

Symantec Storage Foundation and High Availability Solutions SmartIO Deployment Guide for Linux Platforms

Patric Uebele
March, 2015

Table of Contents

Introduction	3
SmartIO technology basics.....	3
SmartIO benefits	3
Configuring and managing cache areas using the <code>sfcache</code> CLI	6
Implementing SmartIO for VxVM read caching	7
Implementing SmartIO for VxFS read and write-back caching	10
Write-back caching	13
Customizing the caching behavior of SmartIO for VxFS	18
Read caching and clustered file systems	20
Configuring cache reflection	23
SmartIO caching for Oracle databases.....	25
Clustered Oracle database and SmartIO.....	28
SmartIO Caching in Virtualized Environments	30
SmartIO caching on the host in KVM/RHEV virtualized environments	30
Managing SmartIO with Veritas Operations Manager	34
Creating cache areas with Veritas Operations Manager	34
Modifying and deleting cache areas with Veritas Operations Manager	38
Enabling or disabling SmartIO caching with Veritas Operations Manager	40
Viewing cache details with Veritas Operations Manager	43
Appendix A: SmartIO terminology	46
Appendix B: <code>sfcache</code> command reference.....	47
Appendix C: <code>sfcache</code> statistics reference.....	49
Appendix D: Further reading.....	50
Appendix E: Known issues and limitations	51

Introduction

This document explains how you can use Symantec Storage Foundation and High Availability (SFHA) Solutions' SmartIO caching technology. It shows you how to setup and manage SmartIO for volumes and file systems, and it covers some application-specific topics.

In today's world of higher performance Service Level Agreements (SLAs) and shrinking budgets, it becomes very important to make better use of existing resources. With SmartIO, administrators can improve performance by introducing onboard Peripheral Component Interconnect (PCI) solid-state drives (SSDs). By introducing SmartIO in the Storage Area Network (SAN), the administrator can ease data congestion and improve I/O response times across the SAN.

SmartIO technology basics

The SmartIO feature has been introduced in SFHA Solutions for Linux version 6.1. It enables data efficiency on your SSDs through I/O caching. Using SmartIO to improve efficiency, you can optimize the cost per Input/Output Operations Per Second (IOPS). SmartIO does not require in-depth knowledge of the hardware technologies underneath. It uses advanced, customizable heuristics to determine what data to cache and how that data gets removed from the cache. The heuristics take advantage of SFHA Solutions' knowledge of workload characteristics.

SmartIO uses a cache area on the target device or devices. The cache area is the storage space that SmartIO uses to store the cached data and the metadata about the cached data. The cache area type determines whether it supports VxFS caching or VxVM caching. To start using SmartIO, you can create a cache area with a single command, while the application is online.

When the application issues an I/O request, SmartIO checks to see if the I/O can be serviced from the cache. As applications access data from the underlying volumes or file systems, certain data is moved to the cache based on the internal heuristics. Subsequent I/Os are processed from the cache.

SmartIO supports read and write-back caching for the VxFS file systems that are mounted on VxVM volumes, in several caching modes and configurations. SmartIO also supports block-level read caching for applications running on VxVM volumes.

SmartIO benefits

Simply by adding any solid state device, such as Peripheral Component Interconnect Express (PCIe), Serial ATA (SATA), or Serial SCSI (SAS), from any enterprise vendor into existing servers and configuring SmartIO, you can drastically improve the performance of IOPS-intense applications, reduce the storage costs, improve storage utilization, and even see cascaded performance gains on systems not equipped with local SSDs.

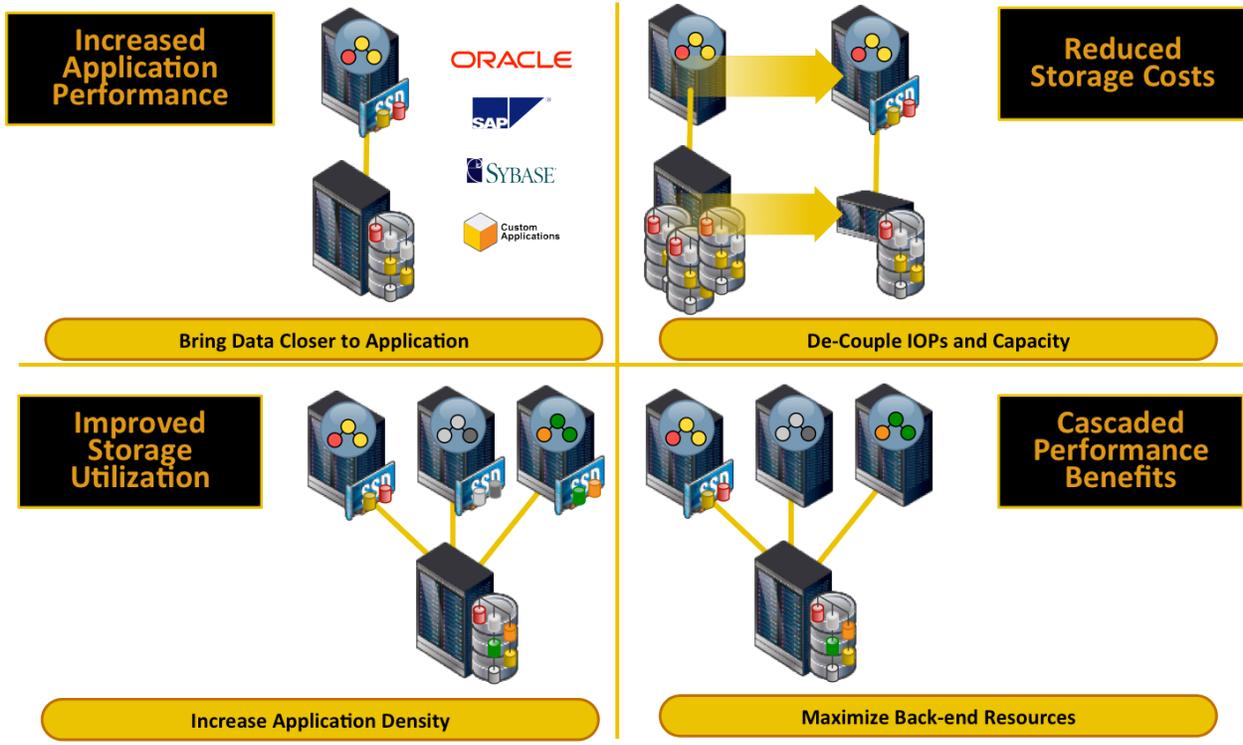


Figure 1: The four benefits of SmartIO

As shown in Figure 2, for a mixed online transaction processing (OLTP) workload, SmartIO provides more than 2.5 times the transactions per minute compared to running the same workload on a traditional Tier 1 array without server-side caching. With SmartIO enabled, the majority of read and write requests can be handled within the server, and therefore served at a much faster speed, significantly increasing application performance.

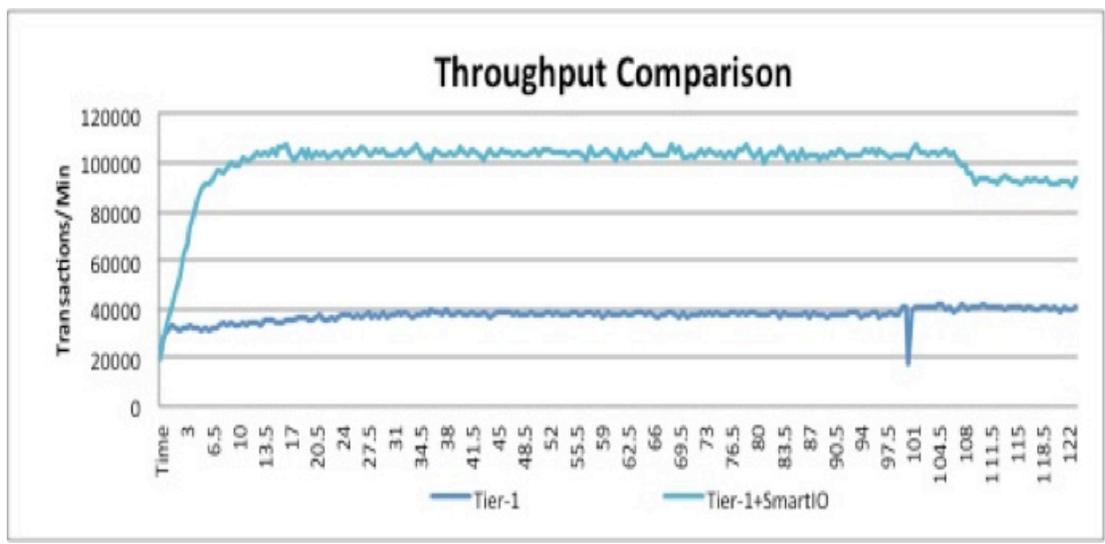


Figure 2: OLTP throughput comparison

While this is a significant gain for the application, other, secondary benefits from SmartIO also provide substantial value in an enterprise data center. As with SmartIO, the majority of random read requests

can be handled within the server. The performance needs of the back-end storage arrays are substantially lower. In a scenario with an EMC VMAX 20K array with 15K drives as a Tier 1 array and an EMC CLARiiON CX4 also with 15K drives as a Tier 2 array, the combination of SmartIO and the cheaper Tier 2 array brings better performance across multiple OLTP workloads than our Tier 1 array at 39% of the cost.

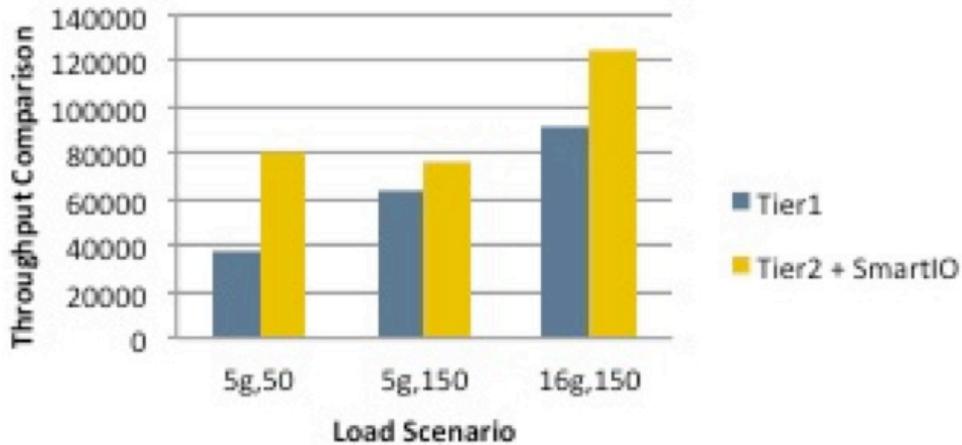


Figure 3: Tier 1 vs. Tier 2 + SmartIO

With the reduction of I/O-needs on the back-end storage array by SmartIO, you can increase the use of existing storage arrays without affecting application performance. Comparing four servers (each running an OLTP workload) attached to a SAN storage array without SmartIO with eight servers running the same workload with SmartIO enabled on the same array, the transactions per server are similar. Therefore, SmartIO lets you double the workload on the storage array without affecting the performance of the applications.

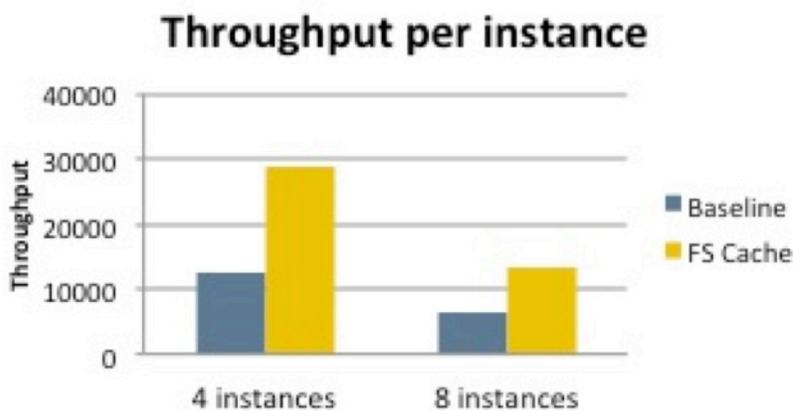


Figure 4: Instance stacking

Essentially, SmartIO caching technology lets you decouple the need for high IOPs and low latency (served by local SSDs) from the need for storage capacity (served by SAN storage).

Configuring and managing cache areas using the `sfcache` CLI

To use SmartIO caching capabilities, you must configure the cache areas to store the locally cached data.

For volume-based and file system-based caching, cache areas of different types are being used. You must configure separate cache areas for volume-based and file system-based caching if you want to cache both volumes and file system. (Note that with Storage Foundation version 6.1, the number of cache areas per system and type is limited to one, but each cache area can cache multiple volumes or file systems.)

First, we identify the SSD devices attached to the server using the `vxdisk` command:

```
[root@ia-lab-r720-03 ~]# vxdisk -o ssd list
DEVICE          SIZE (MB)      PHYS_ALLOC (MB)  GROUP          TYPE
RECLAIM_CMD
fusionio0_0     1149177       N/A              -              ssdtrim      TRIM
intel_ssd0_0    190782        N/A              -              ssd
```

We select the Fusion-io SSD device to use as the cache area and initialize it for use with VxVM:

```
[root@ia-lab-r720-03 ~]# vxdisksetup -i fusionio0_0
```

and then use the `sfcache` command to create a 100 GB cache area for VxVM use:

```
[root@ia-lab-r720-03 ~]# sfcache create -t VxVM 100g fusionio0_0 \
vxvm_cachevol
```

Next, we check how much space is still available on the SSD device:

```
[root@ia-lab-r720-03 ~]# sfcache maxsize fusionio0_0
Maxsize is 2143733760 (1046745Mb)
```

and create a 100 GB cache area for VxFS caching:

```
[root@ia-lab-r720-03 ~]# sfcache create -t VxFS 100g fusionio0_0 \
vxfs_cachevol
```

Alternatively, you can also create a VxVM disk group and volume on the SSD device and configure this volume as a Veritas File System (VxFS) or Veritas Volume Manager (VxVM) cache area with the `sfcache` command. However, resizing these cache areas is not supported, so the first method is highly recommended. Also, it is preferable to use a whole SSD for caching as this delivers more deterministic performance.

By default, the cache areas are created with the auto association type; that is, all existing and newly created VxVM volumes and VxFS file systems use the cache area automatically for read caching. You can change the association type with `sfcache`:

```
[root@ia-lab-r720-03 ~]# sfcache set --noauto vxfs_cachevol
```

We can now list the configured cache areas:

```
[root@ia-lab-r720-03 ~]# sfcache list
NAME          TYPE    SIZE    ASSOC-TYPE  STATE
DEVICE
vxvm_cachevol VxVM    100.00g  AUTO        ONLINE
fusionio0_0
```

```

vxfs_cachevol          VxFS    100.00g    NOAUTO    ONLINE
fusionio0_0

```

If needed, you can grow and shrink the cache areas with `sfcache resize`. For example to grow the VxFS cache area to 200 GB, use:

```
[root@ia-lab-r720-03 ~]# sfcache resize 200g vxfs_cachevol
```

You can check the result with `sfcache list`:

```

[root@ia-lab-r720-03 ~]# sfcache list
NAME                TYPE    SIZE      ASSOC-TYPE  STATE
DEVICE
vxvm_cachevol      VxVM    100.00g    AUTO        ONLINE
fusionio0_0
vxfs_cachevol      VxFS    200.00g    NOAUTO      ONLINE
fusionio0_0

```

and shrink the VxFS cache area again to 100 GB:

```
[root@ia-lab-r720-03 ~]# sfcache resize 100g vxfs_cachevol
```

Note: Shrinking a cache area throws out previously cached data.

Implementing SmartIO for VxVM read caching

In our test system, a volume (`vol2`) in disk group (`testdg`) was already present when we created the VxVM cache:

```

[root@ia-lab-r720-03 ~]# vxlist vol
TY  VOLUME      DISKGROUP      SIZE STATUS  LAYOUT  LINKAGE
vol  vol1        testdg         100.00g healthy  concat  -
vol  vol2        testdg         50.00g healthy  concat  -
vol  vxfs_cachevol sfcache_defaultdg 100.00g healthy  concat  -
vol  vxvm_cachevol sfcache_defaultdg 100.00g healthy  concat  -

```

Note that `vol1` is used for a VxFS file system and is discussed below. `vxfs_cachevol` and `vxvm_cachevol` are the cache areas created by the `sfcache create` commands above.

Because we created the cache area for the VxVM volume with the `auto` association type, caching for `vol2` is already enabled:

```

[root@ia-lab-r720-03 tmp]# sfcache list -l vxvm_cachevol
Cachearea: vxvm_cachevol
Assoc Type: AUTO
Type: VxVM
Size: 100.00g
Cacheline Size: 64.00k
Memory Size: 150.57m
State: ONLINE

ASSOCIATED DATA OBJECTS:

Volume: testdg/vol1
Size: 100.00g
State: AUTO

```

```

Kstate: STOPPED
Caching Mode: read

Volume: testdg/vol2
Size: 50.00g
State: ENABLED
Kstate: ENABLED
Caching Mode: read

```

(Note that `vol1` has a VxFS file system created on it, which is the reason why its `Kstate` is `STOPPED`. In the system output, `State` indicates the caching state for volume, while `Kstate` indicates the actual caching state of the volume. If a file system is mounted on top of a volume, and we have both the VxFS and VxVM cache areas, VxFS caching is used.)

The `sfcache stat` command lets you see various caching statistics like hit ratio and average response times (ART) for read I/Os satisfied from the cache (`Hit`) and not satisfied from the cache (`Miss`):

```

[root@ia-lab-r720-03 tmp]# sfcache stat vxvm_cachevol

```

NAME	%CACHE	HIT RATIO		ART(Hit)ms		ART(Miss)ms		BYTES		
		RD	WR	RD	WR	RD	WR	RD	WR	
TYPE: VxVM										
vxvm_cachevol	0.00	0.00	0.00	0.000	0.000	0.000	0.000	0.00	0.00	
ASSOCIATED DATA OBJECTS:										
testdg/vol1	0.00	0.00	0.00	0.000	0.000	0.000	0.000	0.00	0.00	
testdg/vol2	0.00	0.00	0.00	0.000	0.000	0.000	0.000	0.00	0.00	

Because `vol2` has not been used yet, all values are zero.

For a full description of SmartIO statistics, see Appendix C: `sfcache` statistics.

To see the effects of SmartIO, we generate some read load on volume `vol2` using the `dd` command:

```

[root@ia-lab-r720-03 tmp]# dd if=/dev/vx/rdisk/testdg/vol2 \
of=/dev/null bs=65536 count=10
10+0 records in
10+0 records out
655360 bytes (655 kB) copied, 0.0112241 s, 58.4 MB/s

```

Because this is the first read, the cache is still empty, and all data is satisfied from the disk drive (no cache hits):

```

[root@ia-lab-r720-03 tmp]# sfcache stat vxvm_cachevol

```

NAME	%CACHE	HIT RATIO		ART(Hit)ms		ART(Miss)ms		BYTES		
		RD	WR	RD	WR	RD	WR	RD	WR	
TYPE: VxVM										
vxvm_cachevol	0.00	0.00	0.00	0.000	0.000	0.926	0.000	0.00	0.00	
ASSOCIATED DATA OBJECTS:										
testdg/vol1	0.00	0.00	0.00	0.000	0.000	0.000	0.000	0.00	0.00	
testdg/vol2	0.00	0.00	0.00	0.000	0.000	0.926	0.000	0.00	0.00	

After reading another 10 records with `dd`:

```

[root@ia-lab-r720-03 tmp]# dd if=/dev/vx/rdisk/testdg/vol2 \
of=/dev/null bs=65536 count=10
10+0 records in
10+0 records out
655360 bytes (655 kB) copied, 0.00465974 s, 141 MB/s

```

we see a read cache ratio of 50% on vol2:

```
[root@ia-lab-r720-03 tmp]# sfcache stat vxvm_cachevol
```

NAME	HIT RATIO		ART (Hit)ms		ART (Miss)ms		BYTES		
	%CACHE	RD	WR	RD	WR	RD	WR	RD	WR
TYPE: VxVM									
vxvm_cachevol	0.00	50.00	0.00	0.366	0.000	0.926	0.000	640.00k	0.00
ASSOCIATED DATA OBJECTS:									
testdg/vol1	0.00	0.00	0.00	0.000	0.000	0.000	0.000	0.00	0.00
testdg/vol2	0.00	50.00	0.00	0.366	0.000	0.926	0.000	640.00k	0.00

The sfcache stat -l option gives us more details:

```
[root@ia-lab-r720-03 tmp]# sfcache stat -l vxvm_cachevol
```

```
TYPE: VxVM
Cache Area: vxvm_cachevol
Size: 100.00g
Cache Used: 640.00k (0.00%)
Cache Free: 99.99g
Avg. Hot data : 100.00%
Avg. Warm data: 0.00%
Avg. Cold data: 0.00%
Threshold: 0

READ STATISTICS:
Hit Ratio: 50.00
Avg. Rsp. Time(Hit): 0.366ms
Avg. Rsp. Time(Miss): 0.926ms
Avg. Total Rsp. Time: 0.646ms
Num. Read Operations: 20
Num. Read From Cache Operations: 10
Total Data Read: 1.25m
Data Read From Cache: 640.00k (50.00%)
Num. Read on Invalidated Data Operation: 0
Amount of Read on Invalidated Data: 0.00
Num. Read on Garbage Invalidated Data Operation: 0
Amount of Read on Garbage Invalidated Data: 0.00
Num. Cache Replacement Operation: 10
Cache Replaced Data: 640.00k
Partial Hit Data: 0.00
...
...
ASSOCIATED DATA OBJECTS:
...
...
Volume: testdg/vol2
Size: 50.00g
% Cache Used: 0.00
```

```
READ STATISTICS:
Hit Ratio: 50.00
Avg. Rsp. Time(Hit): 0.366ms
Avg. Rsp. Time(Miss): 0.926ms
Avg. Total Rsp. Time: 0.646ms
Num. Read Operations: 20
Num. Read From Cache Operations: 10
Total Data Read: 1.25m
Data Read From Cache: 640.00k (50.00%)
Num. Read on Invalidated Data Operation: 0
Amount of Read on Invalidated Data: 0.00
Num. Read on Garbage Invalidated Data Operation: 0
Amount of Read on Garbage Invalidated Data: 0.00
Num. Cache Replacement Operation: 10
```

```
Cache Replaced Data: 640.00k
Partial Hit Data: 0.00
```

After we run some more `dd` read commands as above, the cache hit rises beyond 90%, so the majority of read requests are satisfied from the SSD device:

```
[root@ia-lab-r720-03 tmp]# sfcache stat vxvm_cachevol
```

NAME	%CACHE		HIT RATIO		ART(Hit)ms		ART(Miss)ms		BYTES
	RD	WR	RD	WR	RD	WR	RD	WR	
TYPE: VxVM									
vxvm_cachevol	0.00	91.67	0.00	0.194	0.000	0.926	0.000	6.87m	0.00
ASSOCIATED DATA OBJECTS:									
testdg/vol1	0.00	0.00	0.00	0.000	0.000	0.000	0.000	0.00	0.00
testdg/vol2	0.00	91.67	0.00	0.194	0.000	0.926	0.000	6.87m	0.00

Implementing SmartIO for VxFS read and write-back caching

The VxFS cache created in “Configuring and managing cache areas using the `sfcache` CLI” has the `noauto` association type (the default is `auto`), so VxFS file systems on the server do not use SmartIO automatically (like the file system `/test1` created on VxVM volume `/vol1`):

```
[root@ia-lab-r720-03 tmp]# sfcache list -l vxfs_cachevol
```

```
Cachearea: vxfs_cachevol
Assoc Type: NOAUTO
Type: VxFS
Size: 100.00g
State: ONLINE
```

/dev/vx/dsk/sfcache_defaultdg/vxfs_cachevol:	SIZE	MODE	MOUNTPOINT
FSUID			
e23bb5532ae2030085b20000e271e84ae7030000e23bb553	4 KB	nocache	/test1

With VxFS file systems, you have two options for controlling the SmartIO caching of a file system. You can use the `sfcache` command or the `smartiomode` mount options. To start read caching on `/test1`, either (re)mount it with the `smartiomode=read`¹ mount option:

```
[root@ia-lab-r720-03 ~]# mount -t vxfs \
-o remount,smartiomode=read /dev/vx/dsk/testdg/vol1 /test1
```

or use the `sfcache` command:

```
[root@ia-lab-r720-03 ~]# sfcache enable /test1
```

and check it with `sfcache list`:

```
[root@ia-lab-r720-03 ~]# sfcache list -r /test1
```

```
/test1:
```

READ CACHE	WRITEBACK	MODE	PINNED	NAME
0 KB	0 KB	read	no	/test1

For demonstrating the read cache functionality, let's use `dd` to create a large file in `/test1`:

```
[root@ia-lab-r720-03 ~]# dd if=/dev/zero of=/test1/file1 bs=1M count=10240
```

¹ Note that a known issue in Storage Foundation 6.1 prevents the `smartiomode` mount option from enabling caching for a file system during mount or remount if the cache area has the `noauto` association type. Use the `sfcache enable` command to enable VxFS caching or auto cache areas to make sure that VxFS caching is enabled at boot time. This issue will be fixed in version 6.2.

```
10240+0 records in
10240+0 records out
10737418240 bytes (11 GB) copied, 97.6718 s, 110 MB/s
```

Confirm that the cache is still empty:

```
[root@ia-lab-r720-03 ~]# sfcache stat /test1
    Cache Size:      100 GB
Cache Utilization:    4 KB ( 0.00 %)

Read Cache
Hit Ratio   Data Read   Data Written

/test1:
  0.00 %      0 KB        0 KB
```

After reading the file completely for the first time:

```
[root@ia-lab-r720-03 ~]# dd if=/test1/file1 of=/dev/null bs=8k
1310720+0 records in
1310720+0 records out
10737418240 bytes (11 GB) copied, 310.518 s, 34.6 MB/s
```

we see that its whole content has also been written to the cache:

```
[root@ia-lab-r720-03 ~]# sfcache list /test1
/test1:
READ CACHE   WRITEBACK   MODE        PINNED   NAME
  10.0 GB      0 KB       read        no       /test1/file1
  10.0 GB      0 KB       read        no       /test1
[root@ia-lab-r720-03 ~]# sfcache stat /test1
    Cache Size:      100 GB
Cache Utilization:  10.0 GB (10.00 %)

Read Cache
Hit Ratio   Data Read   Data Written

/test1:
  0.00 %      152 KB      10.0 GB
```

Subsequent (direct) reads of the file perform much faster because they are satisfied from the read cache:

```
[root@ia-lab-r720-03 ~]# dd if=/test1/file1 of=/dev/null bs=8k iflag=direct
1310720+0 records in
1310720+0 records out
10737418240 bytes (11 GB) copied, 155.196 s, 69.2 MB/s
[root@ia-lab-r720-03 ~]# sfcache stat /test1
    Cache Size:      100 GB
Cache Utilization:  10.0 GB (10.00 %)

Read Cache
Hit Ratio   Data Read   Data Written

/test1:
  50.00 %     10.0 GB     10.0 GB
```

For example, a check with `iostat` while reading the file shows that all data blocks are being read from the Fusion-io flash device (on top of which the cache area has been built):

```
[root@ia-lab-r720-03 ~]# iostat 5 5
```

Linux 2.6.32-358.el6.x86_64 (ia-lab-r720-03) 07/08/2014 _x86_64_ (16 CPU)

...

```
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.38    0.00   2.85    0.01    0.00   96.76
```

```
Device:            tps    Blk_read/s    Blk_wrtn/s    Blk_read    Blk_wrtn
sdb                 3.19         0.00         196.41         0          984
sdd                 0.00         0.00          0.00         0           0
sdc                 0.00         0.00          0.00         0           0
sda                 0.00         0.00          0.00         0           0
sdf                 0.00         0.00          0.00         0           0
sde                 0.00         0.00          0.00         0           0
fioa                8547.50      136760.08       0.00        685168         0
VxVM10000           0.00         0.00          0.00         0           0
VxVM10001           0.00         0.00          0.00         0           0
```

The cache content is also preserved after we unmount the VxFS file system:

```
[root@ia-lab-r720-03 ~]# umount /test1
[root@ia-lab-r720-03 ~]# mount /test1
[root@ia-lab-r720-03 ~]# sfcache enable /test1
[root@ia-lab-r720-03 ~]# sfcache stat /test1
      Cache Size:      100 GB
Cache Utilization:    10.0 GB (10.00 %)

Read Cache
Hit Ratio    Data Read    Data Written

/test1:
  75.68 %      40.0 GB      12.85 GB
[root@ia-lab-r720-03 ~]# dd if=/test1/file1 of=/dev/null bs=8k iflag=direct
1310720+0 records in
1310720+0 records out
10737418240 bytes (11 GB) copied, 145.296 s, 73.9 MB/s
[root@ia-lab-r720-03 ~]# sfcache stat /test1
      Cache Size:      100 GB
Cache Utilization:    10.0 GB (10.00 %)

Read Cache
Hit Ratio    Data Read    Data Written

/test1:
  79.55 %      50.0 GB      12.85 GB
```

When you write to a file that is already in the read cache, the cached content is also updated during the write. In the example below, `file1`, which is 10 GB and located in `/test1`, is read once to populate the cache. We then overwrite 5 GB of `file1`:

```
[root@ia-lab-r720-03 ~]# sfcache stat /test1
      Cache Size:      100 GB
Cache Utilization:    10.0 GB (10.00 %)

Read Cache
Hit Ratio    Data Read    Data Written

/test1:
  50.00 %      10.0 GB      10 GB

[root@ia-lab-r720-03 ~]# dd if=/dev/zero of=/test1/file2 bs=1M count=5120 \
oflag=direct conv=notrunc
5120+0 records in
5120+0 records out
5368709120 bytes (5.4 GB) copied, 69.0816 s, 77.7 MB/s
```

This results in another 5 GB also written to the cache area:

```
[root@ia-lab-r720-03 ~]# sfcache stat /test1
Cache Size:      100 GB
Cache Utilization: 10.0 GB (10.00 %)

Read Cache
Hit Ratio      Data Read      Data Written

/test1:
50.00 %        10.0 GB        15 GB
```

Note that you need to specify the `dd` write command with the `conv=notrunc` option so the file is not truncated before it is written to. Truncating the file invalidates the cached content of the file.

Write-back caching

SmartIO provides write caching in the write-back mode. In write-back mode, an application write returns success after the data is written to the SmartIO cache, which is usually on an SSD. At a later time, SmartIO flushes the cache, which writes the dirty data to the disk. Write-back caching expects to improve the latencies of **synchronous** user data writes. For each file system with write-back caching enabled, a 512 MB circular log is reserved with the VxFS cache area for dirty data. When write-back caching is enabled, read caching is implicitly enabled, too.

Let's create another file system, `/test3`, for demonstrating write-back caching and set the association type of the VxFS cache area back to its default value, `auto`:

```
[root@ia-lab-r720-03 ~]# mkfs -t vxfs /dev/vx/rdisk/testdg/vol13
version 10 layout
209715200 sectors, 104857600 blocks of size 1024, log size 65536 blocks
rcq size 4096 blocks
largefiles supported
maxlink supported
[root@ia-lab-r720-03 ~]# sfcache set --auto vxfs_cachevol
[root@ia-lab-r720-03 ~]# sfcache list
NAME                TYPE  SIZE      ASSOC-TYPE  STATE
DEVICE
vxvm_cachevol       VxVM  100.00g   AUTO        ONLINE      fusionio0_0
vxfs_cachevol       VxFS  100.00g   AUTO        ONLINE
fusionio0_0,fusionio0_0
```

After mounting `/test3` with just the default mount options, it is automatically enabled for read caching:

```
[root@ia-lab-r720-03 ~]# mount -t vxfs /dev/vx/dsk/testdg/vol13 \
/test3
[root@ia-lab-r720-03 ~]# sfcache list /test3
/test3:
READ CACHE  WRITEBACK  MODE      PINNED  NAME
0 KB        0 KB      read      no      /test3
```

Write-back caching can be enabled by remounting the file system with the `smartiomode=writeback` mount option:

```
[root@ia-lab-r720-03 ~]# mount -t vxfs /dev/vx/dsk/testdg/vol13 \
-o remount,smartiomode=writeback /test3

[root@ia-lab-r720-03 ~]# sfcache list /test3
```

```

/test3:
READ CACHE      WRITEBACK      MODE      PINNED      NAME
      0 KB              0 KB      writeback  no          /test3

```

Note: To make write-back caching mode persistent across system restarts for a file system, specify the `smartiomode=writeback` mount option in `/etc/fstab` or in the cluster configuration.

After enabling write-back caching for `/test3`, we see that 512 MB have been reserved for dirty data:

```

[root@ia-lab-r720-03 ~]# sfcache stat /test3
      Cache Size:      100 GB
Cache Utilization:  512.0 MB ( 0.50 %)

Read Cache
Hit Ratio      Data Read      Data Written      Writeback
Hit Ratio      Data Written

/test3:
      0.00 %              0 KB              0 KB      0.00 %              0 KB

```

In order to show the effects of write-back caching, let's compare the file systems `/test1` with read caching and `/test3` with write-back caching:

```

[root@ia-lab-r720-03 ~]# sfcache list /test1
/test1:
READ CACHE      WRITEBACK      MODE      PINNED      NAME
      0 KB              0 KB      read      no          /test1
[root@ia-lab-r720-03 ~]# sfcache list /test3
/test3:
READ CACHE      WRITEBACK      MODE      PINNED      NAME
      0 KB              0 KB      writeback  no          /test3

```

by running a quick file system benchmark with write workloads on each file system. We use the IOzone benchmarking tool (www.iozone.org) with the write, rewrite, and random read/write tests writing ten 100 MB files with 8 kB records. As write-back caching is effective only for synchronous writes, we use the `iozone -i` and `-o` switches:

```

[root@ia-lab-r720-03 test1]# /opt/iozone/bin/iozone -i 0 -i 2 -i 4 -I -o \
-t 10 -r 8k -s 100m
Iozone: Performance Test of File I/O
      Version $Revision: 3.424 $
      Compiled for 32 bit mode.
      Build: linux

Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
      Al Slater, Scott Rhine, Mike Wisner, Ken Goss
      Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
      Randy Dunlap, Mark Montague, Dan Million, Gavin Brebner,
      Jean-Marc Zucconi, Jeff Blomberg, Benny Halevy, Dave Boone,
      Erik Habbinga, Kris Strecker, Walter Wong, Joshua Root,
      Fabrice Bacchella, Zhenghua Xue, Qin Li, Darren Sawyer,
      Vangel Bojaxhi, Ben England, Vikentsi Lapa.

Run began: Sat Jul 12 12:28:16 2014

O_DIRECT feature enabled
SYNC Mode.
Record Size 8 kB
File size set to 102400 kB
Command line used: /opt/iozone/bin/iozone -i 0 -i 2 -i 4 -I -o -t 10 -r
8k -s 100m

```

Output is in kBytes/sec
 Time Resolution = 0.000001 seconds.
 Processor cache size set to 1024 kBytes.
 Processor cache line size set to 32 bytes.
 File stride size set to 17 * record size.
 Throughput test with 10 processes
 Each process writes a 102400 kByte file in 8 kByte records

Children see throughput for 10 initial writers = 1432.84 kB/sec
 Parent sees throughput for 10 initial writers = 1333.59 kB/sec
 Min throughput per process = 129.20 kB/sec
 Max throughput per process = 155.19 kB/sec
 Avg throughput per process = 143.28 kB/sec
 Min xfer = 85256.00 kB

Children see throughput for 10 rewriters = 1360.21 kB/sec
 Parent sees throughput for 10 rewriters = 1360.14 kB/sec
 Min throughput per process = 113.62 kB/sec
 Max throughput per process = 149.24 kB/sec
 Avg throughput per process = 136.02 kB/sec
 Min xfer = 77968.00 kB

Children see throughput for 10 random readers = 6695.73 kB/sec
 Parent sees throughput for 10 random readers = 6694.93 kB/sec
 Min throughput per process = 661.14 kB/sec
 Max throughput per process = 681.80 kB/sec
 Avg throughput per process = 669.57 kB/sec
 Min xfer = 99312.00 kB

Children see throughput for 10 random writers = 2205.72 kB/sec
 Parent sees throughput for 10 random writers = 2136.28 kB/sec
 Min throughput per process = 215.42 kB/sec
 Max throughput per process = 226.58 kB/sec
 Avg throughput per process = 220.57 kB/sec
 Min xfer = 97360.00 kB

iozone test complete.

Through the random reads, the /test1 read cache has been populated:

```
[root@ia-lab-r720-03 ~]# sfcache stat /test1
  Cache Size:      100 GB
Cache Utilization:    8 KB ( 0.00 %)

Read Cache
Hit Ratio   Data Read   Data Written

/test1:
  0.00 %           0 KB       1.917 GB
```

Running the same workload on /test3 with write-back cache enabled:

```
[root@ia-lab-r720-03 test3]# /opt/iozone/bin/iozone -i 0 -i 2 -i 4 -I -o \
-t 10 -r 8k -s 100m
Iozone: Performance Test of File I/O
  Version $Revision: 3.424 $
  Compiled for 32 bit mode.
  Build: linux

Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
```

Al Slater, Scott Rhine, Mike Wisner, Ken Goss
Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
Randy Dunlap, Mark Montague, Dan Million, Gavin Brebner,
Jean-Marc Zucconi, Jeff Blomberg, Benny Halevy, Dave Boone,
Erik Habbinga, Kris Strecker, Walter Wong, Joshua Root,
Fabrice Bacchella, Zhenghua Xue, Qin Li, Darren Sawyer,
Vangel Bojaxhi, Ben England, Vikentsi Lapa.

Run began: Sat Jul 12 13:09:43 2014

```
O_DIRECT feature enabled
SYNC Mode.
Record Size 8 kB
File size set to 102400 kB
Command line used: /opt/iozone/bin/iozone -i 0 -i 2 -i 4 -I -o -t 10 -r
8k -s 100m
Output is in kBytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 1024 kBytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
Throughput test with 10 processes
Each process writes a 102400 kByte file in 8 kByte records

Children see throughput for 10 initial writers      = 8747.61 kB/sec
Parent sees throughput for 10 initial writers = 8622.86 kB/sec
Min throughput per process = 867.91 kB/sec
Max throughput per process = 877.26 kB/sec
Avg throughput per process = 874.76 kB/sec
Min xfer = 101312.00 kB

Children see throughput for 10 rewriters = 8897.02 kB/sec
Parent sees throughput for 10 rewriters = 8833.66 kB/sec
Min throughput per process = 888.98 kB/sec
Max throughput per process = 890.18 kB/sec
Avg throughput per process = 889.70 kB/sec
Min xfer = 102368.00 kB

Children see throughput for 10 random readers = 6408.30 kB/sec
Parent sees throughput for 10 random readers = 6406.54 kB/sec
Min throughput per process = 627.66 kB/sec
Max throughput per process = 649.12 kB/sec
Avg throughput per process = 640.83 kB/sec
Min xfer = 99024.00 kB

Children see throughput for 10 random writers = 6221.93 kB/sec
Parent sees throughput for 10 random writers = 5768.64 kB/sec
Min throughput per process = 621.77 kB/sec
Max throughput per process = 622.51 kB/sec
Avg throughput per process = 622.19 kB/sec
Min xfer = 102368.00 kB
```

iozone test complete.

We see a 100% write-back cache hit ratio:

```
[root@ia-lab-r720-03 ~]# sfcache stat /test3
Cache Size: 100 GB
Cache Utilization: 512.0 MB ( 0.50 %)
```

Read Cache

Writeback

	Hit Ratio	Data Read	Data Written	Hit Ratio	Data Written
/test3:	0.00 %	0 KB	1.927 GB	100.00 %	2.93 GB

The following table compares the average throughputs (in kB/s) per process for the four workloads:

	Initial write	Re-write	Random read	Random write
Read cache	143.28	136.02	669.57	220.57
Write-back cache	874.76	889.70	640.33	622.19

The data clearly shows the performance improvements through write-back caching for synchronous and direct write workloads.

SmartIO write-back caching caches only direct or synchronous writes of up to 2 MB I/O size. So running the same iозone tests with a 4 MB record size:

```
[root@ia-lab-r720-03 test3]# /opt/iozone/bin/iozone -i 0 -i 2 -i 4 -I -o \
-t 10 -r 4m -s 100m
  Iozone: Performance Test of File I/O
    Version $Revision: 3.424 $
    Compiled for 32 bit mode.
    Build: linux

Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
  Al Slater, Scott Rhine, Mike Wisner, Ken Goss
  Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
  Randy Dunlap, Mark Montague, Dan Million, Gavin Brebner,
  Jean-Marc Zucconi, Jeff Blomberg, Benny Halevy, Dave Boone,
  Erik Habbinga, Kris Strecker, Walter Wong, Joshua Root,
  Fabrice Bacchella, Zhenghua Xue, Qin Li, Darren Sawyer,
  Vangel Bojaxhi, Ben England, Vikentsi Lapa.

Run began: Sat Jul 12 14:07:59 2014

O_DIRECT feature enabled
SYNC Mode.
Record Size 4096 kB
File size set to 102400 kB
Command line used: /opt/iozone/bin/iozone -i 0 -i 2 -i 4 -I -o -t 10 -r
4m -s 100m
Output is in kBytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 1024 kBytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
Throughput test with 10 processes
Each process writes a 102400 kByte file in 4096 kByte records

Children see throughput for 10 initial writers      = 99945.15 kB/sec
Parent sees throughput for 10 initial writers      = 86881.04 kB/sec
Min throughput per process                        = 8793.96 kB/sec
Max throughput per process                        = 12124.86 kB/sec
Avg throughput per process                        = 9994.52 kB/sec
Min xfer                                          = 77824.00 kB

Children see throughput for 10 rewriters           = 109999.95 kB/sec
Parent sees throughput for 10 rewriters           = 109146.19 kB/sec
```

```

Min throughput per process           =    9444.78 kB/sec
Max throughput per process           =   12427.46 kB/sec
Avg throughput per process           =   11000.00 kB/sec
Min xfer                             =   77824.00 kB

Children see throughput for 10 random readers = 118934.32 kB/sec
Parent sees throughput for 10 random readers = 117505.28 kB/sec
Min throughput per process           =   10368.19 kB/sec
Max throughput per process           =   13910.83 kB/sec
Avg throughput per process           =   11893.43 kB/sec
Min xfer                             =   77824.00 kB

Children see throughput for 10 random writers =  75301.09 kB/sec
Parent sees throughput for 10 random writers =  62526.09 kB/sec
Min throughput per process           =    4573.72 kB/sec
Max throughput per process           =    9410.02 kB/sec
Avg throughput per process           =    7530.11 kB/sec
Min xfer                             =   53248.00 kB

```

iozone test complete.

results in a 0% write-back cache hit ratio:

```

[root@ia-lab-r720-03 test3]# sfcache stat /test3
    Cache Size:      100 GB
Cache Utilization:  512.0 MB ( 0.50 %)

Read Cache
Hit Ratio   Data Read   Data Written   Writeback
Hit Ratio   Data Written

/test3:
  0.00 %      0 KB      1.578 GB      0.00 %      0 KB

```

Customizing the caching behavior of SmartIO for VxFS

You can use the `sfcache` command to customize the behavior of SmartIO. For a complete overview, see the [Symantec Storage Foundation and High Availability Solutions SmartIO for Solid State Drives Solutions Guide](#).

You can preload specific files or directories into the cache before I/Os access the file for the first time. Starting with two files in `/test1` and an empty cache

```

[root@ia-lab-r720-03 ~]# ls -l /test1
total 20971536
-rw-r--r-- 1 root root 10737418240 Jul 10 05:06 file1
-rw-r--r-- 1 root root 10737418240 Jul 10 04:57 file2
drwxr-xr-x 2 root root          96 Jul  3 04:17 lost+found
[root@ia-lab-r720-03 ~]# sfcache stat /test1
    Cache Size:      100 GB
Cache Utilization:    4 KB ( 0.00 %)

Read Cache
Hit Ratio   Data Read   Data Written

/test1:
  0.00 %      0 KB      0 KB

```

we preload file2 of 10 GB into the cache:

```
[root@ia-lab-r720-03 ~]# sfcache load /test1/file2
```

By default, the `sfcache load` command returns asynchronously. After some time, we see that the cache has been populated with the file content:

```
[root@ia-lab-r720-03 ~]# sfcache stat /test1
Cache Size:      100 GB
Cache Utilization: 10.0 GB (10.00 %)

Read Cache
Hit Ratio      Data Read      Data Written

/test1:
0.00 %         0 KB           10 GB
```

You can also prevent files from being evicted by the SmartIO caching algorithm with the `sfcache pin` command. Pinned files are kept in the cache indefinitely, until they are deleted or explicitly unpinned.

A pinned file is not loaded into the cache automatically. It is cached based on I/O access:

```
[root@ia-lab-r720-03 ~]# sfcache pin /test3/file1
[root@ia-lab-r720-03 ~]# sfcache list /test3
/test3:
READ CACHE      WRITEBACK      MODE          PINNED      NAME
0 KB            0 KB           writeback     yes         /test3/file1
0 KB            0 KB           writeback     no          /test3
```

You can combine the `pin` operation with the `load` operation. The file is loaded synchronously into the cache:

```
[root@ia-lab-r720-03 ~]# sfcache pin -o load /test3/file1
[root@ia-lab-r720-03 ~]# sfcache list /test3
/test3:
READ CACHE      WRITEBACK      MODE          PINNED      NAME
10.0 GB         0 KB           writeback     yes         /test3/file1
10.0 GB         0 KB           writeback     no          /test3
```

Furthermore, you can set caching modes for files or directories to either `nocache` (preventing a file or directory from being cached at all), `read` (no writes are cached) or `writeback` (enables read and write-back caching for a file or directory) with the `sfcache set mode` command (see

Appendix B: `sfcache` command reference).

Read caching and clustered file systems

A cache area is private to each node in a cluster, also when using a clustered file system. Each node in the cluster maintains its own cache area, and caching occurs on a per-node basis. The cache contents are not shared across the nodes in the cluster. A clustered file system with caching enabled is associated with the local cache area on each node.

For demonstrating SmartIO file system caching with a clustered file system, we have configured a two-node cluster (`ia-lab-r720-03` and `ia-lab-r720-08`) with Symantec Storage Foundation Cluster File System (SFCFS) and created a clustered file system, `/share1`, on a shared volume which is mounted on both nodes:

```
[root@ia-lab-r720-03 ~]# cfsmntadm display -v /share1
Mount Point      : /share1
Shared Volume    : sharevoll
Disk Group       : sharedg

Primary Node:    ia-lab-r720-03
Primary Election Policy:
Dependent Checkpoints or Snapshots:

NODE NAME        STATUS           MOUNT OPTIONS
ia-lab-r720-03  MOUNTED         suid,rw
ia-lab-r720-08  MOUNTED         suid,rw
```

Both nodes have a 100 GB VxFS cache area of the `auto` association type:

```
[root@ia-lab-r720-03 ~]# sfcache list
NAME           TYPE  SIZE    ASSOC-TYPE  STATE    DEVICE
vxfs_cachevol VxFS  100.00g  AUTO        ONLINE   fusionio0_0

[root@ia-lab-r720-08 ~]# sfcache list
NAME           TYPE  SIZE    ASSOC-TYPE  STATE    DEVICE
vxfs_cachevol VxFS  100.00g  AUTO        ONLINE   fusionio0_0
```

And a 10 GB file, `file1`, resides in the shared file system which is automatically enabled for read caching:

```
[root@ia-lab-r720-03 ~]# sfcache list /share1/
/share1:
READ CACHE    WRITEBACK    MODE        PINNED  NAME
0 KB          0 KB        read        no      /share1/file1
0 KB          0 KB        read        no      /share1

[root@ia-lab-r720-08 ~]# sfcache list /share1/
/share1:
READ CACHE    WRITEBACK    MODE        PINNED  NAME
0 KB          0 KB        read        no      /share1/file1
0 KB          0 KB        read        no      /share1
```

As the cache areas are private to each node, a first read of file `file1` on each node populates the respective cache:

```
[root@ia-lab-r720-03 ~]# dd if=/share1/file1 of=/dev/null bs=8k
1310720+0 records in
1310720+0 records out
10737418240 bytes (11 GB) copied, 113.325 s, 94.7 MB/s
[root@ia-lab-r720-03 ~]# sfcache stat -l /share1
```

```

Cache Size:      100 GB
Cache Utilization: 10.0 GB (10.00 %)

Read Cache
Hit Ratio      Data Read      Data Written      Files Cached      Files Pinned      Data Pinned
/share1:
  0.00 %          152 KB           10 GB             1                 0                 0 KB

[root@ia-lab-r720-08 ~]# dd if=/share1/file1 of=/dev/null bs=8k
1310720+0 records in
1310720+0 records out
10737418240 bytes (11 GB) copied, 128.298 s, 83.7 MB/s
[root@ia-lab-r720-08 ~]# sfcache stat -l /share1
Cache Size:      100 GB
Cache Utilization: 10.0 GB (10.00 %)

Read Cache
Hit Ratio      Data Read      Data Written      Files Cached      Files Pinned      Data Pinned
/share1:
  0.00 %          96 KB           10 GB             1                 0                 0 KB

```

Before reading the file a second time from both nodes, let's unmount and remount the shared file system to clear the file system's page caches:

```

[root@ia-lab-r720-03 ~]# cfsu mount /share1
Unmounting...
/share1 got successfully unmounted from ia-lab-r720-08
/share1 got successfully unmounted from ia-lab-r720-03
[root@ia-lab-r720-03 ~]# cfs mount /share1
Mounting...
[/dev/vx/dsk/sharedg/sharevol1] mounted successfully at /share1 on ia-lab-r720-03
[/dev/vx/dsk/sharedg/sharevol1] mounted successfully at /share1 on ia-lab-r720-08

```

We can see that subsequent reads of file1 from both nodes are satisfied by the local caches at a much higher speed:

```

[root@ia-lab-r720-03 ~]# dd if=/share1/file1 of=/dev/null bs=8k
1310720+0 records in
1310720+0 records out
10737418240 bytes (11 GB) copied, 39.687 s, 271 MB/s
[root@ia-lab-r720-08 ~]# dd if=/share1/file1 of=/dev/null bs=8k
1310720+0 records in
1310720+0 records out
10737418240 bytes (11 GB) copied, 40.1163 s, 268 MB/s

[root@ia-lab-r720-03 ~]# sfcache stat -l /share1
Cache Size:      100 GB
Cache Utilization: 10.0 GB (10.00 %)

Read Cache
Hit Ratio      Data Read      Data Written      Files Cached      Files Pinned      Data Pinned
/share1:
  50.00 %       10.0 GB           10 GB             1                 0                 0 KB
[root@ia-lab-r720-08 ~]# sfcache stat -l /share1
Cache Size:      100 GB
Cache Utilization: 10.0 GB (10.00 %)

Read Cache
Hit Ratio      Data Read      Data Written      Files Cached      Files Pinned      Data Pinned
/share1:
  50.00 %       10.0 GB           10 GB             1                 0                 0 KB

```

Rewriting the same file partially, without truncation, from the first node again invalidates the cache on the second node (cache utilization drops to zero), and updates the cache on the first node²:

```
[root@ia-lab-r720-03 ~]# dd if=/dev/zero of=/share1/file1 bs=1M count=5120 conv=notrunc
5120+0 records in
5120+0 records out
5368709120 bytes (5.4 GB) copied, 23.1913 s, 231 MB/s

[root@ia-lab-r720-03 ~]# sfcache stat -l /share1/
Cache Size:      100 GB
Cache Utilization: 10.0 GB (10.00 %)

Read Cache
Hit Ratio      Data Read      Data Written    Files Cached    Files Pinned    Data Pinned
/share1/:
 50.00 %      10.0 GB        15 GB          1               0               0 KB

[root@ia-lab-r720-08 ~]# sfcache stat -l /share1/
Cache Size:      100 GB
Cache Utilization: 4 KB ( 0.00 %)

Read Cache
Hit Ratio      Data Read      Data Written    Files Cached    Files Pinned    Data Pinned
/share1/:
 50.00 %      10.0 GB        10 GB          1               0               0 KB
```

Another read on the second node must be satisfied from disk, and the cache is being repopulated:

```
[root@ia-lab-r720-08 ~]# dd if=/share1/file1 of=/dev/null bs=8k
1310720+0 records in
1310720+0 records out
10737418240 bytes (11 GB) copied, 114.972 s, 93.4 MB/s
[root@ia-lab-r720-08 ~]# sfcache stat -l /share1
Cache Size:      100 GB
Cache Utilization: 10.0 GB (10.00 %)

Read Cache
Hit Ratio      Data Read      Data Written    Files Cached    Files Pinned    Data Pinned
/share1:
 0.00 %        112 KB        20 GB          1               0               0 KB
```

pin operations on files in clustered file systems are shared across all the nodes, while load operations remain local to each node:

```
[root@ia-lab-r720-03 ~]# sfcache pin -o load /share1/file2
[root@ia-lab-r720-03 ~]# sfcache list /share1/
/share1:
READ CACHE      WRITEBACK      MODE          PINNED  NAME
 0 KB           0 KB          read         no      /share1/file1
 10.0 GB        0 KB          read         yes     /share1/file2
 10.0 GB        0 KB          read         no      /share1
[root@ia-lab-r720-08 ~]# sfcache list /share1/
/share1:
READ CACHE      WRITEBACK      MODE          PINNED  NAME
 0 KB           0 KB          read         no      /share1/file1
 0 KB           0 KB          read         yes     /share1/file2
 0 KB           0 KB          read         no      /share1
```

² If the file is open in multiple nodes in the cluster, and one node writes to it, then only the portions we write to are invalidated on other nodes where the file is open. On nodes where the file is not open, the entire file is invalidated.

Configuring cache reflection

In the case of a clustered VxFS file system (CFS) with **two nodes**, when write-back caching is enabled, SmartIO mirrors the write-back data at the file system level to the other node's SSD cache. If a node fails, this behavior, called **cache reflection**, prevents loss of write-back data. If a node fails, the other node flushes the mirrored dirty data of the lost node as part of reconfiguration. Cache reflection ensures that write-back data is not lost even if a node fails with pending dirty data. To avoid negative effects on write performance due to cache reflection, the use of a high-speed, low latency cluster interconnect is highly recommended.

Currently cache reflection is only supported with two-node clusters, when you add a third node to the cluster, write-back caching is disabled on the clustered file system.

For the shared file system `/share1`, we enable write-back caching (and hence cache reflection in the two-node cluster) by remounting it with the `smartiomode=writeback` mount option:

```
[root@ia-lab-r720-03 ~]# cfsmntadm modify /share1 all+=smartiomode=writeback
Cluster-configuration updated with changed Mount Options for node ia-lab-r720-03
Mount Point /share1 remounted successfully on ia-lab-r720-03
Cluster-configuration updated with changed Mount Options for node ia-lab-r720-08
Mount Point /share1 remounted successfully on ia-lab-r720-08

[root@ia-lab-r720-03 ~]# mount -t vxfs -v
/dev/vx/dsk/sharedg/sharevoll on /share1 type vxfs
(rw,mntlock=VCS,cluster,crw,delaylog,largefiles,ioerror=mdisable,smartiomode=writeback)
[root@ia-lab-r720-08 ~]# mount -t vxfs -v
/dev/vx/dsk/sharedg/sharevoll on /share1 type vxfs
(rw,mntlock=VCS,cluster,crw,delaylog,largefiles,ioerror=mdisable,smartiomode=writeback)

[root@ia-lab-r720-03 ~]# sfcache list /share1
/share1:
READ CACHE      WRITEBACK      MODE           PINNED  NAME
      0 KB          0 KB      writeback     no      /share1/file1
      0 KB          0 KB      writeback    no      /share1
[root@ia-lab-r720-08 ~]# sfcache list /share1
/share1:
READ CACHE      WRITEBACK      MODE           PINNED  NAME
      0 KB          0 KB      writeback     no      /share1/file1
      0 KB          0 KB      writeback    no      /share1
```

512 MB of write-back cache is reserved in the cache area of each node for the local and remote write-back log, respectively. This results in a total of 1 GB cache utilization on every node:

```
[root@ia-lab-r720-03 ~]# sfcache stat /share1
Cache Size:      100 GB
Cache Utilization:  1.0 GB ( 1.00 %)

Read Cache
Hit Ratio    Data Read    Data Written    Writeback
Hit Ratio    Data Written
/share1:
0.00 %      0 KB          0 KB          0.00 %      0 KB
[root@ia-lab-r720-08 ~]# sfcache stat /share1
Cache Size:      100 GB
Cache Utilization:  1.0 GB ( 1.00 %)
```

Read Cache				Writeback	
Hit Ratio	Data Read	Data Written		Hit Ratio	Data Written
/share1:					
0.00 %	0 KB	0 KB		0.00 %	0 KB

When we issue a write operation on the first node that qualifies for write-back caching:

```
[root@ia-lab-r720-03 ~]# dd if=/dev/zero of=/share1/file1 bs=1M count=10240 \
oflag=direct,sync
10240+0 records in
10240+0 records out
10737418240 bytes (11 GB) copied, 428.459 s, 25.1 MB/s
[root@ia-lab-r720-03 share1]# sfcache stat -l /share1
Cache Size: 100 GB
Cache Utilization: 1.0 GB ( 1.00 %)

Read Cache
Hit Ratio  Data Read  Data Written  Files Cached  Files Pinned  Data Pinned  Writeback
/share1:   Hit Ratio  Data Written  Data Written  Data Pinned  Hit Ratio  Data Written
0.00 %    0 KB         0 KB         0             0             0 KB        100.00 %   10 GB
```

Cache reflection also sends cache data across the cluster interconnect to the write-back cache on the second node:

```
[root@ia-lab-r720-03 ~]# lltstat -v | grep packets
11797727 Snd data packets
16058 Snd connect packets
999 Snd loopback packets
1311738 Rcv data packets
1311732 Rcv data packets in-sequence
[root@ia-lab-r720-08 ~]# lltstat -v | grep packets
1311738 Snd data packets
15996 Snd connect packets
345 Snd loopback packets
11797727 Rcv data packets
1119 Rcv data packets in-sequence
```

With write operations that don't qualify for write-back caching (in our example below, the block size is larger than 2 MB), no cache data is sent across the cluster interconnect:

```
[root@ia-lab-r720-03 ~]# dd if=/dev/zero of=/share1/file2 bs=4M \
count=1024 oflag=direct,sync
1024+0 records in
1024+0 records out
4294967296 bytes (4.3 GB) copied, 9.9217 s, 433 MB/s
[root@ia-lab-r720-03 ~]# lltstat -v | grep packets
414 Snd data packets
5332 Snd connect packets
326 Snd loopback packets
347 Rcv data packets
347 Rcv data packets in-sequence
[root@ia-lab-r720-08 ~]# lltstat -v | grep packets
347 Snd data packets
5394 Snd connect packets
117 Snd loopback packets
414 Rcv data packets
414 Rcv data packets in-sequence
```

SmartIO caching for Oracle databases

SmartIO provides application templates to optimize caching for databases running on VxFS file systems or VxVM raw volumes. SmartIO uses the templates to apply policies to particular types of information in the database. For example, index files may have different caching policies from data files. Currently, for VxVM raw volumes, templates are available for Oracle and Sybase databases, and for VxFS file systems, a template for Oracle databases is available.

Let's look at the SmartIO plugin for Oracle on VxFS file systems with a small sample database `smartiodb` (Oracle 11.2.0.4) using the mount point `/u02` to store its data.

The intended user for the SmartIO plugin for Oracle is the Oracle database administrator (DBA), but root privileges are required for some operations. So before trying to configure SmartIO for an Oracle database as the Oracle DBA, make sure that the Oracle DBA user (`oracle` in our case) has the necessary `sudo` rights. For example, check the following entry in `/etc/sudoers`:

```
oracle ALL=(ALL) NOPASSWD: /opt/VRTS/bin/fscache3
```

And check with:

```
[oracle@ia-lab-r720-esxi03 ~]$ sudo -l
```

```
User oracle may run the following commands on this host:  
(ALL) NOPASSWD: /opt/VRTS/bin/fscache
```

The `root` user has already configured a cache area for VxFS:

```
[root@ia-lab-r720-esxi03 audit]# sfcache stat -l  
TYPE: VxFS  
NAME: vxfs_cachevol  
Cache Size: 100 GB  
Cache Utilization: 10.07 GB (10.07 %)  
File Systems Using Cache: 6  
Writeback Cache Use Limit: Unlimited  
Writeback Flush Timelag: 10 s  
  
Read Cache  
Hit Ratio Data Read Data Written Files Cached Files Pinned Data Pinned Writeback  
Hit Ratio Data Written  
Total:  
0.00 % 0 KB 0 KB 0 0 0 KB 0.00 % 0 KB  
/u02:  
0.00 % 0 KB 0 KB 0 0 0 KB 0.00 % 0 KB
```

With the correct setting of the Oracle environment variables `$ORACLE_SID` and `$ORACLE_HOME`, in our example:

```
[oracle@ia-lab-r720-esxi03 ~]$ echo $ORACLE_SID  
smartiodb  
[oracle@ia-lab-r720-esxi03 ~]$ echo $ORACLE_HOME  
/home/oracle/product/11gR2
```

The Oracle user can now activate SmartIO caching for the `smartiodb` database with the `sfcache` command. The example below applies the default settings for an OLTP workload. Note that the

³ The `sudo` requirement will be removed in version 6.2 of Symantec Storage Foundation.

database needs to be online when you run the `sfcache` command, as it queries the database catalog tables:

```
[oracle@ia-lab-r720-esxi03 ~]$ sfcache app cachearea=vxfs_cachevol oracle -S \
$ORACLE_SID -H $ORACLE_HOME -o setdefaults --type=OLTP
INFO: Oracle Instance smartiodb is running
INFO: Store DB details at /u02/oracle/oradata/smartiodb/.CACHE_INFO
INFO: Setting OLTP policies
INFO: Setting nocache mode to /u02

INFO: Setting nocache mode to /u02/oracle/oradata/smartiodb/redo03.log
INFO: Setting nocache mode to /u02/oracle/oradata/smartiodb/redo02.log
INFO: Setting nocache mode to /u02/oracle/oradata/smartiodb/redo01.log
INFO: Setting nocache mode to USE_DB_RECOVERY_FILE_DEST
INFO: Setting read mode to /u02/oracle/oradata/smartiodb/temp01.dbf
AWR snapid range min=21 max=43
INFO: Setting read mode to /u02/oracle/oradata/smartiodb/system01.dbf
INFO: Setting read mode to /u02/oracle/oradata/smartiodb/sysaux01.dbf
INFO: Setting read mode to /u02/oracle/oradata/smartiodb/undotbs01.dbf
INFO: Setting read mode to /u02/oracle/oradata/smartiodb/users01.dbf
```

As you can see from the `sfcache` output above, the Oracle template turns off caching for the REDOLOG log files and sets read caching for TEMPFILES and for the data files with the most frequent reads, based on the Automated Workload Repository (AWR) report. If the workload profile of the database changes substantially, it can be beneficial to reapply the template.

The Oracle user can check the SmartIO settings of certain database objects like data files, partitions or tablespaces of a database with the `list` option to the `sfcache app` command. For example, you can check the SmartIO settings for the database file `/u02/oracle/oradata/smartiodb/users01.dbf`:

```
[oracle@ia-lab-r720-esxi03 ~]$ sfcache app cachearea=vxfs_cachevol oracle -S \
$ORACLE_SID -H $ORACLE_HOME -o list \
--datafile=/u02/oracle/oradata/smartiodb/users01.dbf
INFO: Oracle Instance smartiodb is running4
INFO: Store DB details at /u02/oracle/oradata/smartiodb/.CACHE_INFO
FILENAME                                MODE          PINNED        CACHE_USED
-----                                -
/u02/oracle/oradata/smartiodb/users01.dbf  read         no            8 KB
```

You can also specify the tablespace name:

```
[oracle@ia-lab-r720-esxi03 ~]$ sfcache app cachearea=vxfs_cachevol oracle -S \
$ORACLE_SID -H $ORACLE_HOME -o list --tablespace=USERS
INFO: Oracle Instance smartiodb is running
INFO: Store DB details at /u02/oracle/oradata/smartiodb/.CACHE_INFO
FILENAME                                MODE          PINNED        CACHE_USED
-----                                -
/u02/oracle/oradata/smartiodb/users01.dbf  read         no            8 KB
```

The cache statistics can be viewed using the `sfcache` command with `-o stat`⁵:

⁴ The spelling mistake in the `sfcache` output will be fixed in version 6.2.

```
[oracle@ia-lab-r720-esxi03 ~]$ sfcache app cachearea=vxfs_cachevol oracle -S \
$ORACLE_SID -H $ORACLE_HOME -o stat --tablespace=USERS
INFO: Oracle Instance smartiodb is running
INFO: Store DB details at /u02/oracle/oradata/smartiodb/.CACHE_INFO
```

VxFS SmartIO Statistics

Cache Size = 100 GB
Cache Utilization = 80.5 MB (0.08 %)

Mount	Hit Ratio	Cached Files	Pinned Files	Read Bytes	Written Bytes	Data Pinned
/u02	0.12 %	4	0	13.55 MB	146.6 MB	0 KB

For advanced database administrators with knowledge of database activity and usage statistics, the `sfcache app` command also lets them set caching policies for a specified database object like a tablespace, partition, or a datafile. You can pin a specific database object to hold it in the SmartIO cache. (For a complete description, see the [Symantec Storage Foundation and High Availability Solutions SmartIO for Solid State Drives Solutions Guide](#).)

For example, if the database administrator decides to pin the USERS tablespace in the SmartIO cache, they can run:

```
[oracle@ia-lab-r720-03 ~]$ sfcache app cachearea=vxfs_cachevol oracle -S \
$ORACLE_SID -H $ORACLE_HOME -o pin --tablespace=USERS
INFO: Oracle Instance smartiodb is running
INFO: Store DB details at /u02/oracle/oradata/smartiodb/.CACHE_INFO
INFO: Setting pin policy to /u02/oracle/oradata/smartiodb/users01.dbf
```

These explicit settings of caching policies (which are different from the default setting of the respective template) are maintained in the directory `/u02/oracle/oradata/smartiodb/.CACHE_INFO` to ensure that explicit settings are not overwritten by subsequent `sfcache app oracle` commands:

```
[root@ia-lab-r720-03 ~]# cat \
/u02/oracle/oradata/smartiodb/.CACHE_INFO/ADMINPOLICY
/u02/oracle/oradata/smartiodb/users01.dbf:pin
```

You can check the statistics and settings for a given tablespace, datafile, or partition with:

```
[oracle@ia-lab-r720-03 ~]$ sfcache app cachearea=vxfs_cachevol oracle -S \
$ORACLE_SID -H $ORACLE_HOME -o list --tablespace=USERS
INFO: Oracle Instance smartiodb is running
INFO: Store DB details at /u02/oracle/oradata/smartiodb/.CACHE_INFO
FILENAME                                MODE      PINNED    CACHE_USED
-----                                -
/u02/oracle/oradata/smartiodb/users01.dbf  read      yes       24 KB

[oracle@ia-lab-r720-03 ~]$ sfcache app cachearea=vxfs_cachevol oracle -S \
$ORACLE_SID -H $ORACLE_HOME -o stat --tablespace=USERS
```

⁵ The `-o stat` option requires you to specify a tablespace or partition name, but the output is not limited to the specified tablespace or partition. This will be fixed in version 6.2, where no extra arguments will be required.

```
INFO: Oracle Instance smartiodb is running
INFO: Store DB details at /u02/oracle/oradata/smartiodb/.CACHE_INFO
```

VxFS SmartIO Statistics

```
-----
Cache Size = 100 GB
Cache Utilization = 123.5 MB ( 0.12 %)
```

Mount	Hit Ratio	Cached Files	Pinned Files	Read Bytes	Written Bytes	Data Pinned
-----	-----	-----	-----	-----	-----	-----
/u02	1.46 %	5	1	178.9 MB	565.0 MB	24 KB

The SmartIO plugin for Oracle also allows the DBA to temporarily disable or enable caching for a database. Temporarily disabling caching for a database can be helpful for certain database jobs like backup or data warehouse ETL (extract, transform, and load) operations. To disable caching for the smartiodb database while the database is online, run:

```
[oracle@ia-lab-r720-03 ~]$ sfcache app cachearea=vxfs_cachevol oracle -S \
$ORACLE_SID -H $ORACLE_HOME -o disable
INFO: Oracle Instance smartiodb is running
INFO: Store DB details at /u02/oracle/oradata/smartiodb/.CACHE_INFO
INFO: Setting disable mode to /u02
```

And to re-enable caching, run:

```
[oracle@ia-lab-r720-03 ~]$ sfcache app cachearea=vxfs_cachevol oracle -S \
$ORACLE_SID -H $ORACLE_HOME -o enable
INFO: Oracle Instance smartiodb is running
INFO: Store DB details at /u02/oracle/oradata/smartiodb/.CACHE_INFO
INFO: Setting enable mode to /u02
```

Note that disabling caching is a temporary setting and is not persistent across a reboot or failover of the database.

Clustered Oracle database and SmartIO

SmartIO settings like “pinned” for a file are stored in the metadata structure of the VxFS file system. Therefore, in a clustered configuration where /u02 is either a clustered file system (CFS) or in a failover configuration, these explicit settings are kept during a failover of the database.

In our example, we pin the USERS tablespace to the cache:

```
[oracle@ia-lab-r720-03 ~]$ sfcache app cachearea=vxfs_cachevol oracle -S \
$ORACLE_SID -H $ORACLE_HOME -o list --tablespace=USERS
INFO: Oracle Instance smartiodb is running
INFO: Store DB details at /u02/oracle/oradata/smartiodb/.CACHE_INFO
FILENAME                                MODE      PINNED      CACHE_USED
-----                                -
/u02/oracle/oradata/smartiodb/users01.dbf  read      yes         24 KB
```

When we fail over the database service group to the other node ia-lab-r720-08 in our cluster:

```
[root@ia-lab-r720-03 ~]# hagrpr -switch oracle -to ia-lab-r720-08
```

We can confirm that the USERS tablespace is still pinned to the cache after the failover:

```

[oracle@ia-lab-r720-08 ~]$ sfcache app cachearea=vxfs_cachevol oracle -S \
$ORACLE_SID -H $ORACLE_HOME -o list --tablespace=USERS
INFO: Oracle Instance smartiodb is running
INFO: Store DB details at /u02/oracle/oradata/smartiodb/.CACHE_INFO
FILENAME                                MODE          PINNED        CACHE_USED
-----                                -
/u02/oracle/oradata/smartiodb/users01.dbf  read          yes           24 KB

```

As mentioned above, disabling caching for the database is a temporary operation, so after the failover of a database with caching disabled, it has caching enabled again on the other node in the cluster:

```

[oracle@ia-lab-r720-08 ~]$ sfcache app cachearea=vxfs_cachevol oracle -S \
$ORACLE_SID -H $ORACLE_HOME -o disable
INFO: Oracle Instance smartiodb is running
INFO: Store DB details at /u02/oracle/oradata/smartiodb/.CACHE_INFO
INFO: Setting disable mode to /u02

```

```

[oracle@ia-lab-r720-08 ~]$ sfcache app cachearea=vxfs_cachevol oracle -S \
$ORACLE_SID -H $ORACLE_HOME -o list --tablespace=USERS
INFO: Oracle Instance smartiodb is running
INFO: Store DB details at /u02/oracle/oradata/smartiodb/.CACHE_INFO
FILENAME                                MODE          PINNED        CACHE_USED
-----                                -
/u02/oracle/oradata/smartiodb/users01.dbf  nocache       no            0 KB

```

```

[root@ia-lab-r720-03 ~]# hagrps -switch oracle -to ia-lab-r720-esxi03

```

```

[oracle@ia-lab-r720-03 ~]$ sfcache app cachearea=vxfs_cachevol oracle -S \
$ORACLE_SID -H $ORACLE_HOME -o list --tablespace=USERS
INFO: Oracle Instance smartiodb is running
INFO: Store DB details at /u02/oracle/oradata/smartiodb/.CACHE_INFO
FILENAME                                MODE          PINNED        CACHE_USED
-----                                -
/u02/oracle/oradata/smartiodb/users01.dbf  read          yes           24 KB

```

Use the `sfcache app ... -o set -cachemode=nocache` to disable caching persistently.

In a clustered environment, the secondary node does not necessarily need to have a cache area configured as well (although it is recommended). The database still starts on nodes without a cache area, but performance is slower.

To demonstrate this scenario, let's remove the cache area on `ia-lab-r720-08` while the database runs on `ia-lab-r720-03`:

```

[root@ia-lab-r720-08 ~]# sfcache offline vxfs_cachevol
[root@ia-lab-r720-08 ~]# sfcache list
NAME                                TYPE      SIZE          ASSOC-TYPE  STATE
DEVICE
vxfs_cachevol                      VxFS     100.00g      AUTO        OFFLINE
fusionio0_0
[root@ia-lab-r720-08 ~]# sfcache delete vxfs_cachevol
[root@ia-lab-r720-08 ~]# sfcache list
NAME                                TYPE      SIZE          ASSOC-TYPE  STATE
DEVICE

```

And switch the VCS service group controlling the `smartiodb` database to `ia-lab-r720-08`:

```

[root@ia-lab-r720-03 ~]# hagrps -switch oracle -to ia-lab-r720-08
[root@ia-lab-r720-08 ~]# hagrps -state oracle

```

#Group	Attribute	System	Value
oracle	State	ia-lab-r720-03	OFFLINE
oracle	State	ia-lab-r720-08	ONLINE

We can see that the database starts but caching is not enabled obviously:

```
[oracle@ia-lab-r720-08 ~]$ sfcache app cachearea=vxfs_cachevol oracle -S \
$ORACLE_SID -H $ORACLE_HOME -o list --tablespace=USERS
SFCache ERROR V-5-2-6278 Invalid type specified to create cache area
```

SmartIO Caching in Virtualized Environments

In a Linux virtualized environment, namely a Kernel-based Virtual Machine (KVM) or Red Hat Enterprise Virtualization (RHEV), the benefits of SmartIO technology can also be made available to the guest operating systems and the applications running on them. You can either:

- Install Symantec Storage Foundation with SmartIO inside the guest OS.
- Install Symantec Storage Foundation on the hosts (hypervisors) and provision either a VxVM volume or an image file on a VxFS file system to the guests.

In both cases, you can use SmartIO to cache data onto a local SSD attached to the host.

In a VMWare virtualized environment with Linux running in the guests, you can use the SmartPool feature to improve I/O performance by using SSD devices local to the ESXi hosts as cache. See [Improving I/O performance using SmartPool in Symantec™ Storage Foundation and High Availability Solutions 6.2 Virtualization Guide - Linux on ESXi](#)

For a complete list of supported SmartIO configurations in Linux virtualized environments, see [Symantec™ Storage Foundation and High Availability Solutions 6.2 Virtualization Guide - Linux](#).

SmartIO caching on the host in KVM/RHEV virtualized environments

To demonstrate how SmartIO caching on KVM works, we configure a KVM virtual machine, `vm2` (running RHEL 6.5), on the RHEL 6.4 server `ia-lab-r720-08` with Symantec Storage Foundation and High Availability 6.2 installed.

There are multiple ways of provisioning storage on KVM virtual machines. In the example below, we use an image file created on a VxFS file system as storage backend for the virtual machine. The image file will be mapped to the virtual machine via the para-virtualized virtio drivers. For other options, see the KVM documentation or the [Symantec™ Storage Foundation and High Availability Solutions 6.2 Virtualization Guide - Linux](#).

In the VxFS file system mounted on `/test1` on the host `ia-lab-r720-08`, create a pre-allocated file `vm2_vol1` to be used as storage backend:

```
[root@ia-lab-r720-08 ~]# dd if=/dev/zero of=/test1/vm2_vol1 bs=1M count=11240
11240+0 records in
11240+0 records out
11785994240 bytes (12 GB) copied, 27.3084 s, 432 MB/s
```

Next, map the image file to the virtual machine `vm2` as virtio device `vdb`:

```
root@ia-lab-r720-08 ~]# virsh attach-disk vm2 /test1/vm2_vol1 vdb --cache none
Disk attached successfully

[root@ia-lab-r720-08 ~]# virsh dumpxml vm2 > /tmp/vm2.xml
[root@ia-lab-r720-08 ~]# virsh define /tmp/vm2.xml
Domain vm2 defined from /tmp/vm2.xml
```

The section defining the `vdb` disk device in the virtual machine's XML configuration file looks like this:

```
<disk type='file' device='disk'>
  <driver name='qemu' type='raw' cache='none'/>
  <source file='/test1/vm2_vol1'/>
  <target dev='vdb' bus='virtio'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x0a'
function='0x0'/>
</disk>
```

and the disk device `vdb` shows up in the virtual machine:

```
[root@vm2 ~]# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda                                  252:0    0  10G  0 disk
├─vda1                               252:1    0 500M  0 part /boot
└─vda2                               252:2    0  9.5G  0 part
   ├─VolGroup-lv_root (dm-0) 253:0    0  8.5G  0 lvm  /
   └─VolGroup-lv_swap (dm-1) 253:1    0   1G  0 lvm  [SWAP]
vdb                                252:16  0  11G  0 disk
sr0                                  11:0    1 1024M  0 rom
```

For detail on how to configure Symantec Storage Foundation as backend storage for RHEV virtual machines, see Chapter 6 in the [Symantec™ Storage Foundation and High Availability Solutions 6.2 Virtualization Guide - Linux](#).

Inside the virtual machine `vm2`, we create a logical volume manager (LVM) volume with an ext4 file system on the disk device `vdb` and mount it on `/test1`:

```
[root@vm2 ~]# pvcreate /dev/vdb
Physical volume "/dev/vdb" successfully created
[root@vm2 ~]# vgcreate testvg /dev/vdb
Volume group "testvg" successfully created
[root@vm2 ~]# lvcreate -n testlv -l 2809 testvg
Logical volume "testlv" created
[root@vm2 ~]# mkfs -t ext4 /dev/testvg/testlv
mke2fs 1.41.12 (17-May-2010)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
719488 inodes, 2876416 blocks 143820 blocks (5.00%) reserved for the super
user
First data block=0
Maximum filesystem blocks=2948595712
88 block groups 32768 blocks per group, 32768 fragments per group
8176 inodes per group
Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208
Writing inode tables: done
Creating journal (32768 blocks): done
```

Writing superblocks and filesystem accounting information: done
 This filesystem will be automatically checked every 37 mounts or 180 days,
 whichever comes first. Use tune2fs -c or -i to override.

```
[root@vm2 ~]#
[root@vm2 ~]# mount -t ext4 /dev/testvg/testlv /test1
[root@vm2 ~]#
[root@vm2 ~]# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda                                  252:0    0   10G  0 disk
├─vda1                              252:1    0   500M  0 part /boot
├─vda2                              252:2    0    9.5G  0 part
│   └─VolGroup-lv_root (dm-0) 253:0    0    8.5G  0 lvm /
│       └─VolGroup-lv_swap (dm-1) 253:1    0     1G  0 lvm [SWAP]
└─vdb                                252:16   0    11G  0 disk
    └─testvg-testlv (dm-2) 253:2    0    11G  0 lvm /test1
sr0                                  11:0    1 1024M  0 rom
```

On the KVM host, the image file mapped as /dev/vdb to vm2 is enabled for write-back caching:

```
[root@ia-lab-r720-08 ~]# mount -v | grep /test1
/dev/vx/dsk/test2dg/voll on /test1 type vxfs
(rw,nosuid,delaylog,largefiles,noatime,ioerror=mwdisable,_netdev,smartiomode=writeback)
[root@ia-lab-r720-esxi08 ~]#
[root@ia-lab-r720-08 ~]# sfcache list /test1/vm2_voll
/test1/vm2_voll:
READ CACHE      WRITEBACK      MODE           PINNED
      0 KB           0 KB      writeback     no
```

To see the how SmartIO on the host caches I/Os from the virtual machine, let's flush and purge the cache content and reset the cache statistics, the caching disk group VxVM statistics (sfcache_defaultdg), and the disk group hosting the /test1 file system on the host (test2dg):

```
[root@ia-lab-r720-esxi08 test1]# sfcache flush /test1
[root@ia-lab-r720-esxi08 test1]# sfcache purge /test1
[root@ia-lab-r720-esxi08 test1]# sfcache stat -r /test1
[root@ia-lab-r720-esxi08 test1]# vxstat -g sfcache_defaultdg -r
[root@ia-lab-r720-esxi08 test1]# vxstat -g test2dg -r
```

Now, let's create a 100 MB file in the /test1 file system (backed by /test1/vm2_voll on the host) in the virtual machine:

```
[root@vm2 test1]# dd if=/dev/urandom of=/test1/file1 bs=1M count=100 \
oflag=direct
100+0 records in
100+0 records out
104857600 bytes (105 MB) copied, 32.533 s, 3.2 MB/s
[root@vm2 test1]#
```

Looking on the host side again, we can see that the direct write I/Os from the virtual machine have been cached:

```
[root@ia-lab-r720-esxi08 test1]# sfcache list /test1/vm2_voll
/test1/vm2_voll:
READ CACHE      WRITEBACK      MODE           PINNED
      12 KB       100.0 MB      writeback     no
[root@ia-lab-r720-esxi08 test1]# vxstat -g sfcache_defaultdg
OPERATIONS      BLOCKS          AVG
TIME (ms)
TYP NAME          READ    WRITE    READ    WRITE    READ    WRITE
```

```
vol sfcachearea_1          1      12843      16      307938      0.00      0.06
```

After some time (`writeback_interval`), the content of the write-back cache is flushed to disk:

```
[root@ia-lab-r720-08 ~]# sfcache list /test1/vm2_vol1
/test1/vm2_vol1:
READ CACHE      WRITEBACK      MODE      PINNED
    12 KB          0 KB    writeback    no
```

Before looking into the caching behavior when reading the same file, we reset the statistics counters on the host again:

```
[root@ia-lab-r720-esxi08 test1]# sfcache stat -r /test1
[root@ia-lab-r720-esxi08 test1]# vxstat -g test2dg -r
[root@ia-lab-r720-esxi08 test1]# vxstat -g sfcache_defaultdg -r
```

When reading the file `file1` for the first time after creation in the virtual machine

```
[root@vm2 test1]# dd if=/test1/file1 of=/dev/null iflag=direct bs=8k
12800+0 records in
12800+0 records out
104857600 bytes (105 MB) copied, 9.75483 s, 10.7 MB/s
```

its content is written into the SmartIO read cache on the host:

```
[root@ia-lab-r720-esxi08 test1]# sfcache stat -l /test1
    Cache Size:      100 GB
Cache Utilization:  612.0 MB ( 0.60 %)

Read Cache
Hit Ratio      Data Read      Data Written      Files Cached      Files Pinned      Data Pinned
/test1:
    0.00 %          0 KB          100 MB              1              0              0 KB
```

The VxVM statistics show the same amount of reads from the data volume as writes into the cache volume:

```
[root@ia-lab-r720-esxi08 test1]# vxstat -g test2dg
OPERATIONS          BLOCKS          AVG
TIME (ms)
TYP NAME          READ      WRITE      READ      WRITE      READ      WRITE
vol vol1          25600     21      204800     162      0.10     0.38
vol vol2           0         0         0         0         0.00     0.00
vol vol3           0         0         0         0         0.00     0.00
[root@ia-lab-r720-esxi08 test1]# vxstat -g sfcache_defaultdg
OPERATIONS          BLOCKS          AVG
TIME (ms)
TYP NAME          READ      WRITE      READ      WRITE      READ      WRITE
vol sfcachearea_1  1       26041       16     256590      0.00     0.06
```

With the file content in the read cache, a second read of the complete file in the virtual machine (statistics counters were reset on the host again) is completely served from the read cache:

```
[root@vm2 test1]# dd if=/test1/file1 of=/dev/null iflag=direct bs=8k
12800+0 records in
12800+0 records out
104857600 bytes (105 MB) copied, 5.22446 s, 20.1 MB/s
[root@ia-lab-r720-esxi08 test1]# sfcache stat /test1
    Cache Size:      100 GB
Cache Utilization:  612.0 MB ( 0.60 %)
```

```

Read Cache
Hit Ratio      Data Read      Data Written

/test1:
100.00 %      100 MB          0 KB

```

The VxVM statistics also show no reads from the data volume, only from cache:

```

[root@ia-lab-r720-esxi08 test1]# vxstat -g test2dg
OPERATIONS          BLOCKS          AVG
TIME (ms)
TYP NAME           READ           WRITE          READ           WRITE          READ           WRITE
vol vol1           1              13             16             106           15.60          0.98
vol vol2           0              0              0              0             0.00           0.00
vol vol3           0              0              0              0             0.00           0.00
[root@ia-lab-r720-esxi08 test1]# vxstat -g sfcache_defaultdg
OPERATIONS          BLOCKS          AVG
TIME (ms)
TYP NAME           READ           WRITE          READ           WRITE          READ           WRITE
vol sfcachearea_1 25600          1              204800         8             0.10           0.00

```

The image file mapped to the virtual machine can also be explicitly loaded into the cache (`sfcache -load`). By running an IOzone benchmark inside the virtual machine (`iozone -i 0 -i 2 -i 4 -I -o -t 10 -r 8k -s 1000m`) on the file system `/test1` backed by the image file `/test1/vm2_vol1`, we can compare the effects of the different cache options on I/O throughput.

The following table compares the average throughputs of the IOzone runs (in kB/s) per process for the four workloads:

Caching Mode	Image file pre-loaded	Initial Write	Re-Write	Random Read	Random Write
None	No	681.73	2574.02	5982.10	1918.62
Read	No	671.94	2563.37	3935.88	1703.08
Write-back	No	627.96	223.68	10173.74	1716.49
Read	Yes	588.04	2249.34	13482.00	1754.34
Write-back	Yes	581.89	2263.34	13854.63	1793.21

Managing SmartIO with Veritas Operations Manager

Starting with version 6.1, Veritas Operations Manager enables basic management of SmartIO capabilities with Veritas Operations Manager’s web-based console.

Creating cache areas with Veritas Operations Manager

The Veritas Operations Manager Management Server console lets you create cache areas using the available SSD devices. The cache area name is system-generated and cannot be modified.

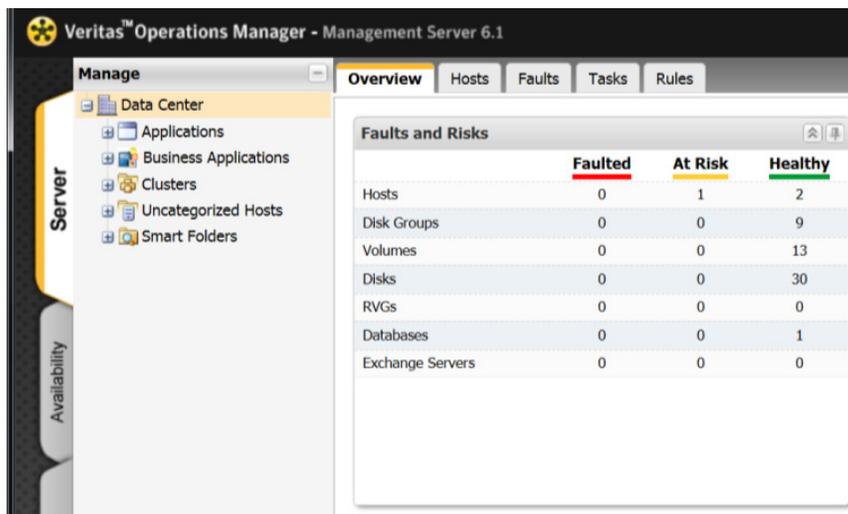
To perform this task, your user group must be assigned the `Admin` role on the host or the Server perspective. The permission on the host may be explicitly assigned or inherited from a parent organization.

To create cache areas with Veritas Operations Manager

- 1 Display the **Management Server** console and select the **Server** perspective:



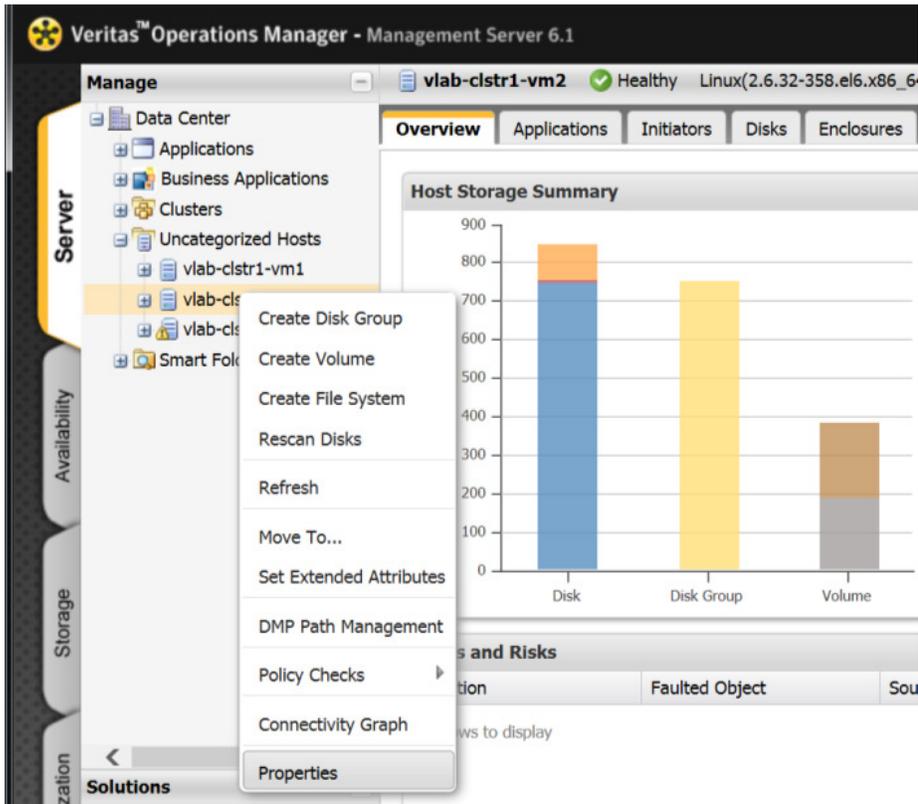
- 2 On the **Server** screen, in the left pane, expand **Manage**:



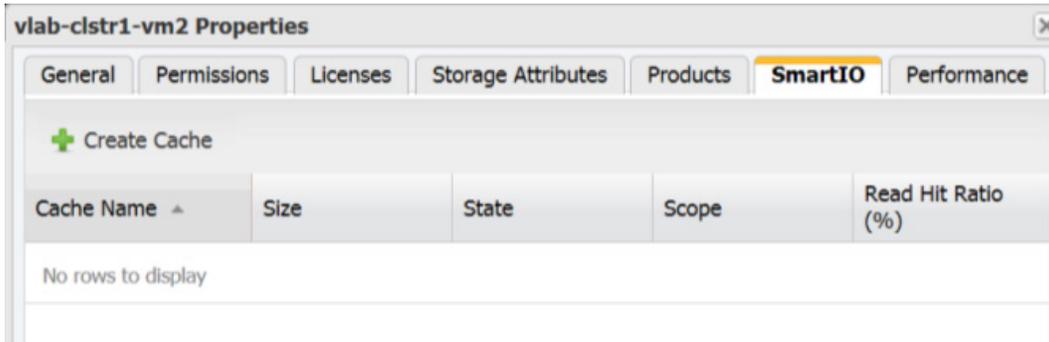
- 3 Expand the Organization or Uncategorized Hosts icons to locate and select the host (vlab-clstr1-vm2):



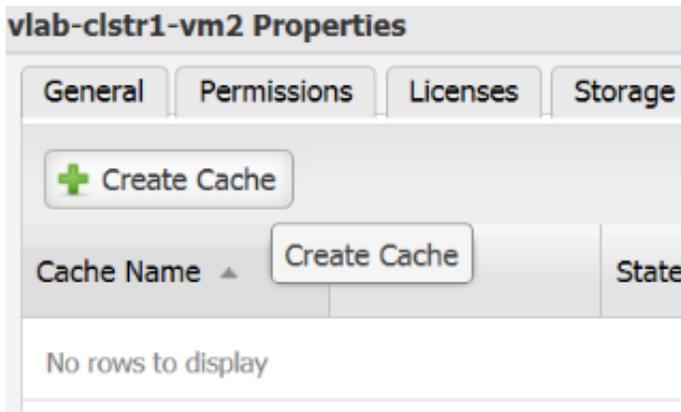
- 4 Right-click on the host and select **Properties**:



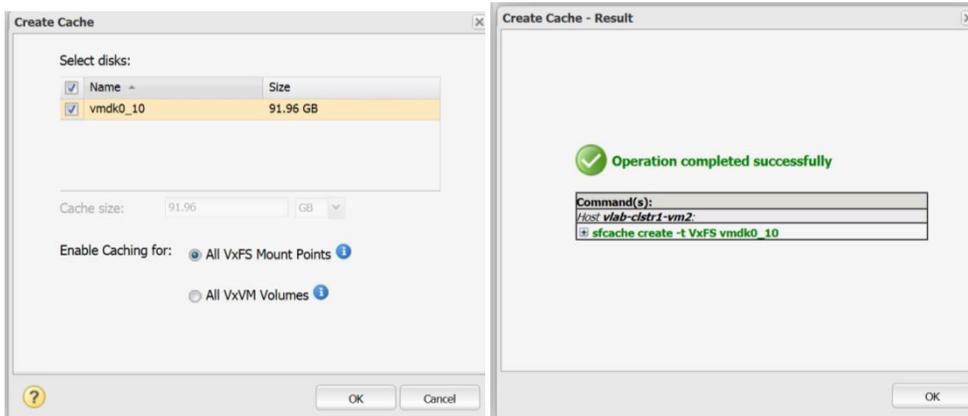
- 5 On the **Properties** window, click the **SmartIO** tab:



- 6 Click **Create Cache**:



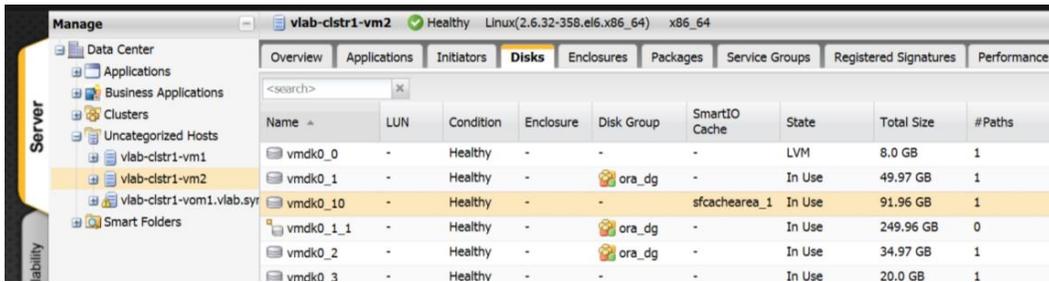
- 7 In the **Create Cache** panel, enter the details, and click **OK**:



The system automatically chooses the cache area name.



We now see that SmartIO caching has been enabled on vmdk0_10:

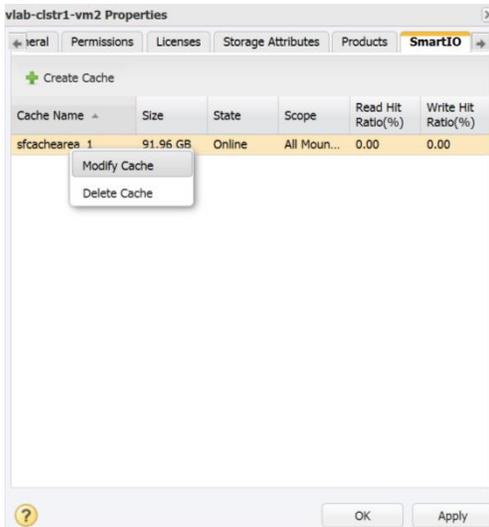


Name	LUN	Condition	Enclosure	Disk Group	SmartIO Cache	State	Total Size	#Paths
vmdk0_0	-	Healthy	-	-	-	LVM	8.0 GB	1
vmdk0_1	-	Healthy	-	ora_dg	-	In Use	49.97 GB	1
vmdk0_10	-	Healthy	-	-	sfcachearea_1	In Use	91.96 GB	1
vmdk0_1_1	-	Healthy	-	ora_dg	-	In Use	249.96 GB	0
vmdk0_2	-	Healthy	-	ora_dg	-	In Use	34.97 GB	1
vmdk0_3	-	Healthy	-	-	-	In Use	20.0 GB	1

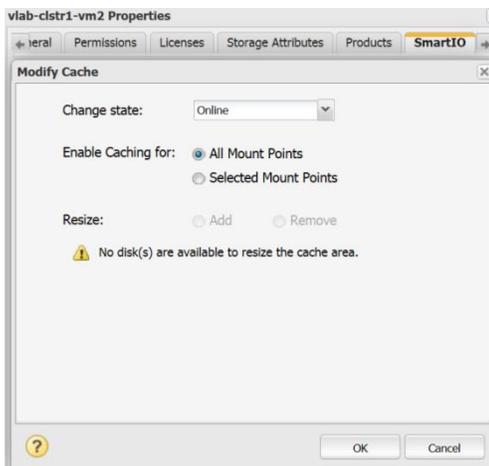
Modifying and deleting cache areas with Veritas Operations Manager

To modify a cache area

- 1 On the **Properties** screen, **SmartIO** tab, select **Modify Cache** from the drop-down list:

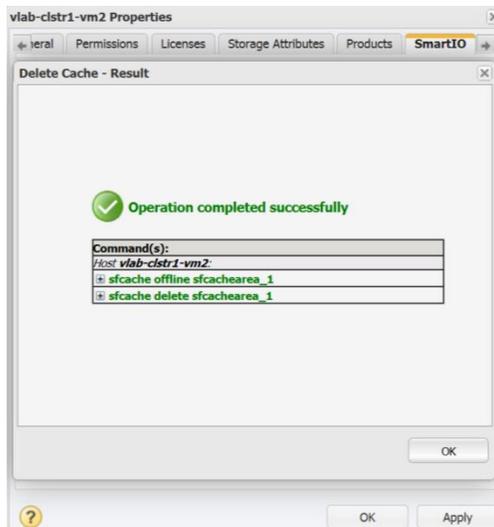
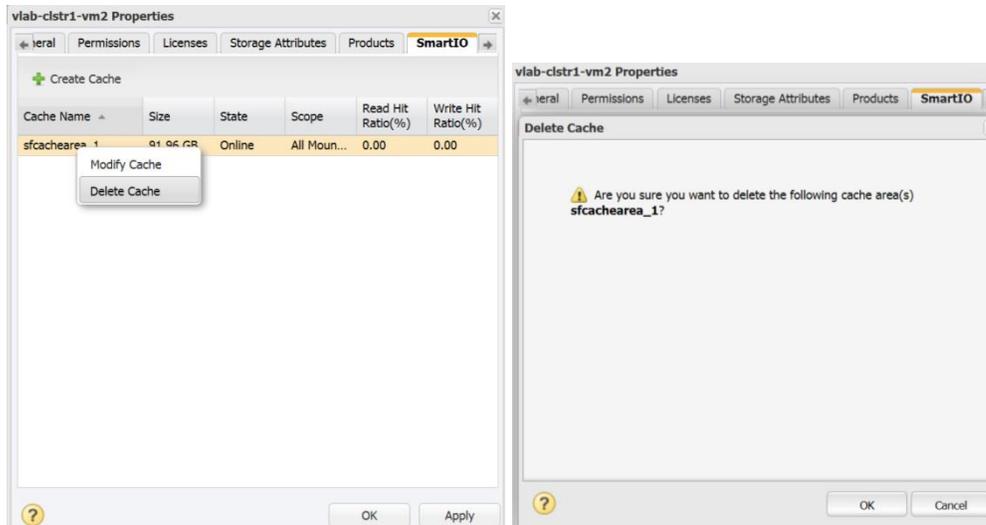


- 2 On the **Modify Cache** screen, we can change the state between online and offline, and enable caching for all mount points or selected mount points:

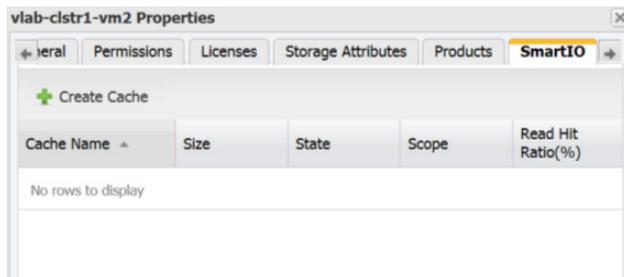


To delete a cache area

- 1 On the **Properties** screen, **SmartIO** tab, right-click on the cache name and select **Delete Cache** from the drop-down list:



We can see that the cache area has been removed or deleted:



Overview	Applications	Initiators	Disks	Enclosures	Packages	Service Groups	Registered Signatures	Performance
<input type="text" value="<search>"/>								
Name	LUN	Condition	Enclosure	Disk Group	SmartIO Cache	State	Total Size	#Paths
vmdk_0	-	Healthy	-	-	-	LVM	8.0 GB	1
vmdk_1	-	Healthy	-	ora_dg	-	In Use	49.97 GB	1
vmdk_10	-	Healthy	-	-	-	Free (In...	91.96 GB	1

Enabling or disabling SmartIO caching with Veritas Operations Manager

By default, the scope of caching is set to all VxFS mount points or all VxVM volumes. You can disable caching on a specific mount point or volume.

If the caching scope is set to a selected mount point or volume, caching is not enabled on any mount point or volume by default. You must explicitly enable caching on the required mount point or volume.

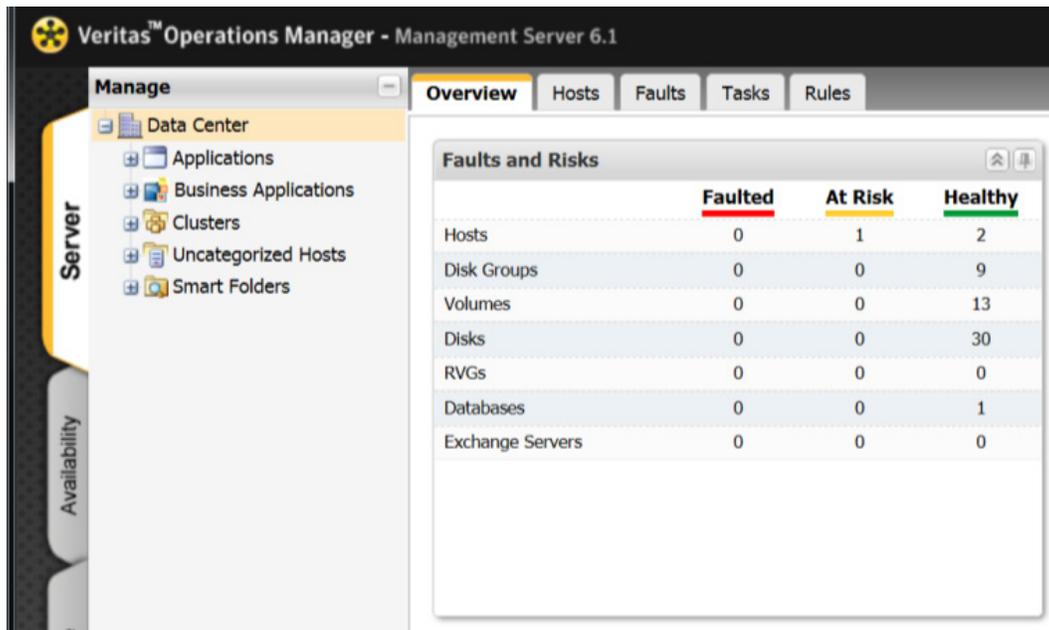
To perform this task, your user group must be assigned the `Admin` role on the host or the `Server` perspective. The permission on the host may be explicitly assigned or inherited from a parent organization.

To enable or disable SmartIO caching with Veritas Operations Manager

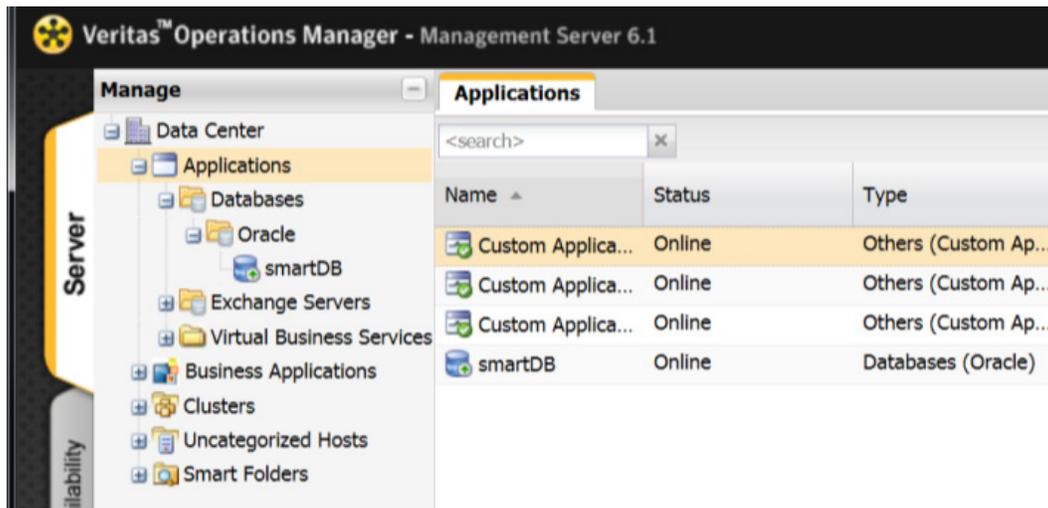
- 1 In the **Management Server** console, and select the **Server** perspective:



- On the **Server** screen, in the left pane, expand **Manage**:

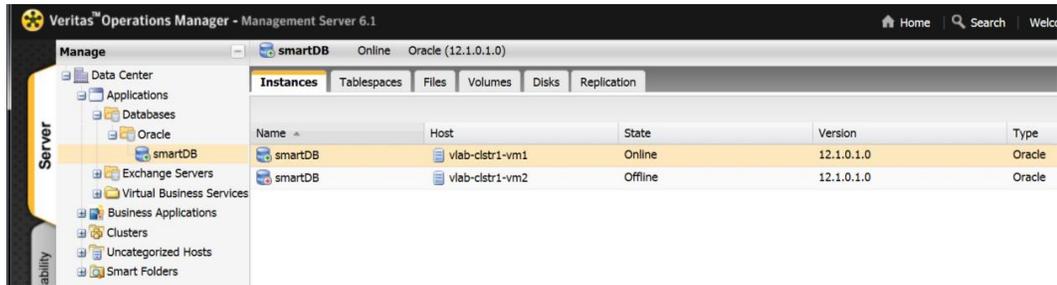


- Expand the Organization, Applications, or Uncategorized Hosts icons:

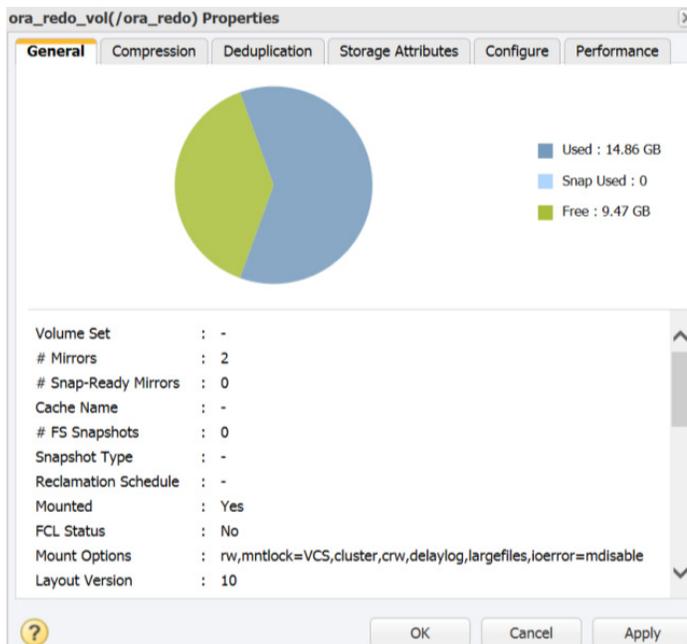
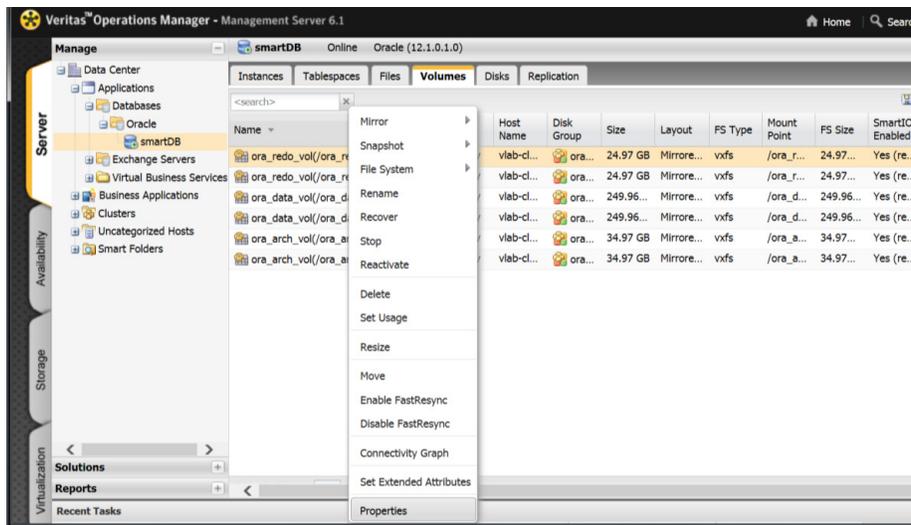


4 Do one of the following:

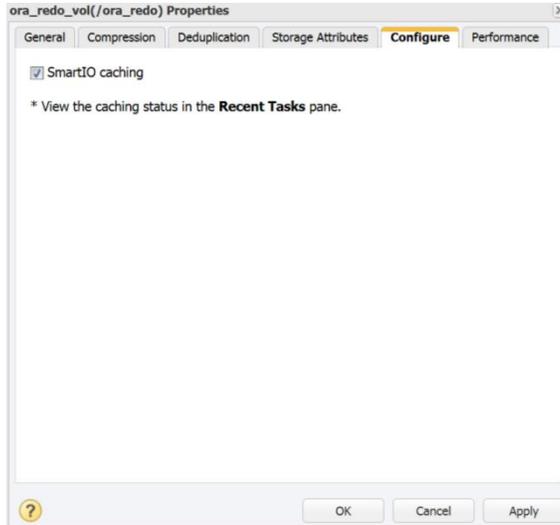
- Expand **Databases** to locate the database:



- Click on the **Volumes** tab and right-click a volume and select **Properties**:



- 5 Click the **Configure** tab and do one of the following:
 - To enable SmartIO caching, select the **SmartIO caching** check box.
 - To disable SmartIO caching, clear the **SmartIO caching** check box.



- 6 Click **Apply** and click **OK**.

Viewing cache details with Veritas Operations Manager

You can use the Management Server console to view the cache details on a host. You can view details such as the cache name, size, state, as well as the following:

- **Scope:** Displays the scope of the cache such as all volumes or all mount points.
- **Read Hit Ratio (%):** Displays the cache read hit ratio as a percentage.
- **Write Hit Ratio (%):** Displays the cache write hit ratio as a percentage. This applies only when write-back caching is enabled for VxFS mount point.

In this view, you can create, modify, and delete the cache area.

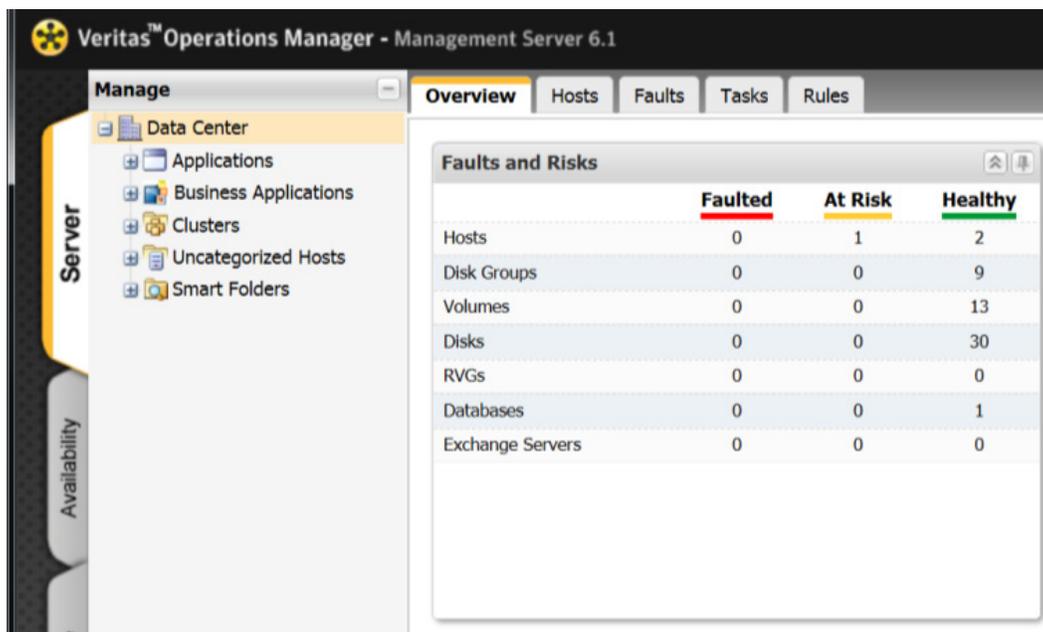
You can view this information, if your user group has at least a Guest role assigned on the perspective or the Organization.

To view the cache details with Veritas Operations Manager

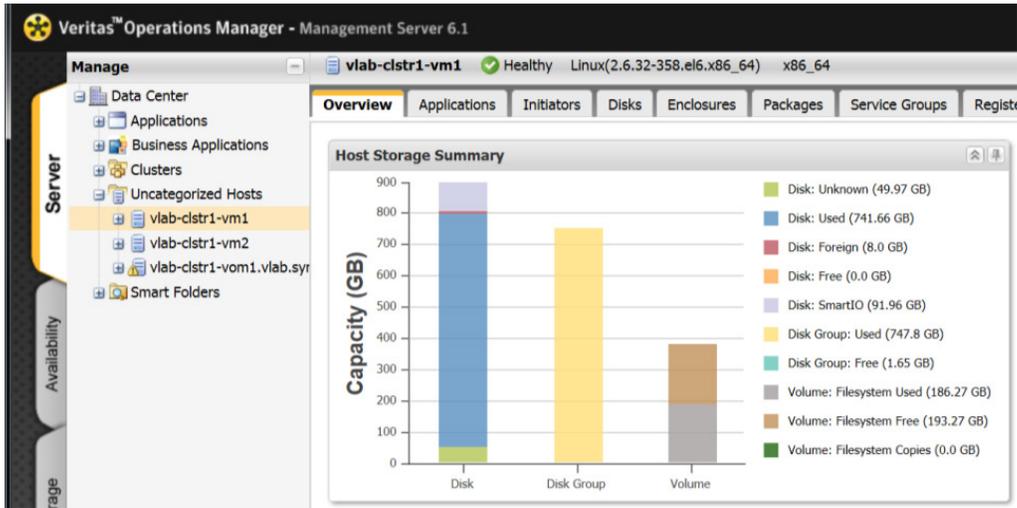
- 1 In the Management Server console, go to the **Server** perspective::



- 2 On the **Server** screen, in the left pane, expand **Manage**:



- Expand the Organization or Uncategorized Hosts icons to locate the host:



- Right-click the host and select **Properties**.

vlab-clstr1-vm1 Properties

General Permissions Licenses Storage Attributes Products SmartIO Performance

CVM Master : Yes
 Cluster : vlab-clstr1
 Site : -
 Build Version : 6.1.0.0-596
 OS Release : v6.4 Red Hat
 Platform : Linux
 Family : Red Hat Enterprise Linux Server release 6.4 (Santiago)
 Architecture : x86_64
 SF Version : 6.1
 VCS Version : 6.1.0.000
 MH Version : 6.1.0.0
 OS Version : 2.6.32-358.el6.x86_64
 IP : 192.168.110.221, 192.168.210.221, fd4b:454e:205a:410:250:56ff:fe88:2ce, 192.168.22
 Is Virtual : Yes

- Click the **SmartIO** tab.

vlab-clstr1-vm1 Properties

General Permissions Licenses Storage Attributes Products **SmartIO** Performance

+ Create Cache

Cache Name	Size	State	Scope	Read Hit Ratio (%)	Write Hit Ratio (%)
sfcachearea_1	91.96 GB	Online	All Mount Po...	91.03	0.00

Here we can see **Read Hit Ratio (%)** and **Write Hit Ratio (%)** for the cache area.

Appendix A: SmartIO terminology

This document uses the following terminology:

Term	Definition
Cache area	A dynamic storage region reserved for caching. For the current release, there can be only one cache area of each type on a system.
Caching modes	<p>The caching mode used for SmartIO. SmartIO has three caching modes; however, the caching mode can be either at the volume level or file system level. The caching modes are:</p> <ul style="list-style-type: none">• Volume level read cache: This block based read cache can be used for applications that store their data on raw volumes. For this release, this is not supported for shared volumes.• File system level read cache: This is a file-level read cache for applications that store their data in files on a file system. This configuration supports very granular data caching, and you can also use it in a clustered file system (CFS) environment. Currently, there is no metadata caching support for file system level caching.• Write-back cache: This is a super set of file system read cache, and allows writes to be buffered on SSD devices. Using this caching mode, the application can improve the performance of latency sensitive I/Os; for example, Oracle redo log writes.
Data object	The object for which caching is enabled, either the Veritas file system or a Veritas raw volume.
Persistent read cache	A read cache in which cache data survives a system power cycle. Also known as warm cache.
Read cache	A caching solution designed to accelerate read I/Os. The solution caches the reads on a system and any re-reads are served from the cache.
Volatile read cache	A read cache in which cached data is lost after a system power cycle.
Write-back cache	At first, data is written only to the cache. Data is not written to the backing store until the cache blocks containing the data are about to be changed or replaced by new content.
Write-through cache	The data is written synchronously both to the cache and to the backing store.

Appendix B: sfcache command reference

The `sfcache` utility includes all the commands you need to administer SmartIO. The following table summarizes basic SmartIO tasks and the command syntax to use. For detailed information, see the [sfcache\(1M\) manual page](#).

Task	Command
Creating or deleting the cache	
Create a cache area and specify the disk access name for the caching device.	<code>sfcache create [-t {VxFS VxVM}] [size] {daname...} [cacheline_size=size] [--auto --noauto] [cachearea_name]</code>
Create a cache area using the specified volume.	<code>sfcache create [-t {VxFS VxVM}] [cacheline_size=size] [--auto --noauto] dg/vol</code>
Delete a cache area.	<code>sfcache delete [cachearea_name dg/volname]</code>
Configuring the cache	
Specify the caching mode. (VxFS only)	<code>sfcache set [-r] mode={nocache read writeback} {file1 dir1} [file2 dir2...]</code>
Set attributes for the cache area.	<code>sfcache set [--auto --noauto] [memsz=size] [cachearea_name dg/volname]</code>
Set the cache area attribute to auto or noauto .	<code>sfcache set cachearea_name {-a -r}</code>
Control the amount of time write-back data remains unflushed. (VxFS only)	<code>sfcache set writeback_interval=interval</code>
Specify the maximum size for the amount of dirty data to be held in the cache. (VxFS only)	<code>sfcache set writeback_size=size</code>
Starting and stopping the cache	
Start caching for the specified volume.	<code>sfcache enable [--auto] [--read] dg/vol</code>
Start caching for the specified VxFS mount point. (VxFS only)	<code>sfcache enable mount_point</code>
Disable caching for the specified data object.	<code>sfcache disable [-o purge] {mount_point dg/volname}</code>
Stop SmartIO from using a cache area. VxVM only: use the flushmeta option to create	<code>sfcache offline [--flushmeta] {cachearea_name dg/volname}</code>

a warm cache for a planned shutdown.	
Explicitly make a cache area available.	<code>sfcache online [cachearea_name dg/volname </code>
Displaying cache information	
Display the cached volumes and their cache usage. (VxVM only)	<code>sfcache list [-l] [cachearea_name dg/volname]</code>
Display the cached file system and its cache usage. (VxFS only)	<code>sfcache [-r] [mount_point file dir]</code>
Display cache statistics, including cache hit rate, misses, average read and write latencies. (VxFS only)	<code>sfcache stat [-l] [-r] mount_point</code>
Display cache statistics, including cache hit rate, misses, average read and write latencies. (VxVM only)	<code>sfcache stat [-l] [-r] [-i time] [cachearea_name dg/volname]</code>
Display the amount of free space in the devices that are already provisioned for caching.	<code>sfcache maxsize [aname ...]</code>
Resizing the cache	
Resize the specified area. Specify additional devices if the existing devices do not have enough free space to grow the specified cache area to the required size. (Only for cache areas created directly on SSD)	<code>sfcache resize [aname...] {newsize maxsize} cachearea_name</code>
Working with files, directories, data objects, and devices	
Load the specified file into the cache area.	<code>sfcache load [-r] [-o sync] {file1 dir1} [file2 dir2...]</code>
Mark a file or directory to be held in cache until the file or directory is deleted, truncated, or unpinned. Only	<code>sfcache pin [-o load] [-r] {file1 dir1} [file2 dir2...]</code>

applies to VxFS caching.	
Remove the file or directory from the pinned state. Only applies to VxFS caching.	<code>sfcache unpin [-r] {file1 dir1} [file2 dir2...]</code>
Remove the cached contents for the specified file system. This command frees up dead space in the cache. Only applies to VxFS caching.	<code>sfcache purge {mount_point fsuuid}</code>
Restore read or write access to files that are missing write-back data. Use this command for recovery. (VxFS only)	<code>sfcache restore-access [-r] {mount_point file dir}</code>
Remove the device or devices from use for caching.	<code>sfcache rmdev daname[...]</code>
Control caching for the specified VxVM data object. (VxVM only)	<code>sfcache set [--pause --resume] dg/volname</code>
Control caching for the specified VxFS data object. (VxFS only)	<code>sfcache set [-r] mode={nocache read writeback} {file1 dir1} [file2 dir2...]</code>
Working with application templates	
Set the default SmartIO caching policies for an Oracle database (run as Oracle user)	<code>sfcache app cachearea=cachearea_name oracle -S ORACLE_SID -H ORACLE_HOME -o setdefaults --type={olap oltp}</code>
Temporarily disable caching for a database (run as Oracle user)	<code>sfcache app cachearea=cachearea_name oracle -S ORACLE_SID -H ORACLE_HOME -o disable</code>
Enable caching for a database (run as Oracle user)	<code>sfcache app cachearea=cachearea_name oracle -S ORACLE_SID -H ORACLE_HOME -o enable</code>

Appendix C: sfcache statistics reference

The following table describes each field displayed in `sfcache` command output.

Field	Description
HIT RATIO (VxVM cache)	Percentage of total I/Os that are satisfied from the cache. Displayed for reads and writes.
ART(Hit)ms (VxVM cache)	Average response time for I/Os that are satisfied from the cache. Displayed for reads (RD) and writes (WR).
ART(Miss)ms (VxVM cache)	Average response time for I/Os that are not satisfied from the cache. Displayed for reads (RD) and writes (WR).

BYTES (VxVM cache)	Total size of I/Os for reads (RD) and writes (WR).
NAME (VxVM cache)	Name of the cache area.
TYPE (VxVM cache)	Whether the cache area is VxVM or VxFS.
%CACHE (VxVM cache)	Percentage of the cache area that is currently used for data for all data objects.
Cache Size (VxFS cache)	Size of the cache area.
Cache Utilization (VxFS cache)	Percentage of the cache area that is currently used for data.
File Systems Using Cache (VxFS cache)	Number of file systems using the cache.
Writeback Cache Use Limit (VxFS cache)	Maximum for the space used for dirty data per node. By default, there is no maximum. If you configure a maximum, you must allow at least 512 MB for each file system or cluster file system. For a cluster file system, the space required for cache reflection is not included in the maximum.
Writeback Flush Timelag	Interval between when the data is written to the cache and when it is flushed to the disk. If the Writeback Flush Timelag is small, such as 10 seconds, then sfcache statistics will not show. Data is flushed to disk faster. In this case, you can determine the caching usage based on the WB Hit Ratio.
Hit Ratio (VxFS cache)	Percentage of total I/Os that are satisfied from the cache. Displayed for reads and writes.
Data Read (VxFS cache)	Data read from the cache.
Data Written (VxFS cache)	Data written to the cache.
Files Cached (VxFS cache)	Number of files present in the cache.
Files Pinned (VxFS cache)	Number of pinned files in the cache.
Data Pinned (VxFS cache)	Amount of data pinned in the cache.

Appendix D: Further reading

Besides this deployment guide, Symantec offers a range of documentation to help you learn about and use SmartIO.

To learn more about ...	See ...
SmartIO benefits, deployment scenarios, and administrative tasks	Symantec Storage Foundation and High Availability Solutions SmartIO Blueprint for Linux

How SmartIO can impact the performance of your environment, and best practices developed from our internal analysis	Symantec Storage Foundation and High Availability SmartIO for Linux Assessment Guide
SmartIO use cases, detailed administrative tasks, troubleshooting, and error handling	Symantec Storage Foundation and High Availability Solutions SmartIO for Solid State Drives Solutions Guide
sfcache command options	sfcache(1M) manual page

Appendix E: Known issues and limitations

- A known issue in Storage Foundation 6.1 prevents the `smartiomode mount` option from enabling caching for a file system during mount or remount if the cache area is of association type `noauto`. As a workaround, use the `sfcache enable` command to enable VxFS caching or auto cache areas to make sure that VxFS caching is enabled at boot time. This issue will be fixed in version 6.2.
- Resizing of cache areas that are created on an existing VxVM volume with the `dg/volname` option of the `sfcache` command is currently not supported.

For further known issues and limitations of SmartIO, see the [Storage Foundation 6.1 Release Notes](#).