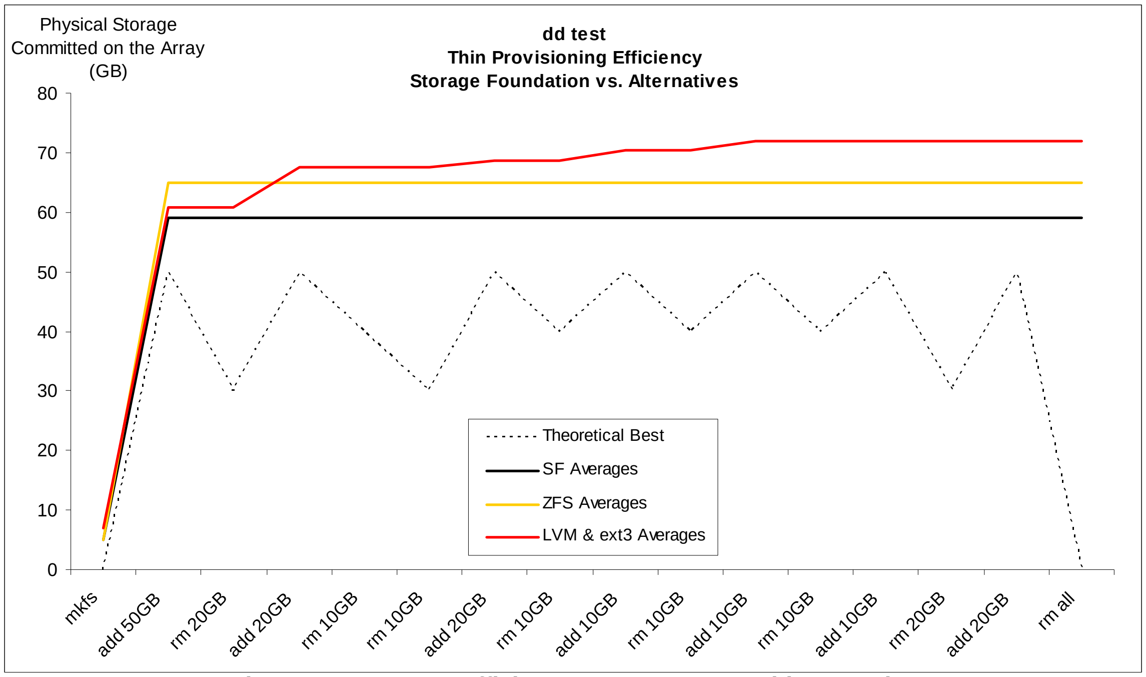


Figure 12a. Storage Efficiency measurements, dd test series

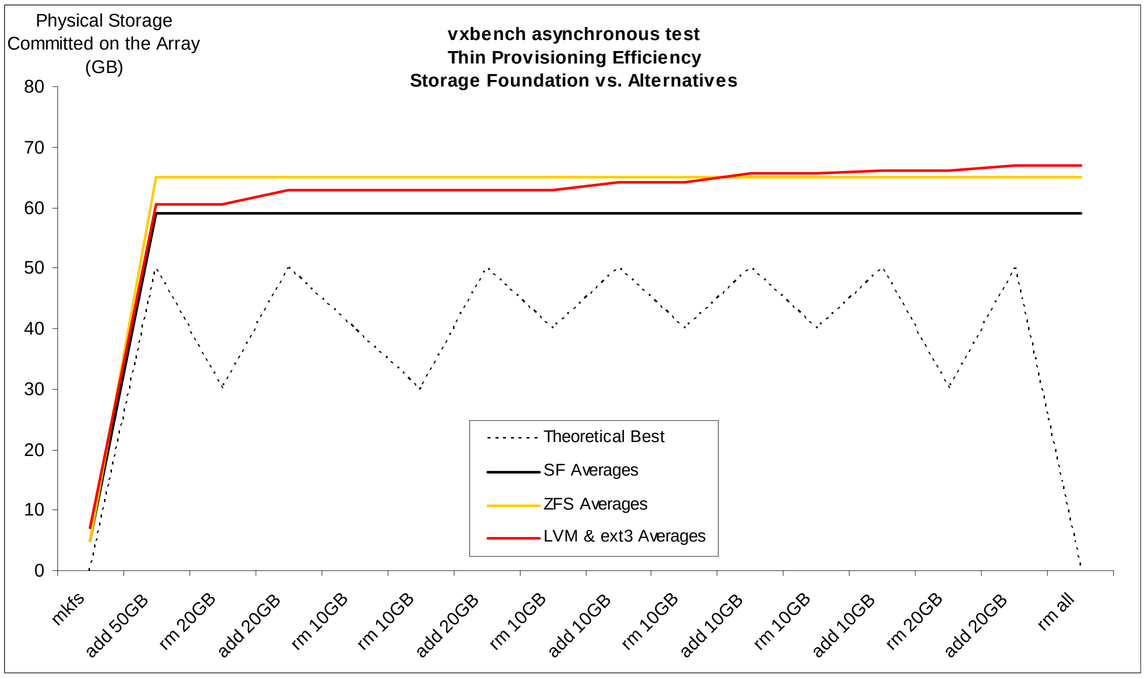


Figure 12b. Storage Efficiency measurements, vxbench asynchronous test series

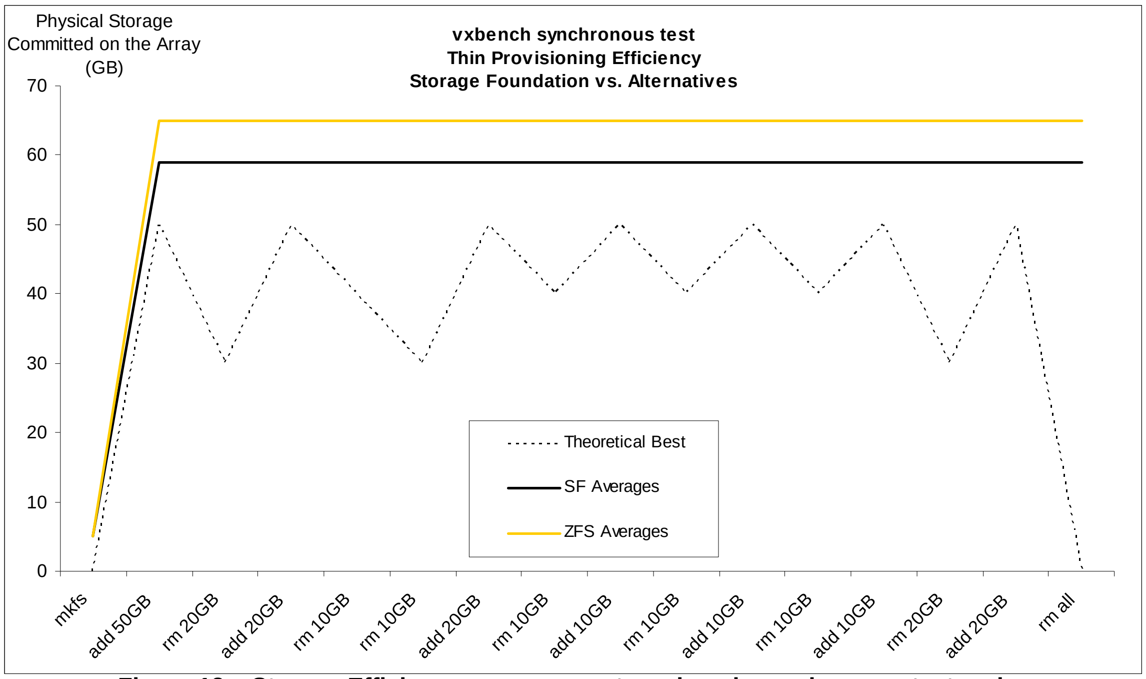


Figure 12c. Storage Efficiency measurements, vxbench synchronous test series

Note: Figure 12c lacks ext3/LVM measurements because ext3/LVM were unable to successfully complete the benchmark. ext3/LVM returned write errors that prevented the test from completing. These errors were only seen during the synchronous write tests, and only with ext3/LVM. The errors were consistent and reproducible. They suggest an ext3 or LVM bug, but further investigations were not made, so this cannot be stated conclusively.

Test Configuration

Storage:

3PAR InServ, firmware version 2.2.1
10 x 10GB Thin Provisioned LUNs

Server & Software:

For Storage Foundation and ZFS tests:
Sun V240, 2 1.5 GHz UltraSparc-IIIi CPUs, 2GB RAM
Storage Foundation 5.0 MP1
ZFS as shipped with Solaris 10 Update 4

For ext3/LVM tests:
Sun v20z, 2 2.2 GHz AMD Opteron CPUs, 8 GB RAM
Ext3/LVM as shipped with RedHat Enterprise Linux 4 Update 4

Commands used for file creation:

Test Series 1:
```
# dd if=/dev/zero of=<filename> bs=1048576k count=$1
```

Test Series 2:
```
# vxbench -m -w write -i iosize=1g,iocount=<filesize> <filename>
```

Test Series 3:
```
# vxbench -m -w write -o sync -i iosize=1g,iocount=<filesize>
<filename>
```

Storage Tiering with Thin Provisioning

Storage Tiering Overview

Since thin provisioning has different costs and different behaviors than traditional storage, it can be treated as a separate storage tier. Because tiered storage is a term used to mean many different things, it is worth quickly recapping the different approaches to implementing tiered storage. Static tiering involves choosing a storage tier for an application when the application is put into production; the tier is rarely (if ever) changed. Infrastructure tiering uses one tier for an application's main data, and a secondary tier for that same application's "infrastructure" storage, such as snapshots, standby site replicas, or disk backups. Dynamic Tiering gives a single application both a primary and a secondary storage tier, and then migrates data between the tiers based on the data's business value. Typical indicators of business value are access frequency, the user, or the file type. Thin provisioning can be used in any of these three tiering scenarios – static, infrastructure, or dynamic.

Dynamic Storage Tiering (DST)

Storage Foundation's Dynamic Storage Tiering (DST) provides the policies and policy engine to move data among tiers. The administrators creates policies based on business requirements, which are translated to attributes such as file name and file type, directory, user, and usage frequency. Uniquely, DST moves data transparently to applications and users, by keeping the logical location of files and directories exactly the same. More information on DST can be found in the Further Reading section.

DST with a Thin Provisioning Tier

DST can help protect against thin provisioning error scenarios. Because thin provisioning hands out logical storage before physical storage, it can run out of physical storage while still having unfulfilled storage commitments to hosts and applications – much like a bounced check. Adding physical storage to the array resolves the situation, but that can take time[34]. DST provides a method to stay up and running – and, indeed, even to add more storage – until physical storage is added to the thin provisioning array. Here's how.

VxFS can use multiple volumes. This overcomes the traditional limitation of mapping a single volume to a file system. For example, take a single thin provisioned volume that runs out of physical storage, shown in Figure 13a. To keep the application up and running, a second volume is added to the file system[35]. This second volume comes from a second array, since the original array has no more physical storage. DST policies are then set to direct all new writes to go to the second volume, while reads and updates to the original data still use the first volume[36]. In effect, Storage Foundation can restrict I/O activity on the original LUN to only those blocks actually backed by physical storage (see Figure 13b). In this way, applications are not confused by the visible, but unavailable, storage on the original volume.

Once more physical storage is added to the first volume, the second volume can be 'ejected' – VxFS will automatically move all data from the second volume to the original volume,

[34] Some array vendors improve response time through phone-home functionality, although that works less well in secure facilities and in geographies with limited telecommunications and logistics maturity.
[35] For online operation, the first VxVM volume must be in an object called a volume set. This can be done at creation time or later. We recommend putting VxVM volumes in volume sets for this added flexibility.
[36] More precisely, overwrites of existing files – common in databases – go to the original volume. New files and appending writes to existing files go to the new volume.

and will then remove the second volume. The system is then back to the original configuration (Figure 13c). The Appendix provides command examples and policies to use in this scenario.

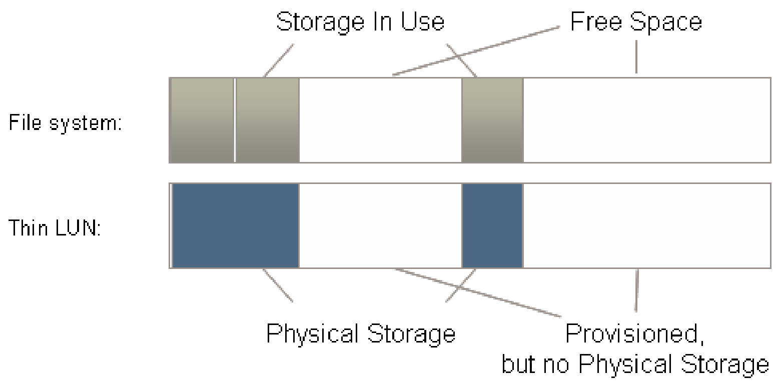


Figure 13a: A Thinly Provisioned LUN runs out of physical storage.

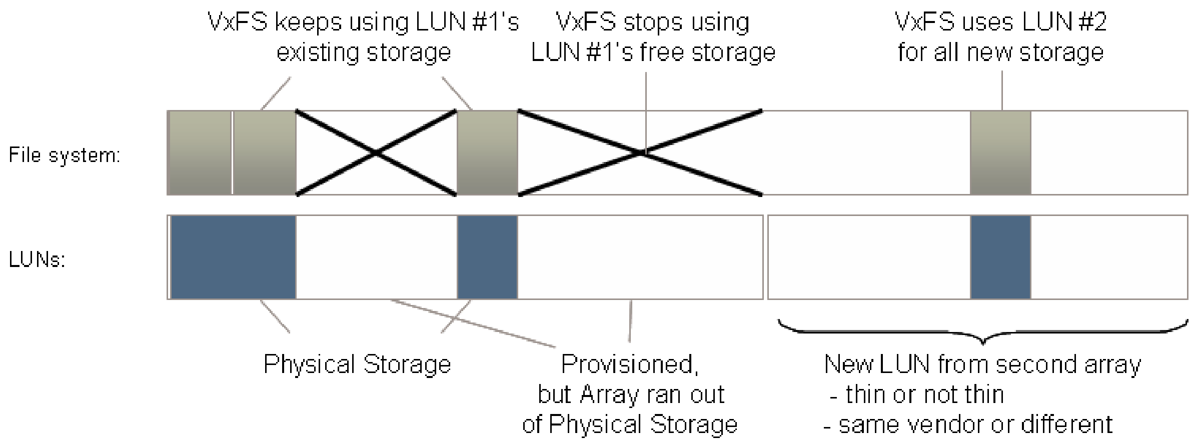

VxFS keeps using LUN #1's existing storage
VxFS stops using LUN #1's free storage
VxFS uses LUN #2 for all new storage
File system:
LUNs:
Physical Storage
Provisioned, but Array ran out of Physical Storage
New LUN from second array
- thin or not thin
- same vendor or different

Figure 13b: A second LUN from another array is added; VxFS policies are set to keep using data from the original LUN, but to place any new data on the second LUN.

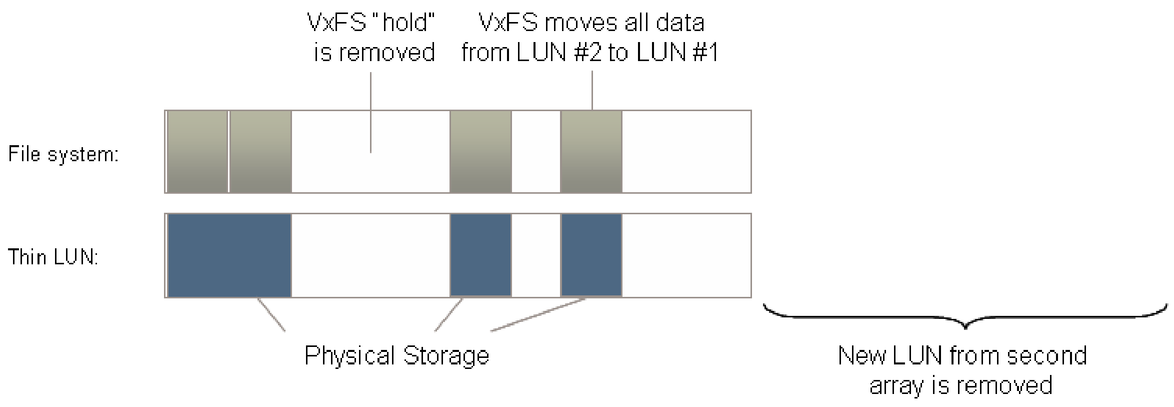

VxFS "hold" is removed
VxFS moves all data from LUN #2 to LUN #1
File system:
Thin LUN:
Physical Storage
New LUN from second array is removed

Figure 13c: Physical storage is added to the original array. VxFS moves all data from the second LUN to the original LUN, then removes the second LUN.

DST may also help specific applications that are a poor fit for thin provisioning. Consider applications that frequently delete files soon after they are created, such as an email system with spam folders, or applications that, in addition to their 'normal' files, rapidly create then delete many lock files. These applications fit less well with thin provisioning, because, as described in the previous section deleted files use thin provisioning less efficiently.

Dynamic Storage Tiering can address both of these situations. A two-tier system can be built, where both tiers come from the same storage array. Tier One is a traditional, fully provisioned LUN. Tier Two is a much larger thinly provisioned LUN. The DST policy instructs the file system to create new files on Tier One (traditional). Once the 'deletion window'[37] is complete, the remaining files are moved to Tier Two, the thin tier. Figure 14 illustrates the concept.

Again, moving files from Tier One to Tier Two is undetectable by users or applications. The file remains in the same directory it was originally created in, and neither stub files nor symbolic links are used. The file system stays online before, during, and after the move and a file can even be read and written while it is moving between tiers.

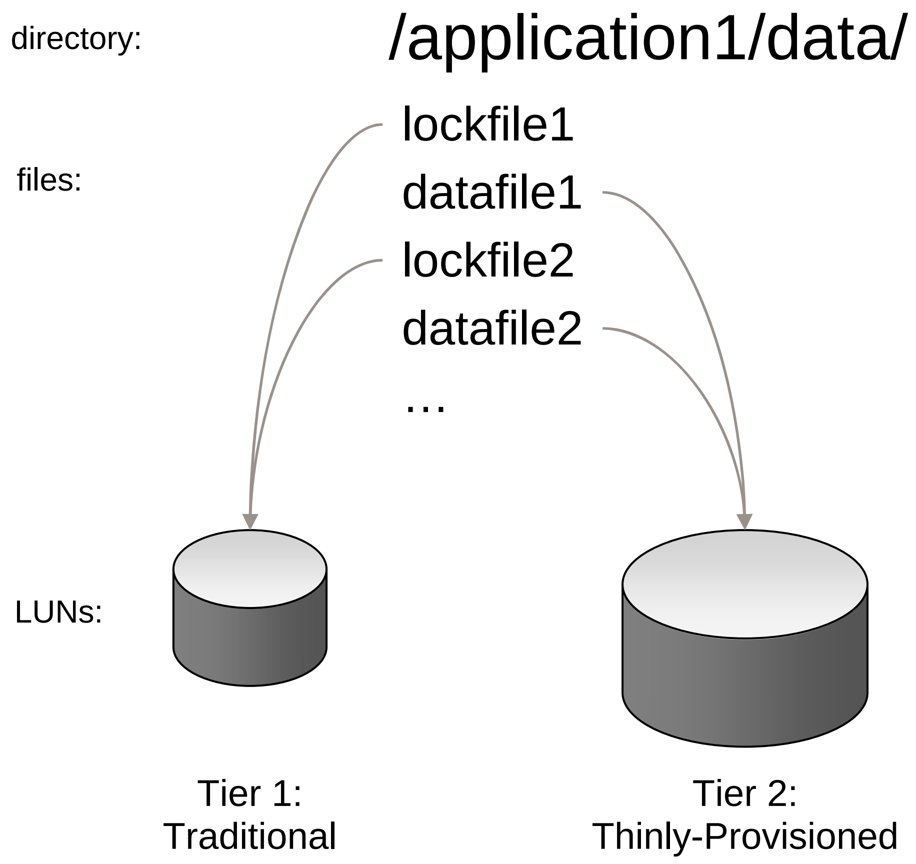


Figure 14: Thin LUN as a Storage Tier. Files stay in the same directory, but their underlying storage can move between Tier 1 and Tier 2.

[37] The 'deletion window' describes the concept that a file is deleted near the start of its life, if at all. A lock file, for example, is deleted as soon as the transaction it protects or flags is complete – typically a few seconds. For an email server, spam will typically be deleted within a few days of receipt, for example by the user checking their mail or by policies set by the ISP for suspected spam. Once the deletion window is closed – a few seconds after creation for lock files, a week or two after creation for spam files – surviving files can move to thin provisioning with low risk of deletion.

Applications that do not cleanly handle I/O errors[38], may also find thin provisioning challenging, particularly for files holding their metadata. To provide an added measure of safety, all metadata files can be stored on a traditional LUN, while regular files can be stored on a thin LUN. In this case, the DST policy would specify that all files with specific attributes – e.g., with names ending in .ctl or .log – must be created on and must remain on the traditional LUN.

DST differs from array-based approaches to storage tiering in three important ways. First, DST gives administrators rich control over migrations using not just aging and usage statistics, but more user-, application- and business-friendly attributes like a file's name, extension, size, owner, group, and directory. Second, array tools cannot list which files' blocks have moved from one LUN to another. They can report which blocks on a LUN have been moved, but they cannot map those blocks to files. In contrast, DST can list the tier on which a file sits, and it can list all files on a given tier. DST keeps a history log of the file name, the starting tier, and ending tier, and the time of migration to give enterprise administrators complete visibility.

Third, when evaluating a tiered storage offering, always understand what happens to Tier 1 data if Tier 2 suffers a failure. If a section of a file is frequently accessed, and another section infrequently accessed – in a database file, for example – how will the application behave if portions of the file are missing due to a Tier 2 failure[39]? How does the vendor's offering guarantee that file system metadata is kept Tier 1? Dig deep enough, and you will find that array-based tiering mechanisms cannot distinguish between data which is required for the system to work – control files, metadata, and so on – and data which is not[40]. This limitation reduces system availability. DST, in contrast, can force database control files, metadata files, its own metadata, and other critical components to stay on Tier 1 storage.

In summary, DST can help avoid or recover from thin provisioning error scenarios. It can filter out transient files from hitting the long-term (and thin) storage, thus improving thin provisioning utilization. It can also provide added security for applications which do not cleanly deal with out of space errors. This architecture will not be appropriate for all, or even most applications, but it lets these applications use thin provisioning, which they might otherwise find to be difficult or of minimal value.

[38] A thin provisioning array typically returns an I/O Error when it cannot allocate physical storage to a thinly provisioned LUN.
[39] UFS, the UNIX file system, panics the server if certain metadata is not available, for example.
[40] In most systems, the blocks on Tier 1 will be available even if Tier 2 fails. However, this is not a sufficient safeguard, because the array vendors' tiering logic cannot determine whether it is migrating critical data to Tier 2.

Conclusion

"You can never be too rich or too thin"

– Truman Capote

Modern datacenters are constantly asked to do more with less. Thin provisioning promises to let them do so. It is a mistake to stop there, however. As we have discussed, thin provisioning can be 'thinner' – more efficient with higher utilization – when Storage Foundation is used to complement thin provisioning hardware.

When selecting which host tools to use with thin provisioning, architecture and operations team members should pay close attention to how such tools impact physical storage usage both for migration and for daily operations. Choosing the wrong technology can reduce thin provisioning's benefit. Furthermore, claims that a tool is 'free' should be carefully evaluated. A Total Cost of Ownership (TCO) approach may show that the ongoing savings from an alternative solution can quickly and easily outweigh any initial acquisition costs.

Storage Foundation delivers all the capabilities needed to reduce wasted disk space before the move to thin provisioning, supports the migration itself, and handles operations – creation, deletion, etc - in a thin provisioning environment with the most efficient approach available today. Integrated storage tiering completes the solution, delivering protection against over subscription of physical storage.

A superior approach is only as good as the range of its applicability. Storage Foundation has the most comprehensive hardware compatibility list available and supports all current thin provision architectures including 3PAR, Hitachi, Hewlett-Packard XP, Network Appliance, and Sun. From an operating system perspective, Storage Foundation is available on AIX, HP-UX, Linux, Solaris, and Windows. This broad coverage of servers and of thin-provisioning storage allows customers to standardize on a common tool set across platforms, reducing complexity and increasing standardization.

Selecting the right host-based storage tools is crucial to the success of any thin provisioning project. Storage Foundation's wide installed base, unmatched support matrix and history of leadership in supporting thin provisioning allow customers to fully realize thin provisioning's benefits.

Further Reading

"I like a thin book because it will steady a table."
- Mark Twain

Thin Provisioning:
Hall, Nigel. "Regaining Control of Storage Growth with Thin Provisioning." Hitachi Data Systems. June 2007.
http://www.hds.com/assets/pdf/wp-regaining-control-of-storage-growth-thin-provisioning.pdf

Hough, Geoffrey and Sandeep Singh. "3PAR Thin Provisioning." 3PAR Incorporated. June 2003.
http://www.3par.com/documents/3PAR-tp-wp-01.2.pdf

Oracle 10G's Reclamation Capability via File Shrink:
Floss, Kimberley. "Shrink Segments Online and in Real Time." *Oracle Magazine*, May-June 2005.
http://www.oracle.com/technology/oramag/oracle/05-may/o35tuning.html

Puri, Aradhana. "More New Features for OCPs." *Oracle Magazine*, January-February 2005.
http://www.orafaq.com/forum/t/22393/0/

Dynamic Storage Tiering:
Karche, Ganesh, Murthy Mamidi and Paul Massiglia, *Using Dynamic Storage Tiering*. 2006.
http://eval.symantec.com/mktginfo/enterprise/yellowbooks/dynamic_storage_tiering_03_2006.en-us.pdf

Wu, Joel C., Bo Hong and Scott A. Brandt. "Ensuring Performance in Activity-Based File Relocation." University of Santa Cruz. 21 April 2007. http://www.ssrc.ucsc.edu/Papers/wu-ipccc07.pdf

"HP and Symantec: Enabling ILM solutions with Dynamic Storage Tiering white paper." Hewlett-Packard Corporation. July 2007.
http://h71028.www7.hp.com/ERC/downloads/4AA1-2792ENW.pdf

Storage Foundation Manager:
Storage Foundation Manager webpage with documentation, FAQ, and free download. Symantec. Accessed 18 October 2007.
http://www.symantec.com/sfm

Converting to Storage Foundation:
"Veritas Storage Foundation™ Migration Guide: AIX." Symantec. 2006.
http://ftp.support.veritas.com/pub/support/products/Foundation_Suite/284308.pdf

"Veritas™ Volume Manager Migration Guide: HP-UX." Symantec. 2006.
http://ftp.support.veritas.com/pub/support/products/VolumeManager_UNIX/283743.pdf

"Veritas™ Volume Manager Migration Guide: Linux." Veritas. 2004.
http://ftp.support.veritas.com/pub/support/products/Foundation_Suite/269978.pdf

"Veritas™ vxvmconvert for Solaris User Guide." Veritas. December, 2002.
http://support.veritas.com/docs/255198

Inventory Planning; Correlated and Uncorrelated Demand:
Cachon, Gerard and Christian Terwiesch. An Introduction to Operations Management. Boston: McGraw-Hill/Irwin, 2005. In particular, see Chapters 11 and 12.

Dynamic Multipathing:
"Dynamic Multipathing: Optimizing Availability and Performance in Multi-Vendor Environments." Symantec. May 2007. Pages 16-23 discuss setup steps.
http://eval.symantec.com/mktginfo/enterprise/white_papers/ent-whitepaper_vsf_5.0_dynamic_multi-pathing_05-2007.en-us.pdf

Appendix

“Our revels now are ended. These our actors, as I foretold you, were all spirits, and are melted… into thin air.”

– Shakespeare

This section uses the following conventions:

Commands are in `courier`; names are in *`italicized courier`*.

The legacy disk group is named *`datadg`*

There is exactly one volume in the disk group, named *`datavol`*

The volume and filesystem are 20GB; the file system is mounted at mount point *`/data`*

The legacy LUN is 20GB and is named *`EMC_CLARiiON1_1`*
This is an enclosure-based name. If using classic naming, such as `hdisk#` on AIX, `c#t#d#` or `disk#` on HP-UX, `sd{letter}#` on Linux, or `c#t#d#` on Solaris, substitute the classic name instead. If you do not know your legacy disk’s name, see the vxprint command examples below.

The new, thin LUN is 40GB and is named *`3PARDATA0_2`*

`EMC_CLARiiON1_1` and *`3PARDATA0_2`* are <u>both the media name and the access name</u> for the LUNs in these examples. The access name is the ‘system name’; the media name is the name used by VxVM. VxVM lets users separate these names, which can be useful. In a real migration, you may find a legacy LUN whose media name differs from its access name. If so, substitute the media name for *`EMC_CLARiiON1_1`* everywhere *`EMC_CLARiiON1_1`* is used below[41]. To determine the media name of your LUNs, run:

```
# vxprint -g datadg –dt
```

Output 1: the legacy LUN’s media name and access name differ. The first name is the media name, the second is the access name.

```
DM NAME         DEVICE         TYPE   PRIVLEN   PUBLEN       STATE
dm 3PARDATA0_2  3PARDATA0_2     auto   65536    83820544        -
dm datadg01     EMC_CLARiiON1_1 auto   65536    41877504        -
```

Output 2: the legacy LUN has the same media name and access name.

```
DM NAME             DEVICE          TYPE    PRIVLEN   PUBLEN    STATE
dm 3PARDATA0_2      3PARDATA0_2      auto    65536    83820544    -
dm EMC_CLARiiON1_1  EMC_CLARiiON1_1 auto    65536    41877504    -
```

These examples may not apply to Storage Foundation for Windows. A future paper on Windows is anticipated.

[41] This is unnecessary for the thin LUN, because the media name is set to the access name by the “vxdg adddisk” step. Similarly, in the real world you control of the thin LUN’s media name, but the legacy LUN’s media name is dealt to you.

Full path names are not given in the examples. From Storage Foundation version 4.0 on, all Veritas commands are in /opt/VRTS/bin except for vxresize, which was overlooked. For vxresize, use the full path name, /etc/vx/bin/vxresize. This will be fixed in a future release.

Setting up Dynamic Multipathing (DMP) to the new LUN is not described. For information on these steps, see the Further Reading section.

LUN creation and provisioning steps are not shown, as these vary substantially by vendor. Similarly, the operating system-specific step of labeling/formatting the LUN is not shown.

If the system on which the legacy migration is being performed is a Cluster File System (CFS) or a Storage Foundation for Oracle RAC cluster, the commands below must be executed on the Cluster Volume Manager master node. Replace the mount and umount commands (needed only in the multi-volume file system method) with the cfsmount and cfsumount commands; for more information, consult the cfsmount(1m) man page.

The file system flag for mount and for fsadm varies. AIX uses -V, HP-UX -F, Linux -t, and Solaris -F. The rest of the commands are identical. The examples use Linux -t.

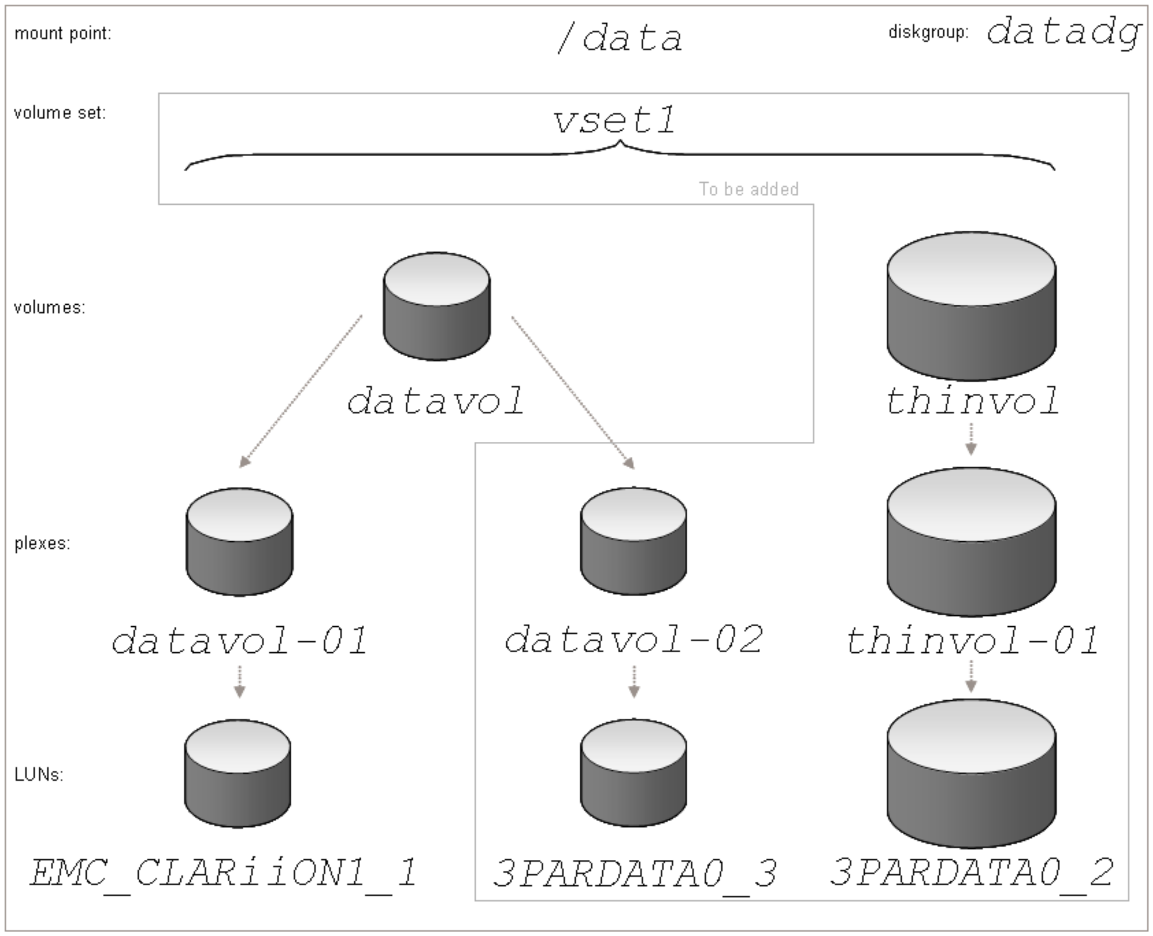


Figure 15: Object Relationships and Names.

Command Example: Legacy Migration with Multi-volume File Systems

This approach requires VxFS and VxVM 4.1 or higher, and a Storage Foundation Enterprise license[42].

1. Convert the existing VxFS file system to a multi-volume file system. The file system is mounted at */data*, the volume is *datavol*, which is in diskgroup *datadg*.

   1a. Determine the version of the volume's diskgroup:
   ```
   # vxdg list datadg | grep version: | awk '{ print $2 }'
   ```
   105
   1b. If the version is less than 110, upgrade the diskgroup:
   ```
   # vxdg upgrade datadg
   ```
   1c. Determine the disk layout version of the file system:
   ```
   # vxupgrade /data
   ```
   Version 5
   1d. If the disk layout is version is less than 6, upgrade to Version 6 or 7[43]:
   ```
   # vxupgrade -n 7 /data
   ```
   1e. Unmount the file system:
   ```
   # umount /data
   ```
   1f. Convert the volume into a volume set. The volume set name *vset1* is used as an example:
   ```
   # vxvset -g datadg make vset1 datavol
   ```
   1g. Edit the file [/etc/filesystems on AIX; /etc/fstab on HP-UX and on Linux; /etc/vfstab on Solaris] to replace the volume device name, *datavol*, with the volume set name, *vset1*.
   1h. Mount the file system.
   ```
   # mount -t vxfs /dev/vx/dsk/datadg/vset1 /data
   ```
   1i. Add the new, thin LUN to the existing disk group. The operating system-specific step of labeling/formatting the LUN for OS recognition is not shown[44].
   ```
   # vxdisksetup –i 3PARDATA0_2 privlen=32m
   # vxdisk init 3PARDATA0_2
   # vxdg -g datadg adddisk 3PARDATA0_2
   ```
   1j. Create and add the volume comprised of the thin LUN to the volume set:
   ```
   # vxassist -g datadg make thinvol 40g 3PARDATA0_2
   # vxvset -g datadg addvol vset1 thinvol
   ```
   1k. Add the new volume to the file system:
   ```
   # fsvoladm add /data thinvol 40g
   ```

2. Create policies to move all files and metadata to the new, thin LUN[45], then verify.

   ```
   # fsapadm define /data datapolicy thinvol datavol
   # fsapadm define /data metadatapolicy datavol
   # fsapadm assignfs /data datapolicy metadatapolicy
   # fsapadm list /data
   ```

[42] Storage Foundation (SF) Cluster File System, SF Cluster File System HA, SF for Oracle RAC, SF for Oracle Enterprise, SF for DB2 Enterprise, and SF for Sybase Enterprise all provide the “Enterprise” level license needed.
[43] Version 7 is only available with VxFS version 5.0 or higher.
[44] The privlen=32m argument in step 1a is not strictly necessary, but it is a recommended best practice. This is not needed with Storage Foundation version 5.0 or higher, since 32MB is the default.
[45] These policies also tell the file system to create any new files on the thin volume, unless it is full, in which case new files are placed on the original volume.

3. Move the existing files to the new, thin LUN.
   Storage Foundation version 5.0 command:
   ```
   # find /data –exec fsapadm enforecefile –f strict '{}' \;
   ```
   Storage Foundation version 4.1 command:
   ```
   # find /data –exec fsapadm enforecefile '{}' \;
   ```

Optional. Verify the files moved. If you have many files, you may wish to verify a subset. The fsmap command is only available with Storage Foundation version 5.0 and higher.
```
# find /data –exec fsmap '{}' \;
```

4. Shrink the original volume to the desired size.
   ```
   # vxresize -g datadg -s datavol 50m
   ```
5. Add a new fully-provisioned, non-thin LUN from the new array to the disk group.
   ```
   # vxdisk init 3PARDATA0_3
   # vxdg -g datadg adddisk 3PARDATA0_3
   ```
6. Mirror that LUN to the original volume.
   ```
   # vxassist –g datadg mirror datavol 3PARDATA0_3
   ```
7. Break off the original mirror and remove the LUN.
   ```
   # vxassist -g datadg remove mirror datavol \!EMC_CLARiiON1_1
   # vxdg -g datadg rmdisk EMC_CLARiiON1_1
   # vxdisk rm EMC_CLARiiON1_1
   ```

Command Example: Mirror Thin Provisioned LUNs

This section uses the same conventions and object names given in the section "Legacy Migration Command Examples", except that a second thin provisioned LUN of 40GB is used, named *3PARDATA1_1*

The step of applying an operating system label to the LUNs is not shown.

Mirroring Thin Provisioned LUNs requires FlashSnap, which is used for recovery if one LUN becomes unavailable. FlashSnap requires a Storage Foundation Enterprise license[46]. The example given combines the FlashSnap Data Change Object (DCO) with the DRL for easier configuration; this requires version 4.0 or higher[47]. The DCO and DRL use a portion of each of the two LUNs for metadata to track changed blocks[48].

[46] Alternately, a FlashSnap Option license may also be used; this is an older license that is no longer sold, but readers may encounter it on an older server

[47] Mirroring thin provisioning LUNs can also be accomplished using Veritas Volume Manager version 3.2 or 3.5, but the DCO and the DRL must each be created separately. For such a setup on these older versions, consult the product documentation.

[48] Since some array vendors license thin provisioning as an additional cost, you may wish to create this object, called a data change object (DCO), on separate small LUNs that are not thinly provisioned. In some cases, this will also improve performance. Further, if you have many LUNs to mirror, you can share the

Note: If you are using Storage Foundation version 5.0MP3 or higher, perform Step 1; if not, skip to Step 2.

1. Turn on SmartMove™. Edit the file /etc/default/vxsf so that the variable usefssmartmove=yes

   This tunable is system-wide and persistent, so it only needs to be set once per server. Note that 5.0MP3 sets usefssmartmove=no by default. In a future release, usefssmartmove will default to yes and step 1 (this step) will no longer be necessary. If you are using Storage Foundation 5.0MP3 in either a failover cluster or in a CVM (active-active) cluster, turn on SmartMove™ on all cluster nodes.

2. Verify you have a license key to use FlashSnap.

   2a. `# vxlicrep -i | grep FMR`

   You should see one or more lines like:

   ```
   FMR_DGSJ                                  = Enabled
   FMR_DGSJ#VERITAS Volume Manager           = Enabled
   ```

   Note that this step does not distinguish between temporary keys and permanent keys – rerun the command without grep to verify that your keys are permanent.

3. Create the diskgroup using the two LUNs.

   3a. `# vxdisksetup –i 3PARDATA0_2`

   3b. `# vxdisksetup –i 3PARDATA1_1`

   3c. `# vxdg init datadg 3PARDATA0_2 3PARDATA1_1`

4. Optional. If you used a legacy diskgroup instead of creating a new diskgroup in step 3, verify your diskgroup version and upgrade if required.

   4a. `# vxdg list datadg | grep version`

   If the version is not 110 or higher, upgrade as shown in step 4b.

   4b. `# vxdg upgrade datadg`

5. Create the mirrored volume with FlashSnap and Dirty Region Logging (DRL) enabled, and without synchronizing the mirrors. The DCO, used by FlashSnap and by DRL, will use a small amount of space, so specify a volume size a few MB smaller than the LUN size.

   5a.
   ```
   # vxassist –g datadg make datavol 40958m nmirrors=2 init=active
   logtype=dco dcoversion=20 drl=on fastresync=on 3PARDATA0_2
   3PARDATA1_1
   ```

Using FlashSnap and SmartMove™ together when mirroring

In general, when mirroring two thin provisioned LUNs, it better to turn on both FlashSnap and SmartMove™. After a transient array failure – due to a power outage at the SAN fabric level, for

DCO among all volumes in a diskgroup. These configurations are beyond the scope of this document; consult the Storage Foundation FlashSnap documentation to learn more.

example – the two plexes must be resynchronized so that each contains the same data as the other (a plex is a Storage Foundation term analogous to a mirror, in this scenario each LUN maps to one plex). To avoid writing data that is already in sync and triggering storage allocations on the thin array, FlashSnap resyncs only the changed blocks and SmartMove™ resyncs only the occupied blocks. But why use both?

One reason is granularity. FlashSnap tracks regions of a volume, not individual blocks. If a single block is written to, FlashSnap will flag the whole region around that block for resync (usually 64k). If the rest of that 64k region contained no files, SmartMove™ complements by just resyncing the single block. That avoids triggering allocations that could occur if the whole region was resync'd.

A second reason is deletions. If a file is modified, then subsequently deleted (or kept but rewritten elsewhere, an operation certain applications frequently use), FlashSnap will flag the whole region for resync. If SmartMove™ is also used, none of the region need be resync'd, so the array would not allocate any new storage.

The above cases illustrate how SmartMove™ complements FlashSnap. When does FlashSnap complement SmartMove™?

The first case is when the file system is not available. VxFS must be online and mounted for SmartMove™ to provide information from the file system to the volume manager. That will usually, but not always be the case. If the file system is not yet mounted, as happens in some recovery sequences, FlashSnap avoids a full resync and the resulting allocations.

The second case is when an application has large, pre-allocated files that are mostly empty. SmartMove™ conservatively treats those as completely full. In this case, FlashSnap is more granular than SmartMove™, since it tracks changed regions since the last resync operation. FlashSnap does not know, nor does it need to know, to which files or to which portions of files those regions correspond. It simply flags the region for resync. Regions of a pre-allocated file that were modified or that were newly written will be flagged for resync. Empty regions that did not receive any writes will not be flagged, and those empty regions will not be resynced during recovery, so the array will not need to unnecessarily allocate storage to those empty regions.

Command Example: Restricting I/O to Physically Allocated Storage

This example follows the scenario described in "DST with a Thin Provisioning Tier". To recap:

A single, thin provisioned volume has run out of physical storage. To keep the application up and running, add a second volume to the file system from a second array (since the original array has no more physical storage). Set Dynamic Storage Tiering (DST) policies to restrict I/O activity to only those blocks on the LUN that actually backed by physical storage. Then, once new storage is added to the originally array, move any new data from the second volume back to the first, and then remove the second volume.

This example assumes that the file system was not built on a volume set. If it was, skip steps 1a through 1h. Conventions and notes given at the beginning of the Appendix apply here as well, i.e., operating system differences, path locations, etc. To illustrate SF's hardware-neutral nature, the second LUN is from a Hitachi array. The second volume's name is 'catchvol' as its purpose, conceptually though not literally, is to catch "overflow" I/O (extending writes or new files) which the first volume cannot hold. The object relationships are illustrated below.

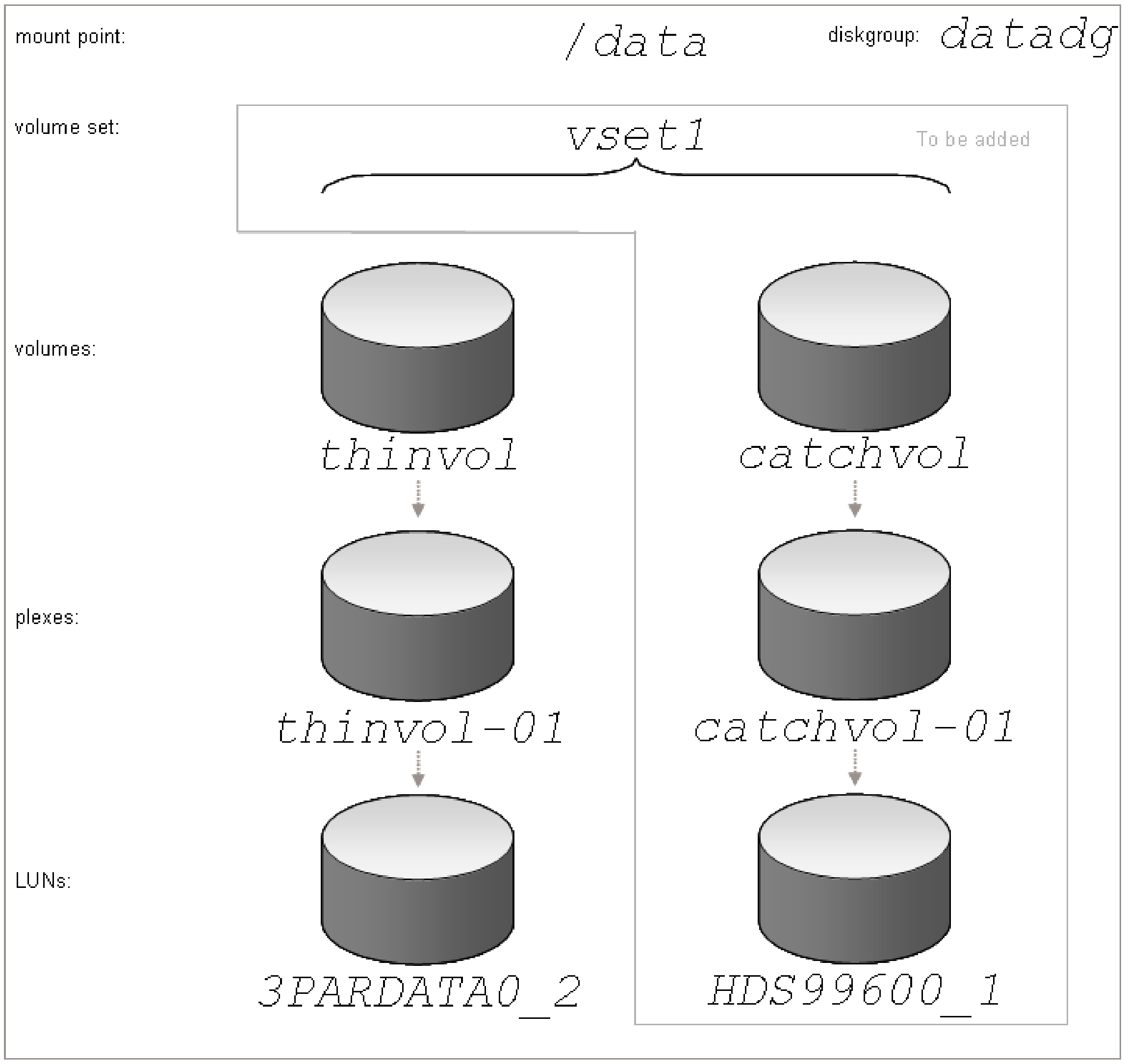


Figure 16: Object Relationships and Names.

VxFS and VxVM 4.1 or higher, and a Storage Foundation Enterprise license[49], is required.

1. Convert the existing VxFS file system to a multi-volume file system. The file system is mounted at /data, the volume is thinvol, which is in diskgroup datadg.

   1a. Determine the version of the volume's diskgroup:
   ```
   # vxdg list datadg | grep version: | awk '{ print $2 }'
   ```
   105
   1b. If the version is less than 110, upgrade the diskgroup:
   ```
   # vxdg upgrade datadg
   ```
   1c. Determine the disk layout version of the file system:
   ```
   # vxupgrade /data
   ```
   Version 5
   1d. If the disk layout is version is less than 6, upgrade to Version 6 or 7[50]:

[49] Storage Foundation (SF) Cluster File System, SF Cluster File System HA, SF for Oracle RAC, SF for Oracle Enterprise, SF for DB2 Enterprise, and SF for Sybase Enterprise all provide the "Enterprise" level license needed.

```
# vxupgrade -n 7 /data
```
1e. Unmount the file system:
```
# umount /data
```
1f. Convert the volume into a volume set. The volume set name *vset1* is used as an example:
```
# vxvset -g datadg make vset1 thinvol
```
1g. Edit the file [/etc/filesystems on AIX; /etc/fstab on HP-UX and on Linux; /etc/vfstab on Solaris] to replace the volume device name, *thinvol*, with the volume set name, *vset1*.
1h. Mount the file system.
```
# mount -t vxfs /dev/vx/dsk/datadg/vset1 /data
```
1i. Add the second LUN to the existing disk group. The operating system-specific step of labeling/formatting the LUN for OS recognition is not shown[51].
```
# vxdisksetup –i HDS99600_1 privlen=32m
# vxdisk init HDS99600_1
# vxdg -g datadg adddisk HDS99600_1
```
1j. Create and add a second volume comprised of the second LUN to the volume set:
```
# vxassist -g datadg make catchvol 40g HDS99600_1
# vxvset -g datadg addvol vset1 catchvol
```
1k. Add the new volume to the file system:
```
# fsvoladm add –f metadataok /data catchvol 40g
```

2. Set the policies for all new allocations to come from the second volume, then verify:

```
# fsapadm define /data datapolicy catchvol
# fsapadm define /data metadatapolicy catchvol
# fsapadm assignfs /data datapolicy metadatapolicy
# fsapadm list /data
```

3. Add physical storage to the original LUN:

   Not shown, as this varies by array vendor.

4. Remove the policies:

```
# fsapadm delete /data datapolicy
# fsapadm delete /data metadatapolicy
```

5. Remove the second LUN[52]. The first command automatically moves any data on the second LUN to the first LUN:

```
# fsvoladm remove /data catchvol
# vxvset –g datadg rmvol vset1 catchvol
# vxassist –g datadg remove volume catchvol
# vxdg -g datadg rmdisk HDS99600_1
# vxdisk rm HDS99600_1
```

[50] Version 7 is only available with VxFS version 5.0 or higher.
[51] The privlen=32m argument in step 1a is not strictly necessary, but it is a recommended best practice. This is not needed with Storage Foundation version 5.0 or higher, since 32MB is the default.
[52] A final, optional step – removing the original volume from the volume set – is not shown. Administrators who have needed to use this procedure will presumably want to keep the volume set capability in case they should need it again! However, this can be done; the steps are described in the VxFS File System Administrator's Guide, Chapter 9: Multi-volume File Systems: http://ftp.support.veritas.com/pub/support/products/FileSystem_UNIX/290254.pdf

Defragmenting before Migration – Pros and Cons

The comments here apply to all file systems that provide a defragmentation utility, such as NTFS and VxFS. They are not specific to VxFS.

Defragmentation is not appropriate for most thin provisioning systems, because the act of defragmenting moves data within a LUN, which in turn triggers allocations of storage within the array. The result is lower utilization of the thinly provisioned storage.

However, putting those costs aside, a defragmented file system is better than a fragmented file system, even in a thinly provisioned LUN. A defragmented file system groups related data together, improving performance. A defragmented file system uses less metadata, which in turn uses both less storage and less memory. So, running 'one last' defragmentation before migrating to thin storage has clear benefits.

Yet defragmenting has costs. It takes clock time: defragmentation can take many hours for a file system with millions of files. It takes staff time: because the time to defragment is hard to predict, progress must be periodically checked. And some applications, like commercial databases, typically see little benefit from defragmentation[53].

Counterarguments are that defragmentation does not cause downtime for either VxFS or NTFS; it is run online, with applications using the file system all the while. And since both VxFS and NTFS have a 'cancel' mechanism if defrag is taking too long, these costs can be capped. VxFS also lets the user specify a time limit up front.

Defragmentation automatically happens using the "Reclaim before move" and the "Multi-volume file system" method, during the shrink step and fsapadm enforcefile step, respectively. It does not need to be run separately.

Ultimately, the decision to defrag before migration remains a judgment call.

The authors wish to thank Mike Root, Laura Shepard, and Thomas Cornely for their assistance.

---

[53] Also, some file systems are just not very fragmented. NTFS provides a simple "yes/no" recommendation on whether the file system is fragmented enough to bother defragmenting. A similar "yes/no" recommendation script for VxFS is available here: http://www.eng.auburn.edu/pub/mail-lists/ssastuff/vxdefrag-pl.html