VERITAS™

VERITAS Storage Checkpoints

AN UNDO BUTTON FOR USERS AND ADMINISTRATORS

TABLE OF CONTENTS

Executive Summary

VERITAS Storage Checkpoint technology provides the administrator with the equivalent of an undo button. File corruption errors, such as a faulty patch to a binary file, are simply and speedily recovered by restoring the original file's checkpointed contents. Storage Checkpoints also interact with VERITAS Netbackup to offer disk-based backup functionality: Disk-based backup is a high-performance backup alternative increasingly popular as dollar-per-gigabyte costs of disk space continue to decline.

This paper examines the business and technical advantages of Storage Checkpoints and documents best practices for effective storage checkpoint use. A detailed appendix includes real-world scripts developed by administrators to leverage the Storage Checkpoint functionality.

Storage Checkpoints

VERITAS Storage Checkpoints allow administrators to create point-in-time copies of the VERITAS File System (VxFS). The file system's checkpointed contents are available to administrators for fast recovery of corrupted files. Using sophisticated copy-on-write technology, the checkpoint process stores only changed data blocks. This technique streamlines disk and I/O processing and permits an unlimited number of checkpoints to be retained.

Copy-on-write technology is exclusive to VERITAS and is activated the moment a checkpoint command is issued. All file system writes, subsequent to the checkpoint, are analyzed and if data blocks change because of the write the old contents of the blocks are noted and stored in a special checkpoint partition housed in the free space of the file system. If a recovery is necessary the original contents of the file are recreated by combining the existing file and the copy-on-write record of all changed blocks.

Business Advantages

System administrators use Storage Checkpoints as an Undo Button to quickly recover a clean copy of a file when a problem occurs. Consider the following scenario. An Administrator is tasked with applying patches to an application's binary file. To safeguard recovery of the binary file, in the event of a problem, the administrator creates a Storage Checkpoint. As soon as the create command is issued the Storage Checkpoint driver begins monitoring the file system for changed data blocks. When the driver encounters changes it copies modified data to the Storage Checkpoint partition. If, for whatever reason, the administrator's upgrade to the application binary file fails, all changes to the file can be quickly undone by recovering the file back to the checkpoint. Restoring from the Storage Checkpoint is faster than the alternative: laboriously having to find the original application binary and reapply any required patches. And, if the administrator's upgrade is successful, the checkpoint data can be discarded. By simplifying the backup and recovery steps of the administrator's upgrade procedure, Storage Checkpoints reduce administrator stress and improve system uptime.

Storage Checkpoints are not just for use by administrators. The ability to quickly recover lost file data can improve the end-user experience of file and mail servers. Although not a replacement for traditional backup processing, checkpoints give users self-service access to file and email recovery. A regular morning checkpoint of the file system will allow users to quickly recover from unintended deletion of a file or email. Users gain fast access to recovery processing without requiring the assistance of the administrator.

Checkpoint functionality is accessed through familiar UNIX commands. This eliminates the need for training and offers users and administrators a feel-good level of familiarity with the backup and recovery process. The minimal setup requirements, compared to alternative backup and recovery techniques, mean that benefits are felt immediately after the first checkpoint is initiated.

In addition to fast file recovery, the combination of Storage Checkpoints and VERITAS NetBackup™ Advanced Client gives the administrator the ability to convert traditional backup processing into disk-based backup: The checkpoint

becomes the backup media. A recent VERITAS study found that NetBackup restored data up to 95% faster by using checkpoints when compared to tape. Faster restore performance translates directly into more uptime.

Technical Advantages

Storage Checkpoints look like an exact image of the underlying VERITAS File System™, frozen in time at the moment the checkpoint was created. This frozen image can be accessed in exactly the same manner as the original file system.

LUN-based snapshots offer an alternate method of point-in-time backup, but they are significantly more cumbersome to implement than Storage Checkpoints. Performing a LUN snapshot is a multi-step process. A LUN, serving as a mirror of the original, must be selected and provisioned. The two LUNs must then be synchronized to create a mirrored pair. Finally, the mirror is split, to give a working original and a backup LUN copy. Each LUN snapshot must then be managed, by the administrator, and made available whenever needed for file recovery. Conversely, space for a Storage Checkpoint is dynamically allocated from the free space resources of the file system, requiring no administrator intervention. And, the checkpoint is created by issuing a single command.

Storage Checkpoints also benefit from their privileged location in the I/O stack. Consider the process of creating a new file. A LUN-based snapshot approach will copy all data blocks to the snapshot LUN, whether they are empty or not. The new file's contents will then be written to the original LUN and the mirror. Residing at the File System layer, a Storage Checkpoint knows that the data blocks that are about to contain the new file's content were originally empty, and the process is smart enough not to copy the empty blocks to the checkpoint. Only modified data blocks are stored. For applications like NFS, eliminating the redundant movement of empty data blocks provides a significant performance advantage. It also requires less storage capacity.

By default, Storage Checkpoints store changed data blocks, and pointers to the original unchanged data, in the free space of the file system. This technique minimizes the need for additional storage capacity, but, as a consequence, checkpoints do not protect against physical media failure. Storage Checkpoints are not a replacement for traditional backup processing. In addition, checkpoint data is not available for use in off-host processing, unless allocated to a node operating within a VERITAS Cluster File System configuration. With VERITAS Storage Foundation 4.0 Storage Checkpoint data can also be hosted on a volume or LUN other than the originating file system's volume or LUN.

An Overview of Storage Checkpoint Best Practices

Availability

Storage Checkpoints are available for Solaris, AIX, and Linux with VxFS 3.4 and higher. HP-UX requires VxFS 3.5 or higher. Checkpoints for databases were available in prior releases through VERITAS Database Edition products.

Each subsequent release of the Storage Checkpoint product has improved performance and added features. VxFS 4.0 continues this trend, adding more space management tools, including quotas and volume placement, reducing checkpoint create time to a few seconds, broadening restore capabilities, and allowing checkpoints to be created from a common graphical user interface (GUI).

Installation

Storage Checkpoint binaries are included with VxFS (also known as JFS, or OJFS on HP-UX). If VxFS is already installed on a server, activating the Storage Checkpoint code is simply a question of adding a license key. No additional installations, reboots, or configuration steps are required.

Naming

It is helpful to give each checkpoint a meaningful name. Although date and time details for each checkpoint can be found in the file system, including this information in the name can help users and administrators quickly identify an appropriate checkpoint when restoring deleted or damaged files.

Space Management

Storage Checkpoint data is stored in the free space of the file system, relieving the administrator of the burden of allocating space for each backup. Quotas can be set, however, to limit how much of a file system's free space is available to the checkpoint process.

Mounting

When choosing a mount point for a checkpoint, consider whether the file system will be exported via NFS. While it is possible to mount another file system in a directory already exported via NFS, the new mount point will not be exported in accordance with NFS standards.

Sharing the contents of a checkpoint with NFS clients requires the creation of a new mount point. Using the automounter masks much of this complexity. With NIS and NIS+ automatic updating of the automounter configuration, from a script, will transparently publish the new mount point to all NIS and NIS+ clients. This gives users access to the latest checkpoint without having to perform a manual mount.

Clustering

If Storage Checkpoints are used in a failover environment, all checkpoints for a given file system must be unmounted before fail over can occur. VERITAS Cluster Server™ 4.1 will automate the unmount process. For earlier versions of Cluster Server, or with other clustering applications, it is recommended to customize the appropriate agent.

Purchasing

Storage Checkpoints are included in the following VERITAS products and options:

VERITAS Storage Foundation Enterprise™

VERITAS Storage Foundation Enterprise for Oracle™

VERITAS Storage Foundation Enterprise for DB2™

VERITAS Storage Foundation Cluster File System™ (Formerly VERITAS SANPoint Foundation Suite™)

VERITAS Storage Foundation for Oracle RAC™

VERITAS FlashSnap™

Older products with Storage Checkpoints include:

VERITAS Database Edition for Oracle™

VERITAS Database Edition Advanced Cluster for Oracle RAC™

VERITAS ServPoint NAS

Conclusion

VERITAS Storage Checkpoints offer administrators an exceptionally valuable tool for their data recovery toolbox. The speedy nature of checkpoint processing, minimal storage requirements, and ability to provide end-users with self-service recovery functionality will significantly ease the task of administration.

Technical Appendix

Using Checkpoints

The following file system will be used in this example:

Mount Point : /home
Device : /dev/vx/dsk/homevol-01

Note: the syntax for AIX, HP-UX, Linux and Solaris are identical, however each OS uses a different convention for the file system switch out: -V for AIX, -F for HP-UX and Solaris, and –t for Linux. In the following examples, -F is used.

1. Configure a server for checkpoints. Assuming VxFS is already installed, the only step is to add a license key that enables checkpoints:

```
# vxlicinst -k AB12-CDEF-GHIJ-K3LM-NOPQ-RS45-TUVW-6789-0XYZ
```

2. Create a checkpoint of /home named “monday_ckpt”:

```
# fsckptadm create monday_ckpt /home
```

3. Mount the checkpoint “monday_ckpt” on directory /home/checkpoints/Monday:

```
# mount -F vxfs -o ckpt=monday_ckpt /dev/vx/dsk/homevol-01:monday_ckpt \ /home/checkpoints/monday
```

4. An end user accidentally deletes a file in home directory. Restores from this checkpoint:

```
$ rm mynovel.txt
$ pwd
/home/users/melissa
$ cd /home/checkpoints/monday/users/melissa
$ ls -l
-rw-r--r--    1 melissa    staff  14910  Jul  4    17:09   mynovel.txt
$ cp mynovel.txt /home/users/melissa
$ cd /home/users/melissa
$ ls –l
-rw-r--r--    1 melissa    staff  14910  Jul  4    18:21   mynovel.txt[1]
```

5. Root user accidentally ran rm –r *. Restore the file system from the checkpoint:

```
# umount /home/checkpoints/monday
# umount /home
# fsckpt_restore –l /dev/vx/dsk/homevol-01

Select checkpoint for restore operation or enter <Return> to list
checkpoints: monday_ckpt
```

[1] Note that the restored file’s timestamps are updated to the time of the restore.

```
Restore the file system from checkpoint monday_ckpt ? (ynq) y
File system restored from monday_ckpt

# mount -F vxfs /dev/vx/dsk/homevol-01 /home
```

Perl Script for checkpoint management (used with cron)

Please note this Perl script has not been tested on all platforms. Some of the unmount commands may need to be changed to be compatible with individual Linux distributions.

```
#!/usr/bin/perl
#
# (c) 2003, VERITAS Software, all rights reserved.
# Redistribution and use in source and binary forms, with or without modification,
# are permitted provided that the following conditions are met:
#
# Redistributions of source code must retain the above copyright notice, this list
# of conditions and the following disclaimer.
# The name of the author may not be used to endorse or promote products derived from
# this software without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY
# EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
# OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
# SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
# TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
# OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
# CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
# ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
# DAMAGE.
#
#
# This script takes a storage checkpoint nightly for the assigned file systems
# the writes them to a standard location.

#note that num_fs must equal the number of file systems
#and that the mnt_point and mnt_special must refer to the same file system!
$num_fs = 1;
$mnt_point[0] = "/myfs";
$mnt_special[0] = "/dev/vx/dsk/mydg/myvol";

$ckpt_dir = "/checkpoints/";
$ckpt_name = "_checkpoint";

#choose your platform
#Solaris: $Today = `/usr/bin/date '+%A'`;
#Linux: $Today = `/bin/date '+%A'`;
$Today = `/bin/date '+%A'`;

#need to strip off the carriage return
$Today =~ s/\n//g;
#convert to all lowercase
$today   = "\L$Today";
```

```
$cmdDir = &getDir;
$flag = &getFlag;

#main
for($x=0; $x<$num_fs; $x++) {
#Solaris: system("/usr/lib/fs/vxfs/umount $mnt_point[$x]$ckpt_dir$today");
#Linux: system("/bin/umount $mnt_point[$x]$ckpt_dir$today");
system("/bin/umount $mnt_point[$x]$ckpt_dir$today");

system("$cmdDir/fsckptadm -s remove $today$ckpt_name $mnt_point[$x]");
system("$cmdDir/fsckptadm -r create $today$ckpt_name $mnt_point[$x]");

#choose your system
#Solaris: system("/etc/fs/vxfs/mount $flag vxfs -o ckpt=$today$ckpt_name $mnt_special[$x]:$today$ckpt_name
$mnt_point[$x]$ckpt_dir$today");
#Linux: system("$cmdDir/mount $flag vxfs -o ckpt=$today$ckpt_name $mnt_special[$x]:$today$ckpt_name
$mnt_point[$x]$ckpt_dir$today");
system("$cmdDir/mount $flag vxfs -o ckpt=$today$ckpt_name $mnt_special[$x]:$today$ckpt_name
$mnt_point[$x]$ckpt_dir$today");
}
#currently the mount command doesn't work for Linux - fails with unsupported option!!!

###############
# Subroutine to get operating system-specific file system 'flag'

sub getFlag {
        $os = `/usr/bin/uname`;
        if ($os = "SunOS" or $os = "HP-UX") {
                $flag="-F";
        } elsif ($os = "AIX") {
        $flag="-V";
        } elsif ($os = "linux" or $os = "Linux") {
        $flag="-t";
        } else {
                die "\n This script does not support $os\n";
        }
        $flag;
}
###############

###############
# Subroutine to get the command location
# note that with VxFS 3.5 and higher (3.4 for AIX and Linux), symlinks
# to all commands are located in /opt/VRTS/bin, so $cmdDir can be replaced with "/opt/VRTS/bin".

sub getDir {
        $os = `/usr/bin/uname`;
        if ($os = "SunOS") {
                $cmdDir="/opt/VRTSvxfs/sbin";
        } elsif ($os = "HP-UX") {
                $cmdDir="/usr/sbin";
        } elsif ($os = "linux" or $os = "Linux" or $os = "AIX") {
        $cmdDir="/opt/VRTS/bin";
        } else {
                die "\n This script does not support $os\n";
```

```
        }
    $cmdDir;
}
###############
```

This cron entry will trigger the Perl script :

```
1 3 * * * /usr/bin/perl /usr/bin/scripts/ckpt_taker > /dev/null 2>&1
```

Checkpoint Management Script

```
#!/bin/sh
#
# (c) 2003, VERITAS Software, all rights reserved.
# Redistribution and use in source and binary forms, with or without modification,
# are permitted provided that the following conditions are met:
#
# Redistributions of source code must retain the above copyright notice, this list
# of conditions and the following disclaimer.
# The name of the author may not be used to endorse or promote products derived from
# this software without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
# CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
# INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
# MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
# DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
# CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
# SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT # NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
# LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
# HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
# CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
# OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
# EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#

MAILIST="example@veritas.com"

FSLIST=" \
       /vxfshome:3 \
       /lhome:3 \
       /lhome/rader:3 \
       /lhome/bkumares:6 \
       /lhome/tyt:6 \
       /lhome/devsip:6 \
"

MAILCMD="mailx"
FSCKPTADM="/usr/lib/fs/vxfs/fsckptadm"
CUMLOGDIR="/var/adm/bkup"
CUMLOG="${CUMLOGDIR}/log"
TMPLOG="/tmp/bkuplog_$$"
LOGPLAIN=0
```

```
LOGERROR=1
LOGWARN=2
LOGINFO=3

VALID_RELEASES=" \
      VxFS 3.4,REV=PATCH02 \
"

CMDNM="`basename $0`"

#
# Start the log.
#
logstart()
{
      rm -rf $TMPLOG
      logmsg $LOGINFO "$CMDNM Started on `date`"
}

#
# End the log.
#
logend()
{
      logmsg $LOGINFO "$CMDNM Ended on `date`"
      echo "\n\n" >> $TMPLOG
      cat $TMPLOG >> $CUMLOG
      #cat $TMPLOG
      $MAILCMD -s "Daily Checkpoint Results" $MAILIST < $TMPLOG
      rm -rf $TMPLOG
}

#
# Log output of a command.
#
logcmd()
{
      logmsg $LOGINFO "================== cmdlog start ==================="
      echo "$1" >> $TMPLOG 2>&1
      time $1 >> $TMPLOG 2>&1
      ret=$?
      logmsg $LOGINFO "================== cmdlog end ==================="
      return $ret
}
#
# First argument is the type of message:
#           $LOGPLAIN has no prefix to message.
#           $LOGERROR has ERROR as prefix and implies exit.
#           $LOGWARN has WARNING as prefix.
#           $LOGINFO has INFO as prefix.
# Second argument is the message to add to the log.
#
logmsg()
{
      msgtype="$1"
      msg="$2"
```

```
        case $msgtype in
        $LOGPLAIN)
                echo "$msg" >> $TMPLOG
                ;;
        $LOGERROR)
                echo "ERROR: $msg" >> $TMPLOG
                ;;
        $LOGWARN)
                echo "WARNING: $msg" >> $TMPLOG
                ;;
        $LOGINFO)
                echo "INFO: $msg" >> $TMPLOG
                ;;
        *)
                echo "INVALID MESSAGE TYPE: $msg" >> $TMPLOG
                ;;
        esac

        if [ $msgtype -ne $LOGERROR ]; then
                return
        fi

        #
        # end the log and exit
        #
        logend
        exit 1
}

#
# Check system/process capabilities.
#
sys_check()
{
        #
        # Root permission is needed.
        #
        id | grep root > /dev/null 2>&1
        if [ $? -ne 0 ]; then
                logmsg $LOGERROR "$CMDNM requires root privilage"
        fi

        #
        # Check vxfs release installed on machine. It should be 3.4
        # update 2 and up.
        #
        modinfo | grep "vxfs" > /dev/null 2>&1
        if [ $? -ne 0 ]; then
                logmsg $LOGERROR "vxfs is not installed on `hostname`"
        fi
        validrel=0
        for vrel in $VALID_RELEASES; do
                modinfo | grep "$vrel" > /dev/null 2>&1
                if [ $? -eq 0 ]; then
                        validrel=1
```

```
                        break
                fi
        done
        if [ $validrel -eq 0 ]; then
                logmsg $LOGERROR "No valid vxfs release was found on `hostname`"
        fi

        #
        # Check license for Storage Checkpoints.
        #
        vxlicense -p | grep VXCKPT > /dev/null 2>&1
        if [ $? -ne 0 ]; then
                logmsg $LOGERROR \
                     "Storage Checkpoint license is not available on `hostname`"
        fi
}

#
# Skip Saturdays and Sundays.
#
skip_weekend()
{
        date | egrep "^Sat|^Sun" > /dev/null 2>&1
        if [ $? -eq 0 ]; then
                logmsg $LOGERROR "Skipping weekend"
        fi
}

#
# Check file system capabilities.
#
fs_check()
{
        thisfs=$1

        if [ ! -d $thisfs ]; then
                logmsg $LOGWARN "Directory $thisfs does not exist"
                return 1
        fi

        mount | grep "^$thisfs " > /dev/null 2>&1
        if [ $? -ne 0 ]; then
                logmsg $LOGWARN "$thisfs is not a mountpoint"
                return 1
        fi
        rdev=`mount | grep "^$thisfs " | awk '{print $3}'`
        fstyp $rdev | grep vxfs > /dev/null 2>&1
        if [ $? -ne 0 ]; then
                logmsg $LOGWARN "$thisfs is not a VxFS file system"
                return 1
        fi
        version=`fstyp -v $rdev | grep version | awk '{print $4}'`
        if [ $version -ne 4 ]; then
                logmsg $LOGWARN "$thisfs is not a VxFS version 4 file system"
                return 1
        fi
```

```
        return 0
}

#
# Remove last checkpoint if needed.
#
remove_last()
{
        thisfs=$1
        thisnkeep=$2

        nckpt=`$FSCKPTADM list $thisfs | grep ctime | wc -l`
        if [ $nckpt -lt $thisnkeep ]; then
                return
        fi
        lastone=`$FSCKPTADM list $thisfs | grep ":$" | tail -1 | \
                 awk -F: '{print $1}'`
        logcmd "$FSCKPTADM -v info $lastone $thisfs"
        # logcmd "$FSCKPTADM -s remove $lastone $thisfs"
        logcmd "$FSCKPTADM remove $lastone $thisfs"
        if [ $? -ne 0 ]; then
                logmsg $LOGWARN "Failed to remove $lastone on $thisfs"
                return 1
        fi
        return 0
}

#
# Create a new checkpoint
#
create_ckpt()
{
        thisfs=$1

        ckptnm=`date '+%a_%m/%d/%y@%H%M'`
        logcmd "$FSCKPTADM -rv create $ckptnm $thisfs"
        if [ $? -ne 0 ]; then
                logmsg $LOGWARN "Failed to create $lastone on $thisfs"
                return 1
        fi
        return 0
}

#
#  M  A  I  N
#
mkdir $CUMLOGDIR > /dev/null 2>&1
logstart
skip_weekend
sys_check

for ent in $FSLIST; do
        fs="`echo $ent | awk -F: '{print $1}'`"
        nkeep="`echo $ent | awk -F: '{print $2}'`"
        fs_check $fs
        if [ $? -ne 0 ]; then
```

```
                continue
        fi
        remove_last $fs $nkeep
        if [ $? -ne 0 ]; then
                continue
        fi
        create_ckpt $fs
done
logend
```

VERITAS Software Corporation

Corporate Headquarters
350 Ellis Street
Mountain View, CA 94043
650-527-8000 or 866-837-4827

For additional information about VERITAS Software, its products, or the location of an office near you, please call our corporate headquarters or visit our Web site at www.veritas.com.