SYMANTEC SOLUTIONS DEPLOYMENT GUIDES

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

Ritesh Shah

December, 2014

Table of Contents

Introduction

This document explains how to use Symantec Storage Foundation and High Availability (SFHA) SmartIO caching technology to increase performance of existing infrastructure by explaining how to setup and manage SmartIO for volumes and file systems. The document also covers some application-specific topics.

In today's world of tighter 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

SmartIO feature was introduced in SFHA Solutions for Linux version 6.1, and has been introduced on the Oracle Solaris platform for SFHA 6.2 release. SmartIO improves the performance per IOP by caching the hot data within the host thereby reducing the load on the SAN. SmartIO does not require in-depth knowledge of the hardware technologies involved, and it can be managed by a single command line interface or through the Veritas Operations Manager (GUI). It uses advanced and customizable heuristics to determine the type of data to cache and how that data gets removed to or from the cache. The heuristics take advantage of SFHA Solutions' knowledge of workload characteristics. The intelligent heuristics is completely transparent to the user.

SmartIO creates a cache area on the device as specified by the user for storing the cache data. Depending on the type that is specified during creation of the cache area, it can be controlled either by VxFS (VxFS caching) or VxVM (VxVM caching). As mentioned earlier, the cache area can be administered either through the command line interface (sfcache) or through the Veritas Operations Manager (VOM) G.U.I without any application downtime.

SmartIO supports read and write-back caching for the VxFS file systems that are mounted on VxVM volumes, in multiple caching modes and configurations. SmartIO also supports block-level read caching for applications running on VxVM volumes. By design, the device being used for caching needs to be under the VxVM control.

After caching is enabled, SmartIO controls the population of cache as well as removal of cold data to make space for newer data to be cached. SmartIO does this based on intelligent heuristics built into the product.

SmartIO benefits

By adding any solid-state drive, such as Peripheral Component Interconnect Express (PCIe), Serial ATA (SATA), or Serial SCSI (SAS), from any enterprise vendor into existing servers and configuring SmartIO, the performance of I/O intensive applications can be significantly improved. This enables the system to reduce the load on the backend storage and also reduce the congestion in the SAN. SmartIO has a cascading effect, wherein the reduction is SAN congestion can potentially improve the performance of hosts which do not have SmartIO configured.

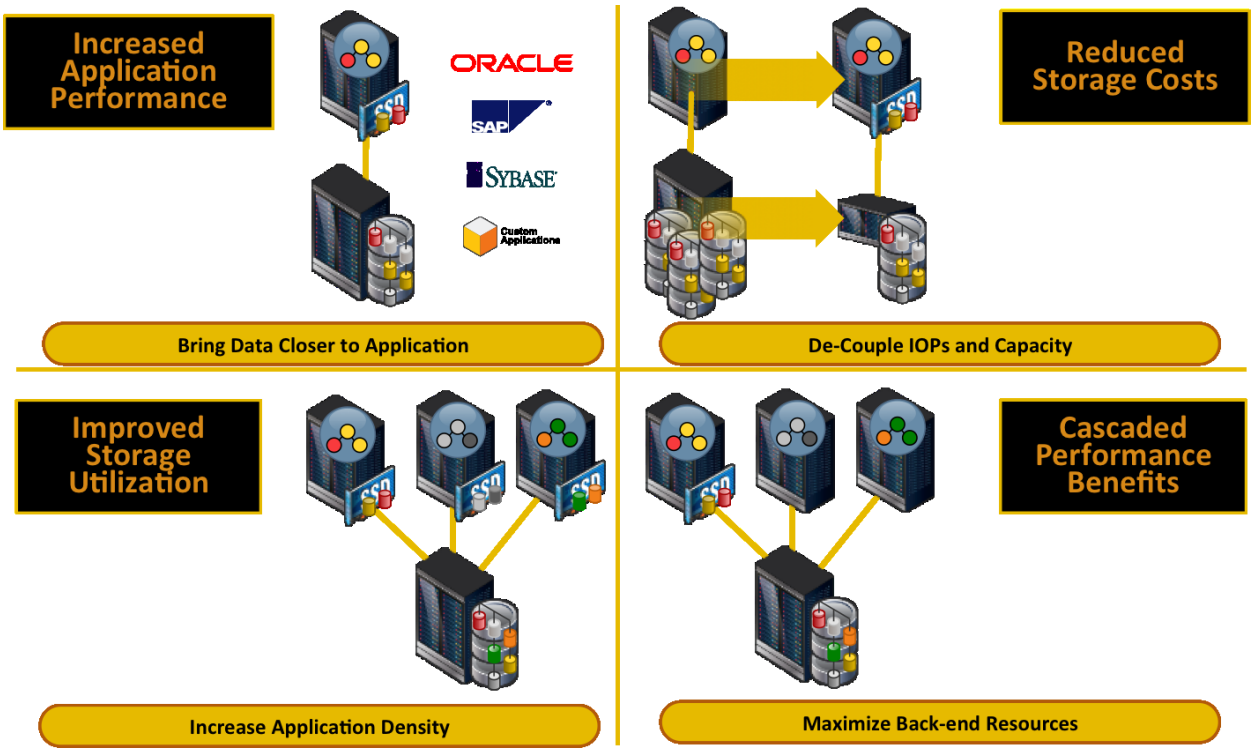


Figure 1: The four benefits of SmartIO

As shown in Figure 2, for an 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 requests can be handled within the server, and therefore served at a much faster speed, significantly improving the application's performance.

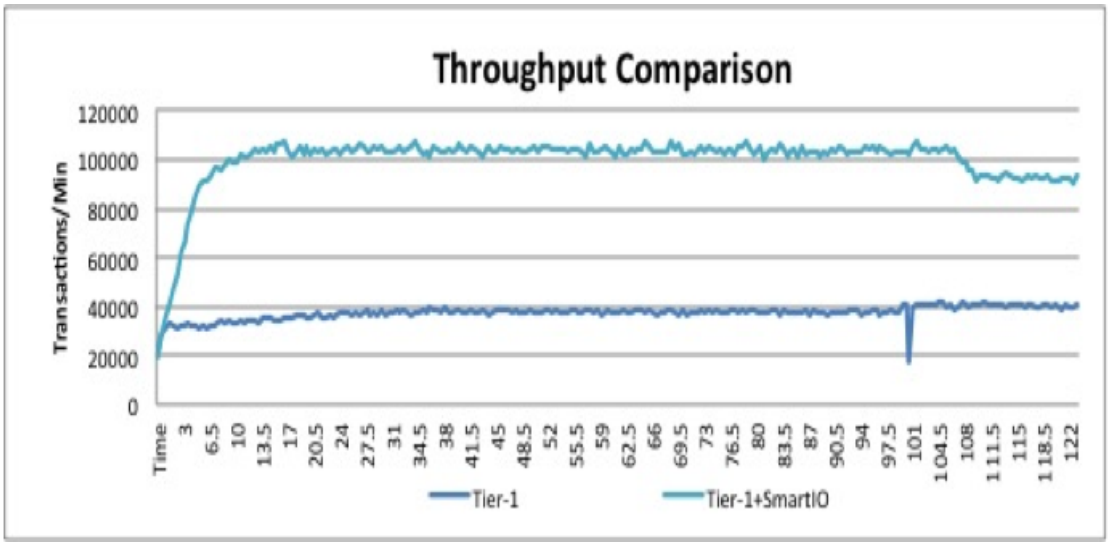


Figure 2: OLTP throughput comparison

While this is a significant direct gain for the application, other benefits from SmartIO also provide substantial value in an enterprise datacenter. With SmartIO, the majority of read requests can be handled within the server so that the performance needs of the back-end storage arrays are

substantially reduced. As can be seen form the below SAN IO graph (Figure 3), the read IO requirement from the storage array decreases substantially as more and more reads are serviced from SmartIO.

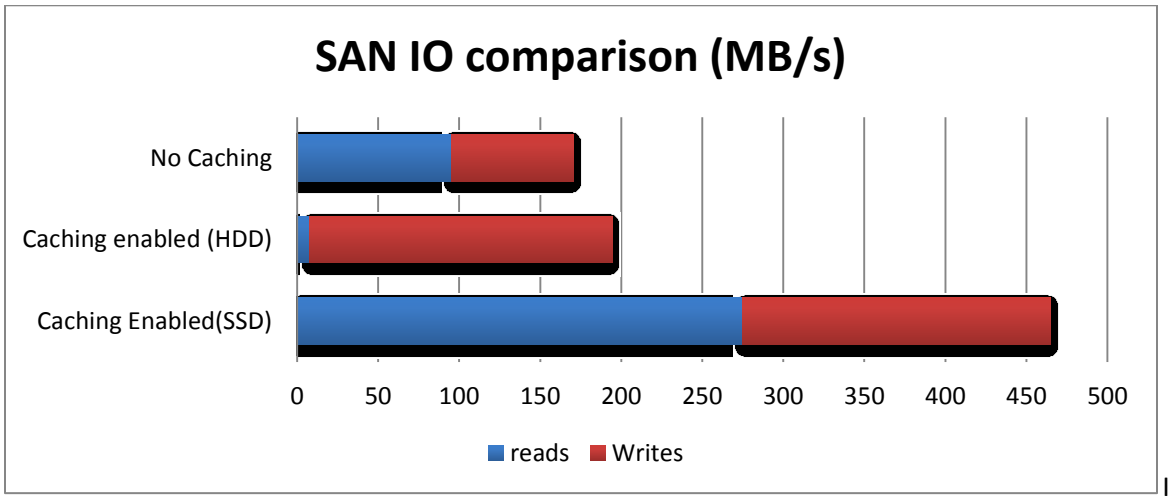


Figure 3 Comparing the break up of IOs for caching enabled and disabled

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 cheaper Tier 2 array brings better performance across multiple OLTP workloads than our Tier 1 array at 39% of the cost.

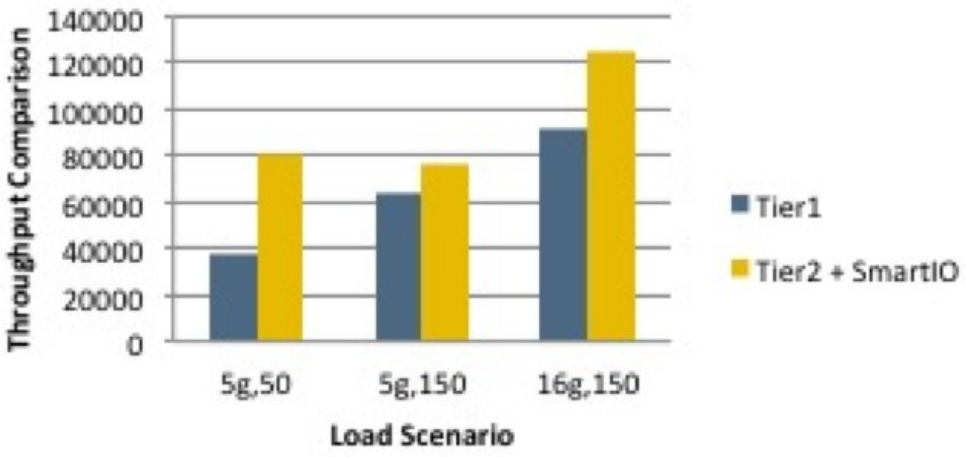


Figure 4: Tier 1 vs. Tier 2 + SmartIO

Essentially, SmartIO caching technology lets you decouple the need for high IOPs and low latency 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 and High Availability Solutions 6.2, the number and type of cache areas per system is limited to one, but each cache area can cache multiple volumes or file systems).

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

```
/root #vxdisk -o ssd list
DEVICE          SIZE(MB)      PHYS_ALLOC(MB)  GROUP           TYPE      RECLAIM_CMD
sunf80-0_0      190734        N/A             -               ssd
sunf80-1_0      190734        N/A             -               ssd
sunf80-2_0      190734        N/A             -               ssd
sunf80-3_0      190734        N/A             -               ssd
```

We selected the F80 SSD device to use as the cache area and initialize it for use with VxVM:

```
/root #vxdisksetup -i sunf80-0_0
/root #vxdisksetup -i sunf80-1_0
```

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

```
/root #sfcache create -t vxvm 100g sunf80-0_0 sunf80-1_0 \ vxvm_cachevol
```

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

```
/root #sfcache maxsize sunf80-0_0 sunf80-1_0
Maxsize is 571375616 (278992Mb))
```

and create a 100-GB cache area for VxFS caching:

```
/root #sfcache create -t vxfs 100g sunf80-0_0 sunf80-1_0 \ vxfs_cachevol
```

Alternatively, you can create a VxVM disk group and volume on the SSD device and configure this volume as a VxFS cache area or VxVM cache area with the `sfcache` command. However, resizing these cache areas is not supported, so the above method is highly recommended. Also, it is preferable to use a whole SSD for caching because 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 # sfcache set --noauto vxfs_cachevol
```

We can now list the configured cache areas:

```
/root #sfcache list
NAME              TYPE    SIZE        ASSOC-TYPE  STATE       DEVICE
vxvm_cachevol     VxVM    100.00g     AUTO        ONLINE      sunf80-0_0,sunf80-1_0
vxfs_cachevol      VxFS    100g       NOAUTO      ONLINE      sunf80-0_0,sunf80-1_0
```

`sfcache resize` can be used to resize the cache. For example to grow the VxFS cache area to 200 GB, use:

```
/root # sfcache resize 200g vxfs_cachevol
```

You can check the result with `sfcache list`:

```
/root #sfcache list
NAME             TYPE    SIZE       ASSOC-TYPE  STATE      DEVICE
vxvm_cachevol    VxVM    100.00g    AUTO        ONLINE     sunf80-0_0,sunf80-1_0
vxfs_cachevol    VxFS    200g       NOAUTO      ONLINE     sunf80-0_0,sunf80-1_0
```

and shrink the VxFS cache area again to 100 GB:

```
/root # 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 #vxlist vol
TY   VOLUME          DISKGROUP               SIZE STATUS    LAYOUT    LINKAGE
vol  vol1            testdg                100.00g healthy   striped   -
vol  vol2            testdg                100.00g healthy   striped   -
vol  vxfs_cachevol   sfcache_defaultdg     100.00g healthy   striped   -
vol  vxvm_cachevol   sfcache_defaultdg     100.00g healthy   striped   -
```

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 #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
Layout: STRIPE
Number of Columns: 2


ASSOCIATED DATA OBJECTS:

Volume: testdg/vol1
Size: 100.00g
State: AUTO
Kstate: STOPPED
Caching Mode: read
```

```
Volume: testdg/vol2
Size: 100.00g
State: AUTO
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 #sfcache stat vxvm_cachevol
                                          HIT RATIO      ART(Hit)ms     ART(Miss)ms
BYTES
NAME                              %CACHE RD      WR      RD      WR      RD       WR
RD      WR
TYPE: VxVM
vxvm_cachevol                     0.00   87.23   14.78   0.092   0.638   8.656
1.749    139.00k 17.00k

ASSOCIATED DATA OBJECTS:
testdg/vol1                       0.00   87.23   14.78   0.092   0.638   8.656
1.749    139.00k 17.00k
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 reference.

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

```
/root # dd if=/dev/vx/rdsk/testdg/vol2 \
of=/dev/null bs=65536 count=10
10+0 records in
10+0 records out
```

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

```
/root # sfcache stat vxvm_cachevol
                                  HIT RATIO      ART(Hit)ms     ART(Miss)ms      BYTES
NAME                          %CACHE 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 # dd if=/dev/vx/rdsk/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 #sfcache stat vxvm_cachevol
                                    HIT RATIO      ART(Hit)ms     ART(Miss)ms       BYTES
NAME                          %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 #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 #sfcache stat vxvm_cachevol
                                          HIT RATIO     ART(Hit)ms    ART(Miss)ms      BYTES
NAME                          %CACHE 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

To see clearly the effect of SmartIO caching ensure that you are using directIO reads, i.e. eliminating the effect of page cache. In the following examples, the file system was mounted with –o cio to force directIO. This is not required in case the application does directIO on its own. Please note that directIO is not a requirement for SmartIO; It is used here only to clearly see the benefits from SmartIO and also the changes to caching stats.

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 # sfcache list -l vxfs_cachevol
Cachearea: vxfs_cachevol
Assoc Type: NOAUTO
Type: VxFS
Size: 100.00g
State: ONLINE
Layout: STRIPE
Number of Columns: 2

/dev/vx/dsk/sfcache_defaultdg/vxfs_cachevol:
FSUUID                                                  SIZE  MODE       MOUNTPOINT
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` command:

```
/root #mount -F vxfs \
-o cio,remount,smartiomode=read /dev/vx/dsk/testdg/vol1 /test1
```

or use the `sfcache` command:

```
/root #sfcache enable /test1
```

and check it with `sfcache list`:

```
/root #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 create a large file in `/test1` using `dd`:

```
/root #dd if=/dev/zero of=/test1/file1 bs=1M count=10240
10240+0 records in
10240+0 records out
```

Confirm that the cache is still empty:

```
/root #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 #dd if=/test1/file1 of=/dev/null bs=8k
1310720+0 records in
1310720+0 records out
```

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

```
/root #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 #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 #dd if=/test1/file1 of=/dev/null bs=8k
1310720+0 records in
1310720+0 records out

/root #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 F80 flash device (on top of which the cache area has been built):

```
/root #iostat -xnC 5 5
                    extended device statistics
r/s   w/s    kr/s kw/s wait actv wsvc_t asvc_t  %w  %b device…
0.0    0.0    0.1    0.0   0.0   0.0    0.0      0.0       0    0  c4t5006016808602AEEd3
0.0    0.0    0.0    0.0   0.0   0.0    0.0      0.0       0    0  c4t5006016808602AEEd2
0.0    0.0    0.0    0.0   0.0   0.0    0.0      0.0       0    0  c4t5006016808602AEEd1
0.0    0.0    0.0    0.0   0.0   0.0    0.0      0.0       0    0  c4t5006016808602AEEd0
0.3    0.1    0.5    0.2   0.0   0.0    0.0      0.2       0    0  c4t5006016008602AEEd0
0.0    0.0    0.0    0.0   0.0   0.0    0.0      0.0       0    0  c4t5006016008602AEEd3
…
```

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

```
/root #umount /test1
/root #mount –F vxfs –o cio /dev/vx/dsk/testdg/vol1 /test1
/root #sfcache enable /test1
/root #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 #dd if=/test1/file1 of=/dev/null bs=8k
1310720+0 records in
1310720+0 records out
/root #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 writing to a file that is already in the read cache, the cached content is also updated during the write. In the example below, a 10-GB file `file1`, in `/test1` has been read once to populate the cache, and we'll overwrite 5 GB of `file1`:

```
/root #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 #dd if=/dev/zero of=/test1/file2 bs=1M count=5120 \ conv=notrunc
5120+0 records in
5120+0 records out
```

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

```
/root #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 `conv=notrunc` option to the `dd write` command to make sure that the file is not truncated before it is written. Truncating the file makes the cached content of the file invalid.

Write-back caching

SmartIO provides write caching in the write-back mode. In write-back mode, the application writes returns success after the data is written to the SmartIO cache, which is usually on an SSD. Later, 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. Write-back caching is supported for synchronous and direct writes.

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 # mkfs -F vxfs -o cio /dev/vx/rdsk/testdg/vol3
    version 10 layout
    209715200 sectors, 104857600 blocks of size 1024, log size 65536 blocks
    rcq size 4096 blocks
    largefiles supported
    maxlink supported
/root sfcache set --auto vxfs_cachevol
/root 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
```

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

```
/root mount -F vxfs /dev/vx/dsk/testdg/vol3 \
/test3
/root sfcache list /test3
/test3:
READ CACHE    WRITEBACK    MODE       PINNED  NAME
      0 KB         0 KB    read       no      /test3
```

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

```
/root #mount -F vxfs /dev/vx/dsk/testdg/vol3 \
-o remount,smartiomode=writeback /test3
```

```
/root #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 #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            0 KB       0.00 %            0 KB
```

To show the effects of write-back caching, let's compare the file systems `/test1` with read caching and `/test3` with write-back caching by running a quick file system benchmark with write workloads on each file system:

```
/root # sfcache list /test1
/test1:
READ CACHE    WRITEBACK    MODE       PINNED  NAME
      0 KB         0 KB    read       no      /test1
/root # sfcache list /test3
/test3:
READ CACHE    WRITEBACK    MODE       PINNED  NAME
      0 KB         0 KB    writeback  no      /test3
```

We use the IOzone benchmarking tool (www.iozone.org) with the write, rewrite, and random read/write tests writing 10 100-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 # iozone -i 0 -i 2 -i 4 -o -t 10 -r 8k -s 100m
        Iozone: Performance Test of File I/O
                Version $Revision: 3.429 $
                Compiled for 64 bit mode.
                Build: Solaris10gcc-64

        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: Tue Nov 18 01:57:55 2014

        SYNC Mode.
        Record Size 8 KB
        File size set to 102400 KB
        Command line used: iozone -i 0 -i 2 -i 4 -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  =   56933.34 kB/sec
            Parent sees throughput for 10 initial writers   =   56487.15 kB/sec
            Min throughput per process                      =    5645.47 kB/sec
            Max throughput per process                      =    5720.11 kB/sec
            Avg throughput per process                      =    5693.33 kB/sec
            Min xfer                                        =  101072.00 kB

            Children see throughput for 10 rewriters        =   54285.12 kB/sec
            Parent sees throughput for 10 rewriters         =   54273.40 kB/sec
            Min throughput per process                      =    5372.43 kB/sec
            Max throughput per process                      =    5474.16 kB/sec
            Avg throughput per process                      =    5428.51 kB/sec
            Min xfer                                        =  100496.00 kB

            Children see throughput for 10 random readers   = 6394304.56 kB/sec
            Parent sees throughput for 10 random readers    = 6171061.83 kB/sec
            Min throughput per process                      =  582879.44 kB/sec
            Max throughput per process                      =  696248.44 kB/sec
            Avg throughput per process                      =  639430.46 kB/sec
            Min xfer                                        =   86128.00 kB

            Children see throughput for 10 random writers   =   43340.84 kB/sec
            Parent sees throughput for 10 random writers    =   43260.01 kB/sec
            Min throughput per process                      =    4330.47 kB/sec
            Max throughput per process                      =    4339.80 kB/sec
            Avg throughput per process                      =    4334.08 kB/sec
            Min xfer                                        =  102160.00 kB



      iozone test complete.
```

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

```
      /root #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 # iozone -i 0 -i 2 -i 4 -o -t 10 -r 8k -s 100m
              Iozone: Performance Test of File I/O
                      Version $Revision: 3.429 $
                      Compiled for 64 bit mode.
                      Build: Solaris10gcc-64

              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: Tue Nov 18 02:00:59 2014

            SYNC Mode.
            Record Size 8 kB
            File size set to 102400 kB
            Command line used: iozone -i 0 -i 2 -i 4 -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  =   68781.56 kB/sec
            Parent sees throughput for 10 initial writers   =   65751.86 kB/sec
            Min throughput per process                      =    6722.80 kB/sec
            Max throughput per process                      =    6968.93 kB/sec
            Avg throughput per process                      =    6878.16 kB/sec
            Min xfer                                        =   98792.00 kB

            Children see throughput for 10 rewriters        =   71935.48 kB/sec
            Parent sees throughput for 10 rewriters         =   70719.01 kB/sec
            Min throughput per process                      =    7140.71 kB/sec
            Max throughput per process                      =    7271.46 kB/sec
            Avg throughput per process                      =    7193.55 kB/sec
            Min xfer                                        =  100584.00 kB

            Children see throughput for 10 random readers   = 6352464.31 kB/sec
            Parent sees throughput for 10 random readers    = 6128047.30 kB/sec
            Min throughput per process                      =  590566.94 kB/sec
            Max throughput per process                      =  739009.62 kB/sec
            Avg throughput per process                      =  635246.43 kB/sec
            Min xfer                                        =   81856.00 kB

            Children see throughput for 10 random writers   =   58425.62 kB/sec
            Parent sees throughput for 10 random writers    =   54086.18 kB/sec
            Min throughput per process                      =    5817.23 kB/sec
            Max throughput per process                      =    5872.07 kB/sec
            Avg throughput per process                      =    5842.56 kB/sec
            Min xfer                                        =  101504.00 kB


      iozone test complete.
```

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

```
      /root #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	55.59	53.01	6244	42.3
Write-back cache	67.76	70.2	6203	57

The data clearly shows the performance improvements through write-back caching for synchronous and direct write workloads. Please note the data and benefits depend on your environment.

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

```
/root #iozone -i 0 -i 2 -i 4 -o -t 10 -r 4m -s 100m
        Iozone: Performance Test of File I/O
                Version $Revision: 3.429 $
                Compiled for 64 bit mode.
                Build: Solaris10gcc-64

        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: Tue Nov 18 02:18:47 2014

        SYNC Mode.
        Record Size 4096 kB
        File size set to 102400 kB
        Command line used: iozone -i 0 -i 2 -i 4 -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  =  429114.27 kB/sec
        Parent sees throughput for 10 initial writers   =  411692.07 kB/sec
        Min throughput per process                      =   40901.77 kB/sec
        Max throughput per process                      =   46936.67 kB/sec
        Avg throughput per process                      =   42911.43 kB/sec
        Min xfer                                        =   90112.00 kB

        Children see throughput for 10 rewriters        =  674523.90 kB/sec
        Parent sees throughput for 10 rewriters         =  669177.26 kB/sec
        Min throughput per process                      =   65058.79 kB/sec
        Max throughput per process                      =   71826.53 kB/sec
        Avg throughput per process                      =   67452.39 kB/sec
        Min xfer                                        =   94208.00 kB

        Children see throughput for 10 random readers   =  696955.95 kB/sec
```

```
        Parent sees throughput for 10 random readers    =  688786.00 kB/sec
        Min throughput per process                      =   67668.24 kB/sec
        Max throughput per process                      =   74726.20 kB/sec
        Avg throughput per process                      =   69695.60 kB/sec
        Min xfer                                        =   94208.00 kB

        Children see throughput for 10 random writers   =  670250.29 kB/sec
        Parent sees throughput for 10 random writers    =  655977.06 kB/sec
        Min throughput per process                      =   65093.09 kB/sec
        Max throughput per process                      =   70356.97 kB/sec
        Avg throughput per process                      =   67025.03 kB/sec
        Min xfer                                        =   94208.00 kB


    iozone test complete.
```

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

```
/root #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        1.578 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.

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

```
/root #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`, which is 10 GB, into the cache:

```
/root #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 #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 #sfcache pin /test3/file1
/root #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 # sfcache pin -o load /test3/file1
/root #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).

Application awareness

While caching is intended to improve the performance of applications, most caching solutions only provide a way to enable caching for OS level constructs (volumes, files etc.). SmartIO provides a way to enable caching based on application constructs by providing intelligent caching templates for some of the more common applications. By providing these templates, SmartIO attempts to reduce the guesswork involved in configuring caching for the applications. It provides templates for Sybase and Oracle databases, and based on the workload (OLTP/OLAP), the DBA can enable the caching heuristics on either the whole database or specific logical constructs (e.g partitions, tablespaces etc.)

To enable caching on a database based on the workload characteristics:

oracle:~$ **sfcache app cacharea=vxfs_cachevol oracle -S tpcc –H\ /export/home/oracle/product/11gR2 -o setdefaults --type=OLTP**
sfcache: INFO: V-3-30092: Oracle Instance tpcc is running
sfcache: INFO: V-3-30093: Storing DB details at /data_disks/.CACHE_INFO
sfcache: INFO: V-3-30094: Setting OLTP policies
sfcache: INFO: V-3-30095: Setting nocache mode to /data_disks

sfcache: INFO: V-3-30095: Setting nocache mode to /log_disks
sfcache: INFO: V-3-30095: Setting nocache mode to /log_disks/log_1_1
sfcache: INFO: V-3-30095: Setting nocache mode to /log_disks/log_1_2
sfcache: INFO: V-3-30095: Setting nocache mode to /log_disks/log_1_3
sfcache: INFO: V-3-30095: Setting nocache mode to /log_disks/log_1_4
sfcache: INFO: V-3-30095: Setting nocache mode to /log_disks/log_1_5
sfcache: INFO: V-3-30095: Setting nocache mode to /log_disks/log_1_6
sfcache: INFO: V-3-30095: Setting nocache mode to /export/home/oracle/product/11gR2/dbs/arch
sfcache: INFO: V-3-30099: No AWR snapshots are available

To enable caching on a specific Oracle database construct. The following command can be used for enabling read caching for an Oracle tablespace called TESTTBS1.

oracle:~$ **sfcache app cacharea=vxfs_cachevol oracle -S tpcc –H\ /export/home/oracle/product/11gR2 -o set --cachemode=read --tablespace=TESTTBS1;**
sfcache: INFO: V-3-30092: Oracle Instance tpcc is running
sfcache: INFO: V-3-30093: Storing DB details at /data_disks/.CACHE_INFO
sfcache: INFO: V-3-30095: Setting read mode to /data_disks/file1
sfcache: INFO: V-3-30095: Setting read mode to /data_disks/file2
sfcache: INFO: V-3-30095: Setting read mode to /data_disks/file3
sfcache: INFO: V-3-30095: Setting read mode to /data_disks/file4
sfcache: INFO: V-3-30095: Setting read mode to /data_disks/file5

For more details and options, please refer to the sfcache (1M) manual page on the SORT website.

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 (`sflorat52–1-v01 and sflorat52-1-v02`) 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@ sflorat52-1-v01~]# cfsmntadm display -v /share1
  Mount Point    : /share1
  Shared Volume  : sharevol1
  Disk Group     : sharedg

  Primary Node:   sflorat52-1-v01
  Primary Election Policy:
  Dependent Checkpoints or Snapshots:

  NODE NAME       STATUS            MOUNT OPTIONS
   sflorat52-1-v01 MOUNTED           suid,rw
   sflorat52-1-v02 MOUNTED           suid,rw
```

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

```
[root@ sflorat52-1-v01 ~]# sfcache list
NAME                          TYPE   SIZE       ASSOC-TYPE  STATE      DEVICE
vxfs_cachevol                 VxFS   100.00g    AUTO        ONLINE     sunf80-0_0

[root@ sflorat52-1-v02 ~]# sfcache list
NAME                          TYPE   SIZE       ASSOC-TYPE  STATE      DEVICE
vxfs_cachevol                 VxFS   100.00g    AUTO        ONLINE     sunf80-0_0
```

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

```
[root@ sflorat52-1-v01 ~]# 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@ sflorat52-1-v02 ~]# 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 caches:

```
[root@ sflorat52-1-v01 ~]# dd if=/share1/file1 of=/dev/null bs=8k
1310720+0 records in
1310720+0 records out
[root@ sflorat52-1-v01 ~]# 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@ sflorat52-1-v02 ~]# dd if=/share1/file1 of=/dev/null bs=8k
1310720+0 records in
1310720+0 records out
[root@ sflorat52-1-v02 ~]# 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@ sflorat52-1-v01 ~]# cfsumount /share1
  Unmounting...
  /share1 got successfully unmounted from sflorat52-1-v02
  /share1 got successfully unmounted from sflorat52-1-v01
[root@ sflorat52-1-v01 ~]# cfsmount /share1
  Mounting...
  [/dev/vx/dsk/sharedg/sharevol1] mounted successfully at /share1 on sflorat52-1-v01
  [/dev/vx/dsk/sharedg/sharevol1] mounted successfully at /share1 on sflorat52-1-v02
```

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

```
[root@ sflorat52-1-v01 ~]# dd if=/share1/file1 of=/dev/null bs=8k
1310720+0 records in
1310720+0 records out
[root@ sflorat52-1-v02 ~]# dd if=/share1/file1 of=/dev/null bs=8k
1310720+0 records in
1310720+0 records out

[root@ sflorat52-1-v01 ~]# 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@ sflorat52-1-v02 ~]# 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 node[1]:

```
[root@ sflorat52-1-v01 ~]# dd if=/dev/zero of=/share1/file1 bs=1M count=5120
conv=notrunc
5120+0 records in
5120+0 records out

[root@ sflorat52-1-v01 ~]# 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@ sflorat52-1-v02~]# 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@ sflorat52-1-v02 ~]# dd if=/share1/file1 of=/dev/null bs=8k
1310720+0 records in
```

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

```
1310720+0 records out

[root@ sflorat52-1-v02 ~]# 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@ sflorat52-1-v01 ~]# sfcache pin -o load /share1/file2
[root@ sflorat52-1-v01 ~]# 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@ sflorat52-1-v02 ~]# 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
```

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. This behavior, called **cache reflection**, prevents loss of write-back data if a node fails. 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@ sflorat52-1-v01 ~]# cfsmntadm modify /share1 all+=smartiomode=writeback
  Cluster-configuration updated with changed Mount Options for node sflorat52-1-v01
  Mount Point /share1 remounted successfully on sflorat52-1-v01
  Cluster-configuration updated with changed Mount Options for node sflorat52-1-v02
  Mount Point /share1 remounted successfully on sflorat52-1-v02


[root@ sflorat52-1-v01 ~]# 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@ sflorat52-1-v02 ~]# 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, resulting in a total of 1 GB cache utilization on every node:

```
 [root@sflorat52-1-v01 ~]# 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
[root@sflorat52-1-v02 ~]# 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@sflorat52-1-v01 ~]# dd if=/dev/zero of=/share1/file1 bs=1M count=10240 \
oflag=direct,sync
10240+0 records in
10240+0 records out
[root@sflorat52-1-v01 share1]# sfcache stat -l /share1
       Cache Size:     100 GB
Cache Utilization:     1.0 GB ( 1.00 %)

Read Cache                                                                    Writeback
Hit Ratio    Data Read    Data Written    Files Cached    Files Pinned    Data Pinned    Hit Ratio    Data Written

/share1:
   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@sflorat52-1-v01 ~]# 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@sflorat52-1-v01 ~]# dd if=/dev/zero of=/share1/file2 bs=4M \
count=1024 oflag=direct,sync
1024+0 records in
1024+0 records out
 [root@sflorat52-1-v01; ~]# 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@sflorat52-1-v02 ~]# 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
```

Managing SmartIO with Veritas Operations Manager

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

Creating cache areas with Veritas Operations Manager

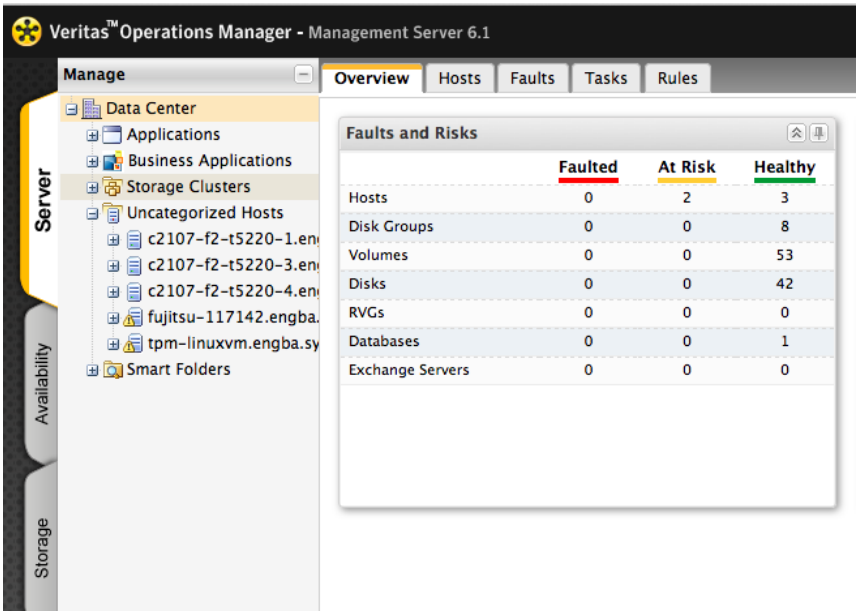
The Veritas Operations Manager Management Server console lets you create cache areas using the available SSD devices. The cache area name is system-generated and cannot be modified.

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.

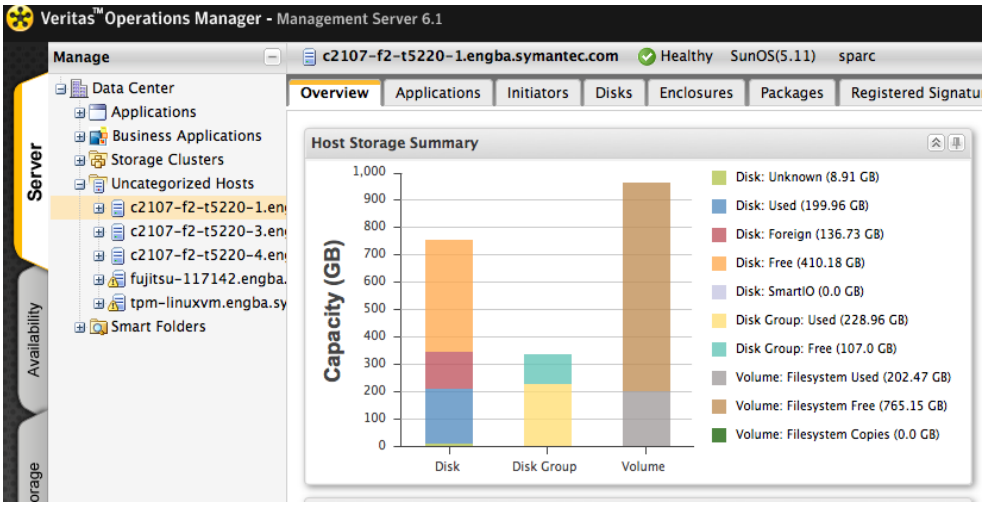
To create cache areas with Veritas Operations Manager

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

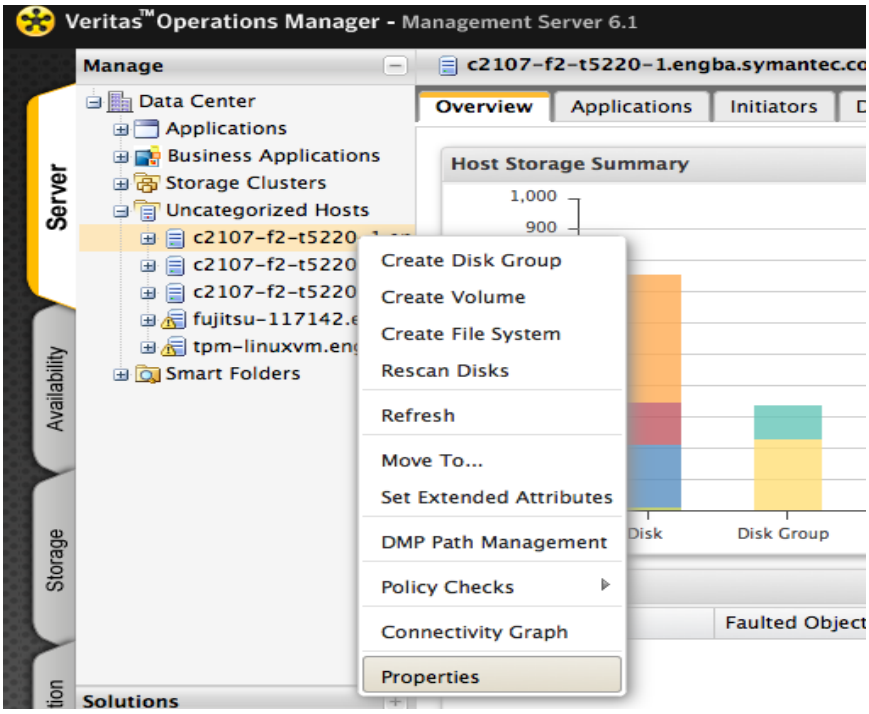
2 On the **Server** screen, in the left pane, expand **Manage**:



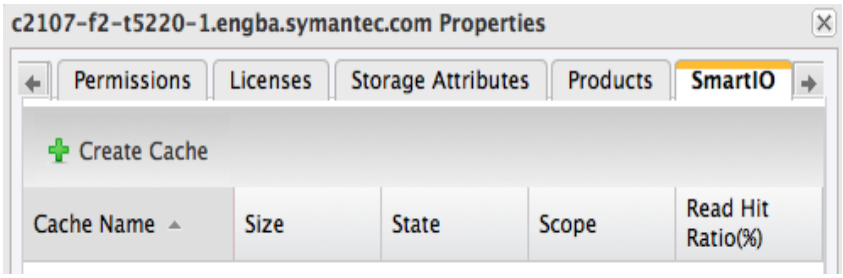
3 Expand the **Organization** or **Uncategorized Hosts** icons to locate and select the host (vlab-clstr1-vm2).



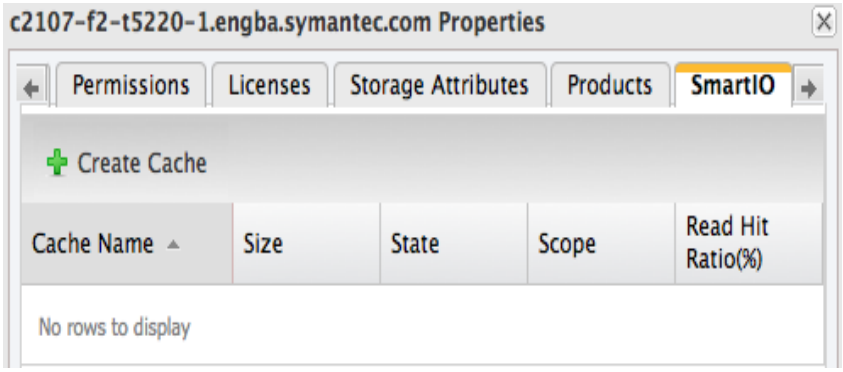
4 Right-click on the host and select **Properties**.



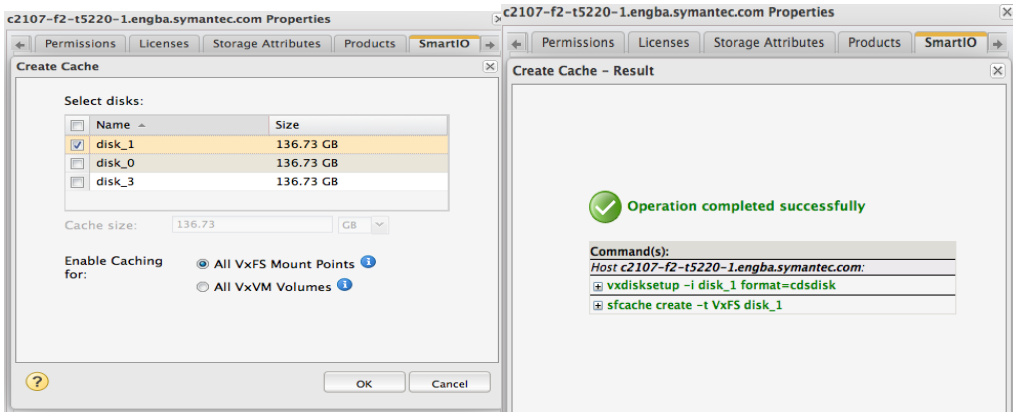
5 On the **Properties** window, click the **SmartIO** tab:



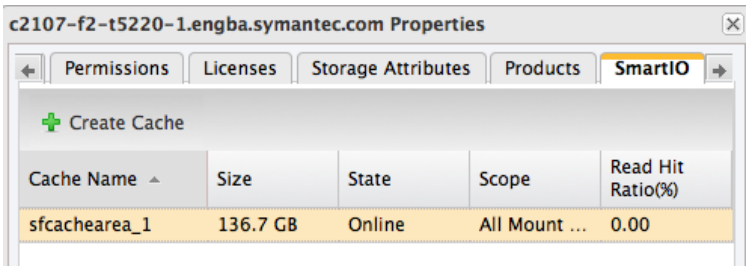
6 Click Create Cache.



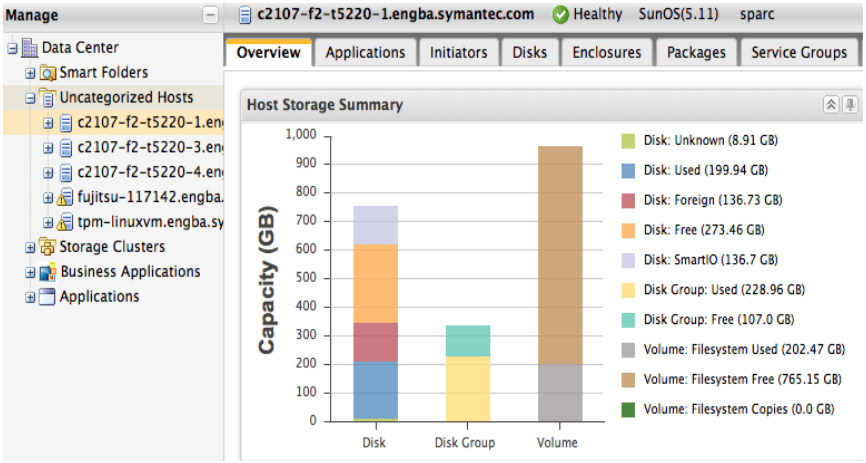
7 In the **Create Cache** panel, enter the details, and click **OK**.



The system automatically chooses the cache area name.



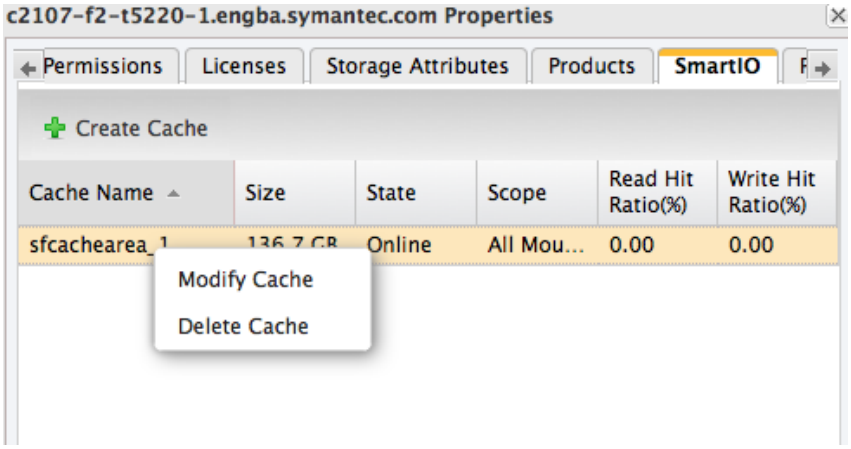
We now see that SmartIO caching has been enabled on vmdk0_10:



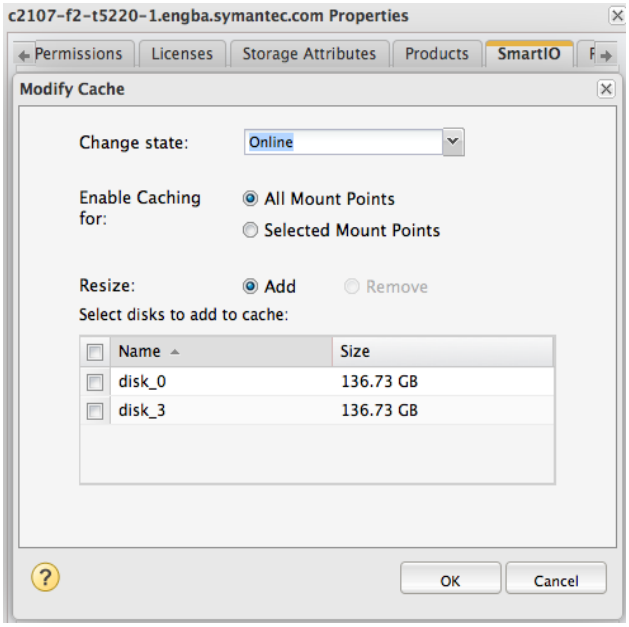
Modifying and deleting cache areas with Veritas Operations Manager

To modify a cache area

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

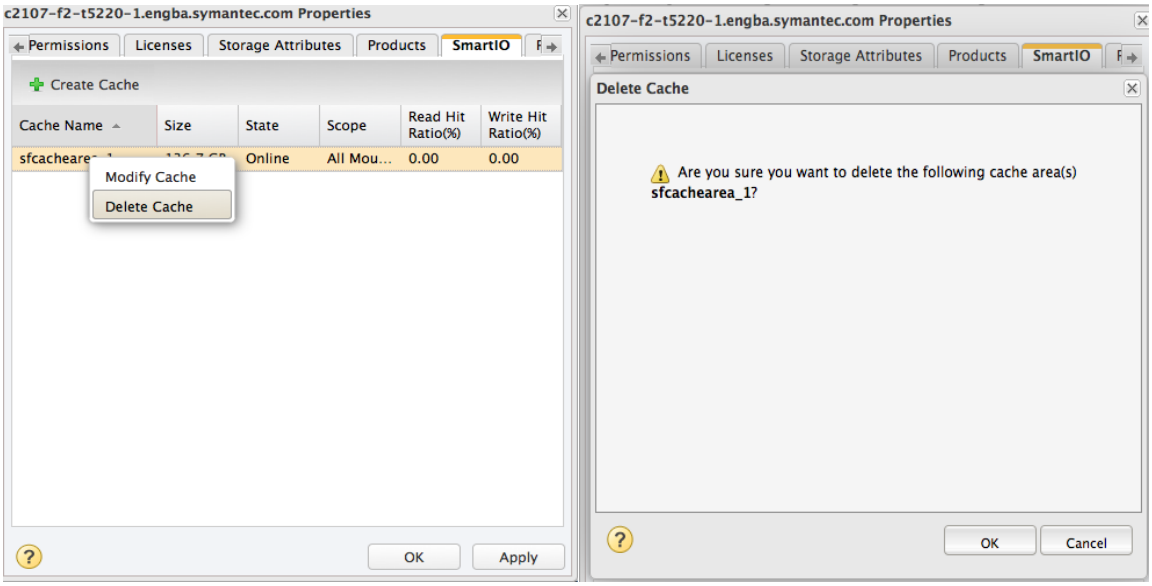


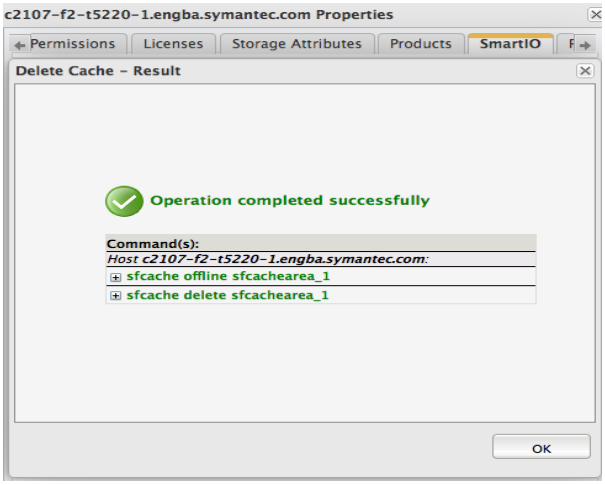
2. On the **Modify Cache** screen, we can change the state between online and offline, and enable caching for all mount points or selected mount points.



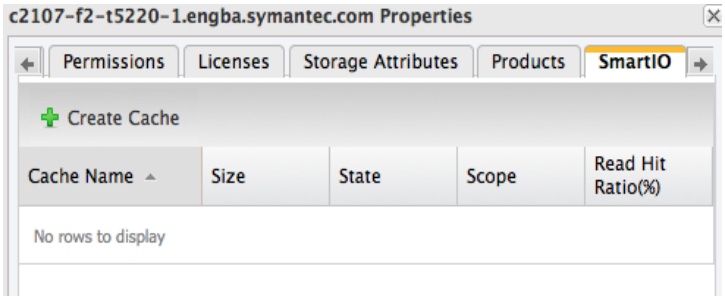
To delete a cache area

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:



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.

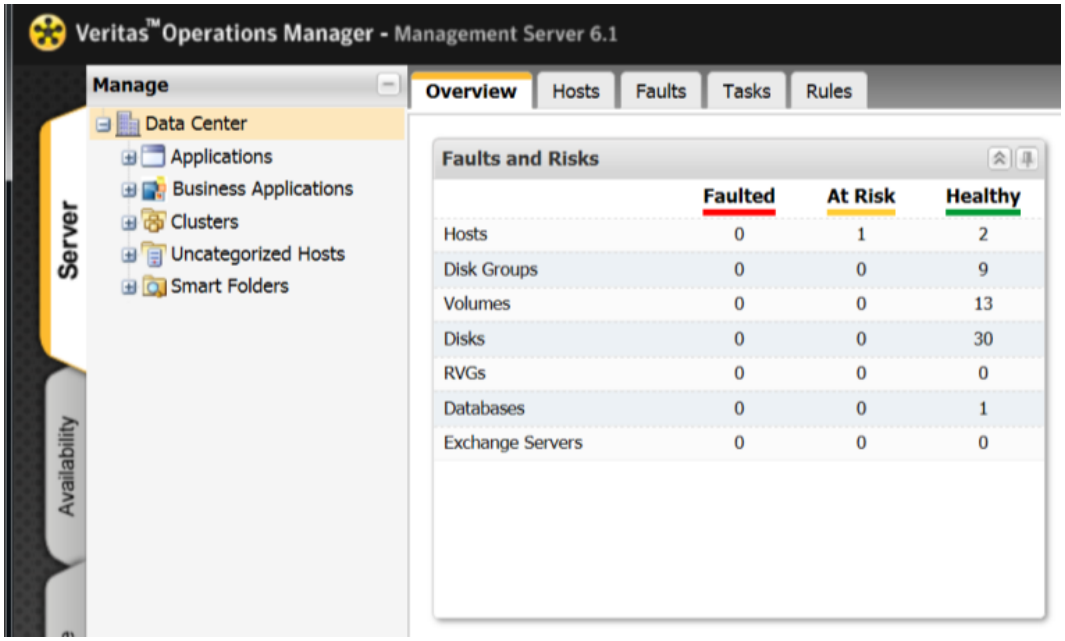
If the caching scope is set to a selected mount point or volume, caching is not enabled on any mount point or volume by default. You must explicitly enable caching on the required mount point or volume.

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.

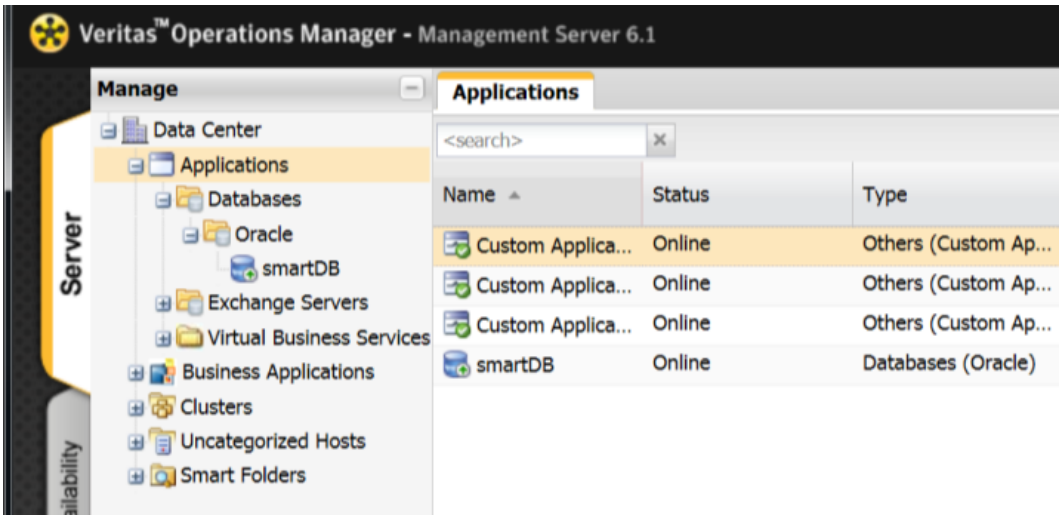
To enable or disable SmartIO caching

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

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

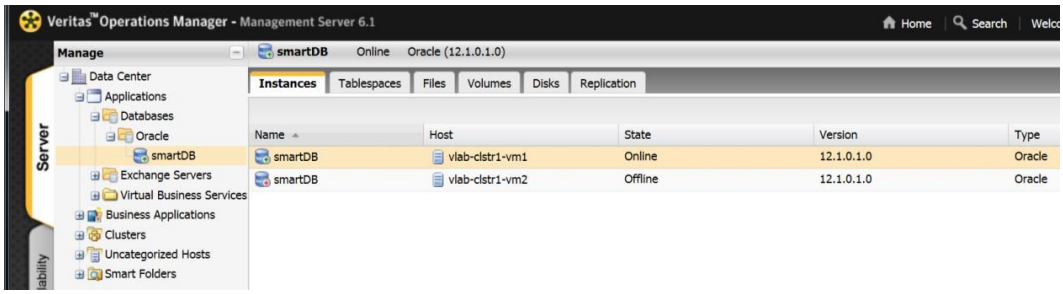


3 Expand the Organization (if configured), Applications, or Uncategorized Hosts icons:

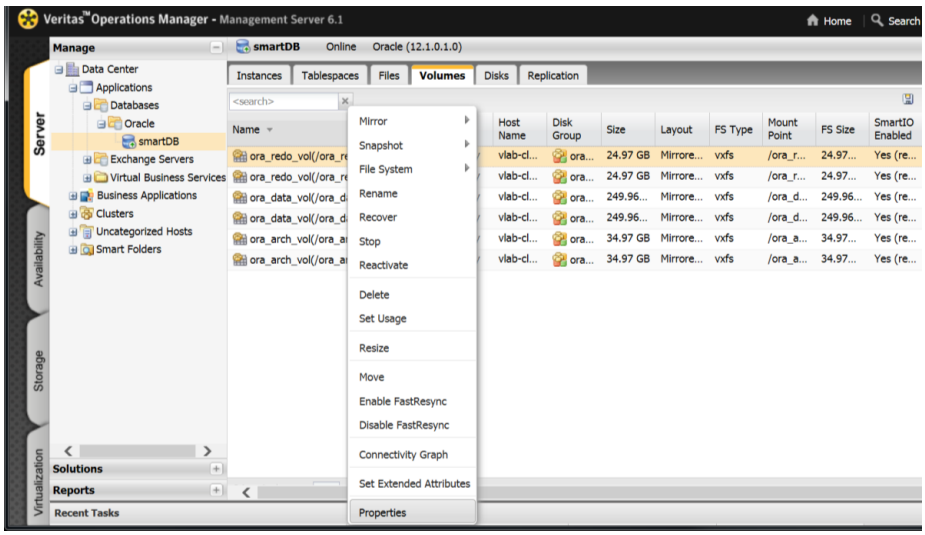


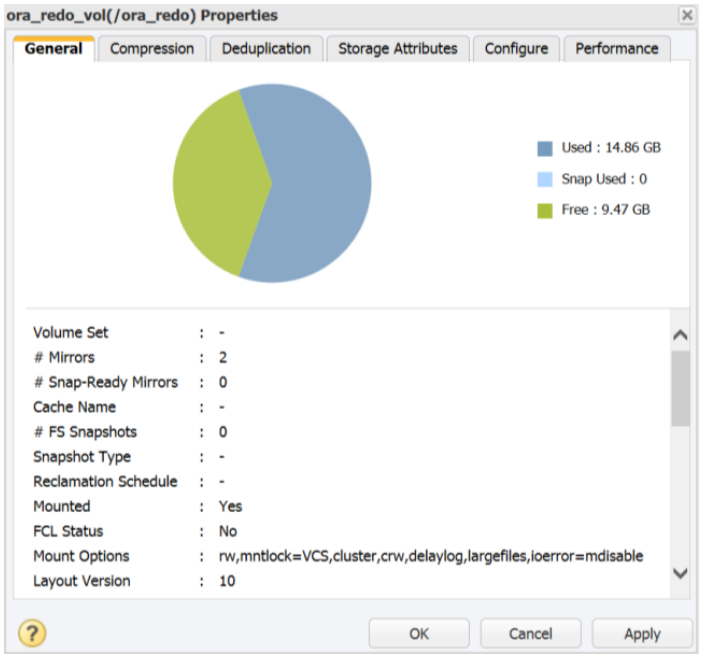
4 Do one of the following:

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



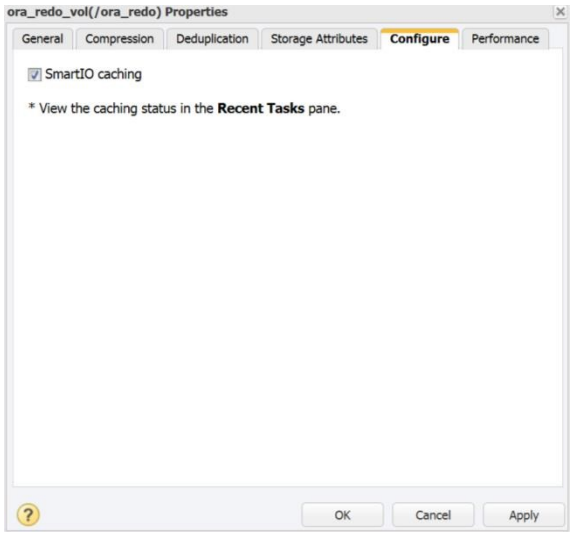
- Click on the **Volumes** tab and right-click a volume and then select **Properties**:





5 Click the **Configure** tab and do one of the following:

- Select the **SmartIO caching** check box to enable SmartIO caching.
- Clear the **SmartIO caching** check box to disable SmartIO caching.



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:

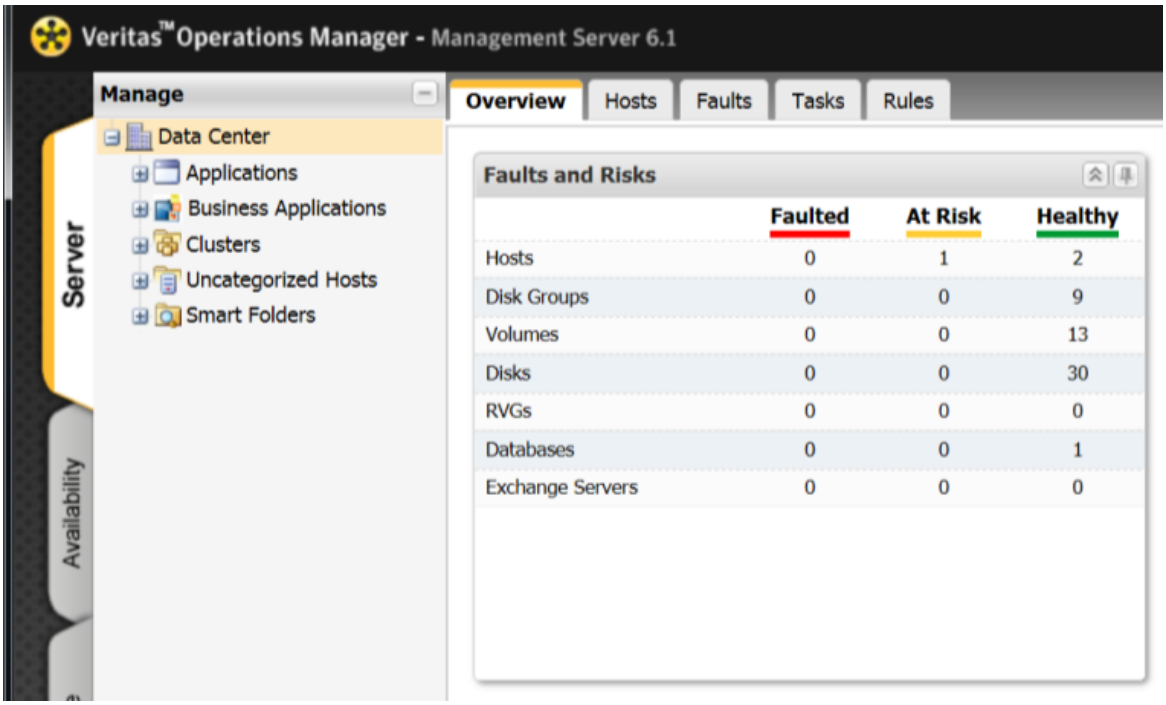
- Scope: Displays the scope of the cache such as all volumes or all mount points.

- 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.

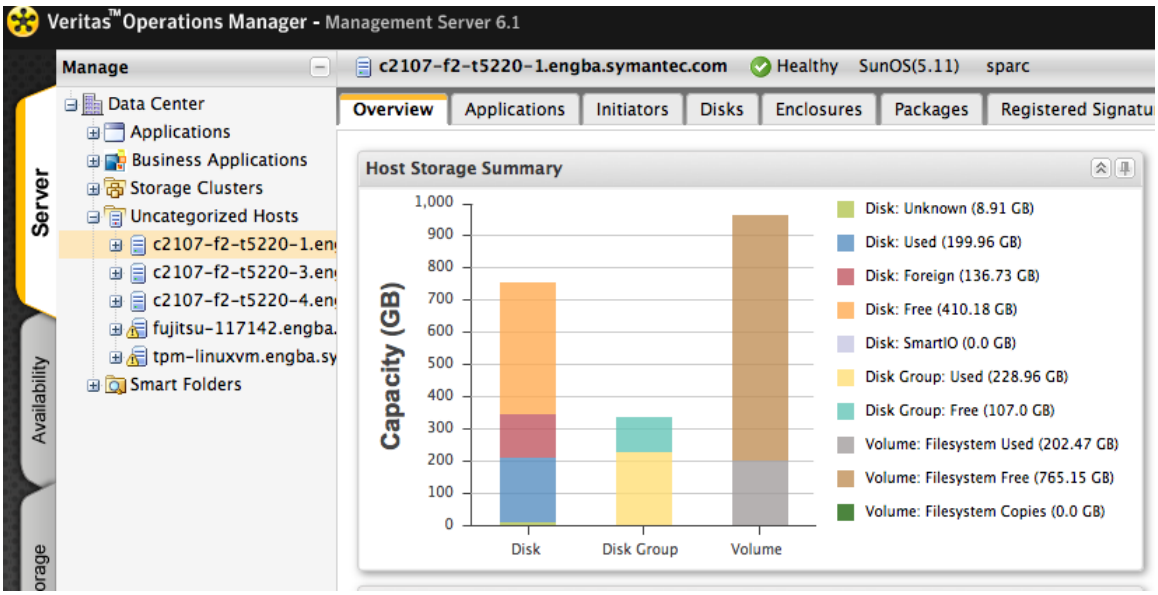
In this view, you can create, modify, and delete the cache area.

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

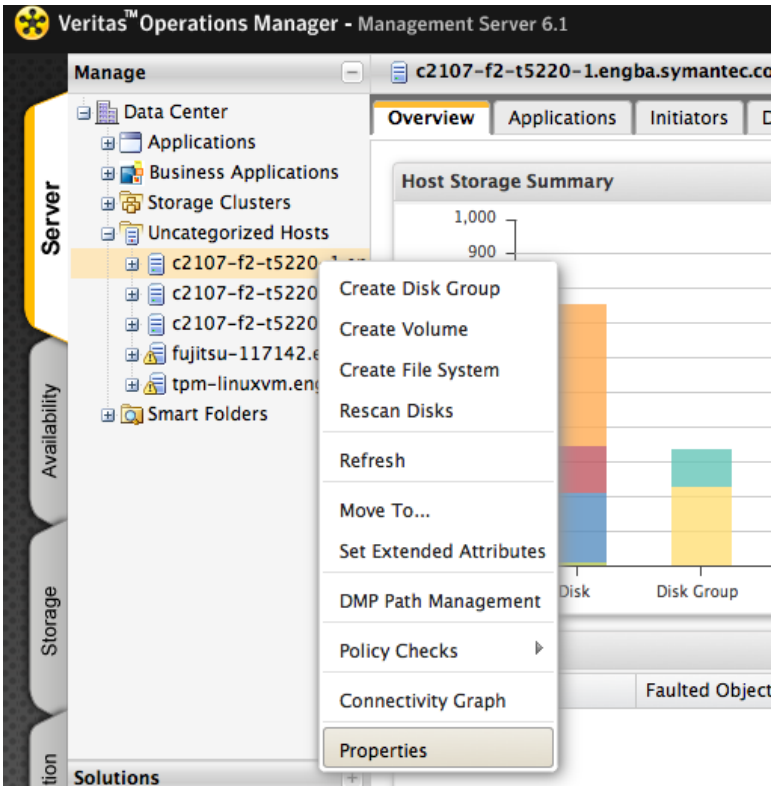
1. To view the cache details in the Management Server console, go to the **Server** perspective, and expand **Manage** in the left pane.



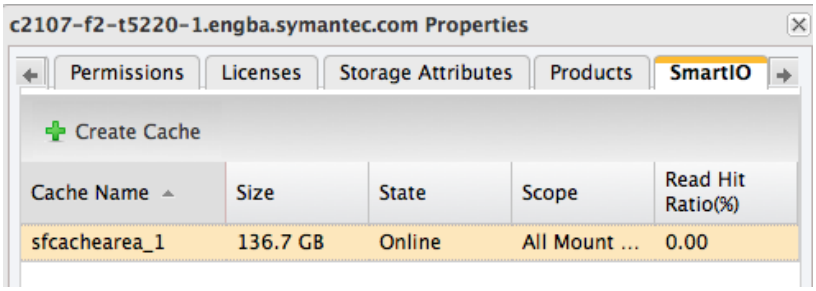
2 Expand the **Organization (if configured)** or **Uncategorized Hosts** icons to locate the host.



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



4 Click the **SmartIO** tab.



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	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: • Volume level read cache: This block based read cache can be used for the applications that store their data on raw volumes. • File system level read cache: This is a file-level read cache for the 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 IOs. 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 on the SORT website.

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

Task	Command
Explicitly make a cache area available.	`sfcache online [cachearea_name|dg/volname|`
Displaying cache information	
Display the cached volumes and their cache usage. (VxVM only)	`sfcache list [-l] [cachearea_name|dg/volname]`
Display the cached file system and its cache usage. (VxFS only)	`sfcache [-r] [mount_point|file|dir]`
Display cache statistics, including cache hit rate, misses, average read, and write latencies. (VxFS only)	`sfcache stat [-l] [-r] mount_point`
Display cache statistics, including cache hit rate, misses, average read, and write latencies. (VxVM only)	`sfcache stat [-l] [-r] [-i time ] [cachearea_name|dg/volname]`
Display the amount of free space in the devices that are already provisioned for caching.	`sfcache maxsize [daname ...]`
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.	`sfcache resize [daname...] {newsize|maxsize} cachearea_name`
Working with files, directories, data objects, and devices	
Load the specified file into the cache area.	`sfcache load [-r] [-o sync] {file1|dir1} [file2|dir2...]`
Mark a file or directory to be held in cache until the file or directory is deleted, truncated, or unpinned. Only applies to VxFS caching.	`sfcache pin [-o load] [-r] {file1|dir1} [file2|dir2...]`

Task	Command
Remove the file or directory from the pinned state. Only applies to VxFS caching.	`sfcache unpin [-r] {file1|dir1} [file2|dir2...]`
Remove the cached contents for the specified file system. This command frees up dead space in the cache. Only applies to VxFS caching.	`sfcache purge {mount_point|fsuuid}`
Restore read or write access to files that are missing write-back data. Use this command for recovery. (VxFS only)	`sfcache restore-access [-r] {mount_point|file|dir}`
Remove the device or devices from use for caching.	`sfcache rmdev daname[...]`
Control caching for the specified VxVM data object. (VxVM only)	`sfcache set [--pause|--resume] dg/volname`
Control caching for the specified VxFS data object. (VxFS only)	`sfcache set [-r] mode={nocache|read|writeback} {file1|dir1} [file2|dir2...]`

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.

Field	Description
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 do 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 Solaris.
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

- 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 Symantec Storage Foundation and High Availability Solutions 6.2 release notes.