Oracle 9i for UNIX: Cloning Database Using NetApp Filer Flexible Volumes in NAS and SAN Environments

Sunny Ng, Network Appliance | November 2004 | TR-3355

TECHNICAL REPORT

Network Appliance, a pioneer and industry leader in data storage technology, helps organizations understand and meet complex technical challenges with advanced storage solutions and global data management strategies.

Abstract

This document describes the cloning process of Oracle 9i™ databases using Data ONTAP® flexible-volume clone feature and Oracle® backup/recovery techniques. Cloning processes under both NFS and FCP environments are considered. Procedures for validating the cloned data are also discussed.

Table of Contents

1. Introduction

Database administrators and developers clone databases for various reasons, such as updating test or development environments from production or switching the production database to the clone in case of an emergency. Cloning databases usually involves both backup and recovery procedures. The clone can exist on the same host as the production database or on a different host.

This paper discusses the cloning of Oracle9i databases using the NetApp Data ONTAP flexible-volume clone feature under both NAS and SAN environments. Specifically, this paper covers the following topics:

(a) Creating Oracle9i a production database and its components in NetApp filer flexible volumes
(b) Performing full and partial clones of the production database in both NAS and SAN environments using the NetApp flexible-volume clone feature, Oracle User-Managed and RMAN recovery techniques
(c) Methods to validate the cloned database

It is important to note that the cloned database is created locally in the same filer aggregate that accommodates the original production database. Remote cloning such as using NetApp SnapMirror® technology is not covered here. Interested readers can refer to Ref. [1] for the related topic.

For the purpose of this paper, *clone* will refer to the cloned database, *production* to the source or production database, and *Oracle* to Oracle9i database products. The terms *source* and *production* are used interchangeably.

2. Requirements and Assumptions

It is assumed that the reader is familiar with the administrative commands and operations of the NetApp filer, Data ONTAP, Oracle9i, and the UNIX® operating system. It is further assumed that the reader has basic knowledge of the NAS and SAN technologies.

The NetApp filer must be loaded with the Data ONTAP version that supports the flexible-volume clone feature. At the time of writing, the version is "Anchorsteam". The filer must also have NFS and FCP capabilities. The Oracle database server is assumed to have been installed on a UNIX machine and to be ready for creating database objects.

In the database cloning process, some of the database objects and transactions are created using the Benchmark Factory TPCC tool [3]. However, the reader is not required to know the TPCC tool since this paper focuses on the cloning procedures and not on how data are created.

The sample scripts and commands in this paper assume the following:

The name of the filer is "filer".
The name of the Oracle server is "oracle9".
The name of the production instance is "DB1".
The name of the clone instance is "DB1CLONE".
The names of filer aggregates are "aggr1", "aggr2", and "aggr3".
The filer flexible volumes to store the production database are "dbuserdata", "sysdata", "log1", "log2", and "archivedlog".
The filer flexible volumes to store the clone are "userdataClone", "sysdataClone", "log1Clone", "log2Clone", "archClone".
The mount point of the production is "`/export/home/DB1`".
The mount point of the clone is "`/export/home/db1clone`".

The tests used an Oracle 9.2.0.1.0 database server that ran on Sun Solaris™ 8. The filer was an F880 loaded with the Data ONTAP - Anchorsteam release.

3. Network and Storage Infrastructure

The current database cloning tests were conducted in the System Test Simcity Customer Simulation Lab facility [2]. Figure 1 shows a schematic of the lab facility. The Simcity lab is equipped with a Gigabit Ethernet network, multiple SAN channels, various models of NetApp filers and NearStore® platforms, a WAN simulator, a server farm, and a client farm, The server farm comprises mostly non NetApp applications such as Oracle 9i Server, Clear Case™, Microsoft® Exchange™, NIS, DNS, and Microsoft Domain Controller®, etc. The client farm includes more than a hundred of UNIX and Windows® hosts; NetApp applications such as DataFabric® Manager and Snap products; and non NetApp applications like Benchmark Factory™ [3], ORASIM® (Oracle Server simulator) [9], and SQLSIM® (a Microsoft SQL Server simulator). The Simcity lab configuration is designed to be flexible in order to simulate various customer network and storage infrastructure configurations accurately.

NetApp has shipped thousands of multiprotocol filers to customers in different industries. Customers have benefited from the filer performance, reliability, and stability for their storage operations. Because these customer environments are diverse, it is crucial to have a lab facility that can provide flexible configurations to simulate customer environments for product testing.

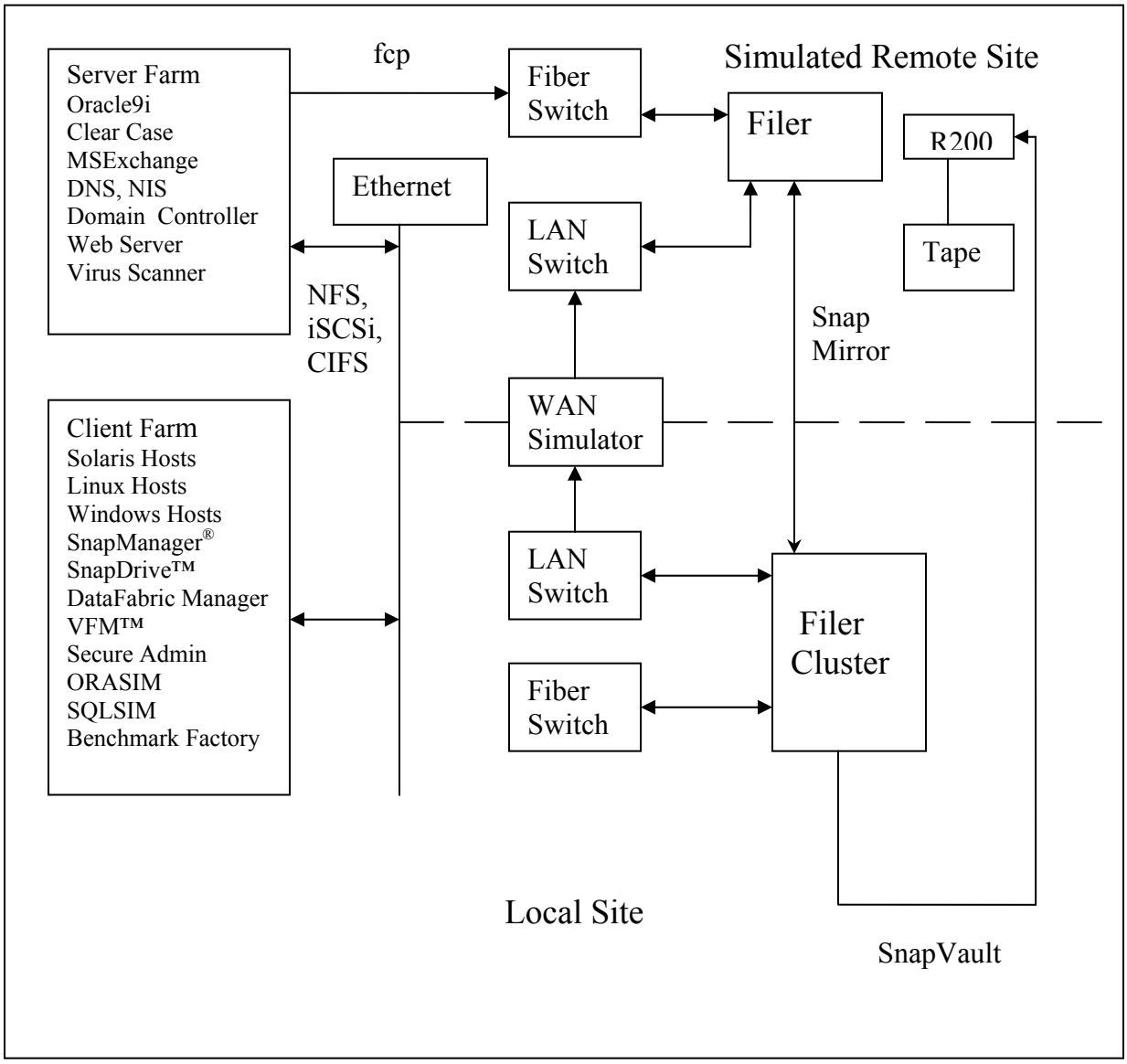


Figure 1) A Schematic of Simcity Customer Simulation Lab.

4. Creating a Production Database on a Filer in a NAS Environment

This section discusses the creation of a production database and its components and of the filer aggregates and flexible volumes that store that database. The tools that generate database data and objects such as tables, indexes, and procedures are also described here.

4.1 Creating Filer Aggregates and Flexible Volumes

Before creating the database, we first create aggregates and flexible volumes on the filer. An aggregate is a RAID-level physical pool of storage. A flexible volume is a logical storage container inside an aggregate. A flexible volume can be as small as a few megabytes and as large as the aggregate. There are many advantages to storing database components in flexible volumes. To name a few, (a) a distinct volume can be created for a distinct dataset; (b) the volume size can easily be tailored to meet the component space requirement; and (c) flexible backup of any selected database component can be made.

In the tests, two aggregates—"aggr1" and "aggr2"—were created. The aggregate "aggr1" was assigned 10 disks and "aggr2" 12 disks. Distinct volumes were then created in the aggregates for distinct datafiles. The online redo logfiles were multiplexed in different aggregates to protect transactions from media failure. Specifically, the volume "dbuserdata" was created for storing user tablespaces datafiles; "archivedlog" for archived redo logfiles; "sysdata" for system, undo, and temporary tablespaces datafiles; and "log1" and "log2" for online redo logfiles.


```
filer> aggr create aggr1 10
filer> aggr create aggr2 12
filer> vol create dbuserdata aggr2 200g
filer> vol create sysdata aggr1 20g
filer> vol create log1 aggr1 20g
filer> vol create log2 aggr2 20g
filer> vol create archivedlog aggr2 100g
```

Now, mount the filer volumes onto the Oracle server "oracle9". Note that the NFS mount uses TCP. In general, UDP has less protocol overhead and consequently better performance, provided that there is a reliable and dedicated connection between the filer and the Oracle server. However, the test environment lacks such a dedicated connection and TCP is therefore used.

```
oracle9# mount -o hard,intr,suid,vers=3,proto=tcp,llock,forcedirectio,
rsize=32768,wsize=32768, filer:/vol/dbuserdata /export/home/DB1/user_data
oracle9# mount -o hard,intr,suid,vers=3,proto=tcp,llock,forcedirectio,
rsize=32768,wsize=32768,filer:/vol/dbsysdata /export/home/DB1/sys_data
oracle9# mount -o hard,intr,suid,vers=3,proto=tcp,llock,forcedirectio,
rsize=32768,wsize=32768, filer:/vol/log1 /export/home/DB1/log_1
oracle9# mount -o hard,intr,suid,vers=3,proto=tcp,llock,forcedirectio,
rsize=32768,wsize=32768, filer:/vol/log2 /export/home/DB1/log_2
oracle9# mount -o hard,intr,suid,vers=3,proto=tcp,llock,forcedirectio,
rsize=32768,wsize=32768, filer:/vol/archivedlog /export/home/DB1/archlog
```

Add the following entries to the file `/etc/vfstab` on the host "oracle9" for an automatic remount of the filer volumes at the host reboot.

```
filer:/vol/dbuserdata - /export/home/DB1/user_data nfs –
Yes hard,intr,suid,vers=3,proto=tcp,rsize=32678,wsize=32678,
llock,forcedirectio
filer:/vol/dbsysdata - /export/home/DB1/sys_data nfs –
yes hard,intr,suid,vers=3,proto=tcp,rsize=32678,wsize=32678,
llock,forcedirectio
filer:/vol/log1 - /export/home/DB1/log_1 nfs –
yes hard,intr,suid,vers=3,proto=tcp,rsize=32678,wsize=32678,
llock,forcedirectio
filer:/vol/log2 - /export/home/DB1/log_2 nfs –
yes hard,intr,suid, vers=3,proto=tcp,rsize=32678,wsize=32678,
llock,forcedirectio
filer:/vol/archivedlog - /export/home/DB1/archlog nfs - yes
hard,intr,suid,vers=3,proto=tcp,rsize=32678,wsize=32678,
llock,forcedirectio
```

4.2 Creating Production Database Components on Filer Volumes

We are now ready to create the production database at the Oracle home, which is the recommended place for the filer volume [5, 10]. We name this database "DB1" and create it using the following SQL script. The database can also be created by using DBCA tool.


```
connect / as SYSDBA
set echo on
spool /export/home/OraHome1/assistants/dbca/logs/CreateDB.log
startup nomount pfile="/export/home/OraHome1/admin/DB1/pfile/initDB1.ora";
CREATE DATABASE DB1
MAXINSTANCES 1
MAXLOGHISTORY 1
MAXLOGFILES 5
MAXLOGMEMBERS 3
MAXDATAFILES 100
DATAFILE '/export/home/DB1/sys_data/system01.dbf' SIZE 250M REUSE
AUTOEXTEND ON NEXT 10240K MAXSIZE UNLIMITED EXTENT MANAGEMENT LOCAL
DEFAULT TEMPORARY TABLESPACE TEMP TEMPFILE
'/export/home/DB1/sys_data/temp01.dbf' SIZE 200M REUSE
UNDO TABLESPACE "UNDOTBS1" DATAFILE
'/export/home/DB1/sys_data/undotbs01.dbf' SIZE 200M AUTOEXTEND ON
NEXT 5120K MAXSIZE UNLIMITED,
'/export/home/DB1/sys_data/undotbs02.dbf' SIZE 200M AUTOEXTEND ON
NEXT 5120K MAXSIZE UNLIMITED,
'/export/home/DB1/sys_data/undotabs03.dbf'SIZE 200M AUTOEXTEND ON
NEXT 5120K MAXSIZE UNLIMITED
CHARACTER SET WE8ISO8859P1
NATIONAL CHARACTER SET AL16UTF16
LOGFILE
GROUP 1 ('/export/home/DB1/log_1/redo01a.log',
'/export/home/DB1/log_2/redo01b.log') SIZE 102400K,
GROUP 2 ('/export/home/DB1/log_1/redo02a.log',
'/export/home/DB1/log_2/redo02b.log') SIZE 102400K,
GROUP 3 ('/export/home/DB1/log_1/redo03a.log',
'/export/home/DB1/log_2/redo03b.log') SIZE 102400K;
spool off
exit;
```

Additional tablespaces are then created and added to DB1. For instance, the tablespace "USERS" is created and assigned 40GB of space for storing user data. This is illustrated in the following SQL script.

```
connect / as SYSDBA
set echo on
spool /export/home/OraHome1/assistants/dbca/logs/CreateDBFiles.log
CREATE TABLESPACE "INDX" LOGGING DATAFILE
'/export/home/DB1/sys_data/indx01.dbf' SIZE 25M REUSE
AUTOEXTEND ON NEXT 1280K MAXSIZE UNLIMITED EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;
CREATE TABLESPACE "TOOLS" LOGGING DATAFILE
'/export/home/DB1/sys_data/tools01.dbf' SIZE 10M REUSE
AUTOEXTEND ON NEXT 320K MAXSIZE UNLIMITED EXTENT MANAGEMENT LOCAL SEGMENT
SPACE MANAGEMENT AUTO;
CREATE TABLESPACE "USERS" LOGGING DATAFILE
    '/export/home/DB1/user_data/users01.dbf' SIZE 4000M REUSE AUTOEXTEND
ON NEXT 1280K MAXSIZE UNLIMITED,
    '/export/home/DB1/user_data/users02.dbf' SIZE 4000M REUSE AUTOEXTEND
ON NEXT 1280K MAXSIZE UNLIMITED,
    '/export/home/DB1/user_data/users03.dbf' SIZE 4000M REUSE AUTOEXTEND
ON NEXT 1280K MAXSIZE UNLIMITED,
    '/export/home/DB1/user_data/users04.dbf' SIZE 4000M REUSE AUTOEXTEND
ON NEXT 1280K MAXSIZE UNLIMITED,
    '/export/home/DB1/user_data/users05.dbf' SIZE 4000M REUSE AUTOEXTEND
ON NEXT 1280K MAXSIZE UNLIMITED,
    '/export/home/DB1/user_data/users06.dbf' SIZE 4000M REUSE AUTOEXTEND
ON NEXT 1280K MAXSIZE UNLIMITED,
    '/export/home/DB1/user_data/users07.dbf' SIZE 4000M REUSE AUTOEXTEND
ON NEXT 1280K MAXSIZE UNLIMITED,
    '/export/home/DB1/user_data/users08.dbf' SIZE 4000M REUSE AUTOEXTEND
ON NEXT 1280K MAXSIZE UNLIMITED,
    '/export/home/DB1/user_data/users09.dbf' SIZE 4000M REUSE AUTOEXTEND
ON NEXT1280K MAXSIZE UNLIMITED,
    '/export/home/DB1/user_data/users10.dbf' SIZE 4000M REUSE AUTOEXTEND
ON NEXT 1280K MAXSIZE UNLIMITED EXTENT MANAGEMENT LOCAL SEGMENT SPACE
MANAGEMENT AUTO;
CREATE TABLESPACE "XDB" LOGGING DATAFILE
'/export/home/DB1/sys_data/xdb01.dbf' SIZE 20M REUSE AUTOEXTEND
ON NEXT640K MAXSIZE UNLIMITED EXTENT MANAGEMENT LOCAL SEGMENT SPACE
MANAGEMENT AUTO;
spool off
exit;
```

4.3 Creating Database Objects and Data using TPCC and STB

In the current tests, database objects—tables, indexes, procedures, etc.—and data are primarily created by a couple of database tools: Benchmark Factory for Oracle-TPCC® from Quest Software [3] and STB [4]. Benchmark Factory for Oracle is a load and benchmark test tool for Oracle database testing. It creates multiple database objects and thousands of user transactions. An average run in the test took approximately six hours. STB was developed in-house [4] and was written in Oracle PL/SQL. It creates multiple user procedures and tables for data of various sizes and

patterns. STB was designed to generate known data, which can be used later for validating the cloned database. Data generated by TPCC and STB are stored in the "USER" tablespace.

Figure 2 depicts a GUI panel from the Benchmark Factory-TPCC run. It shows the types of database objects created and operations performed.

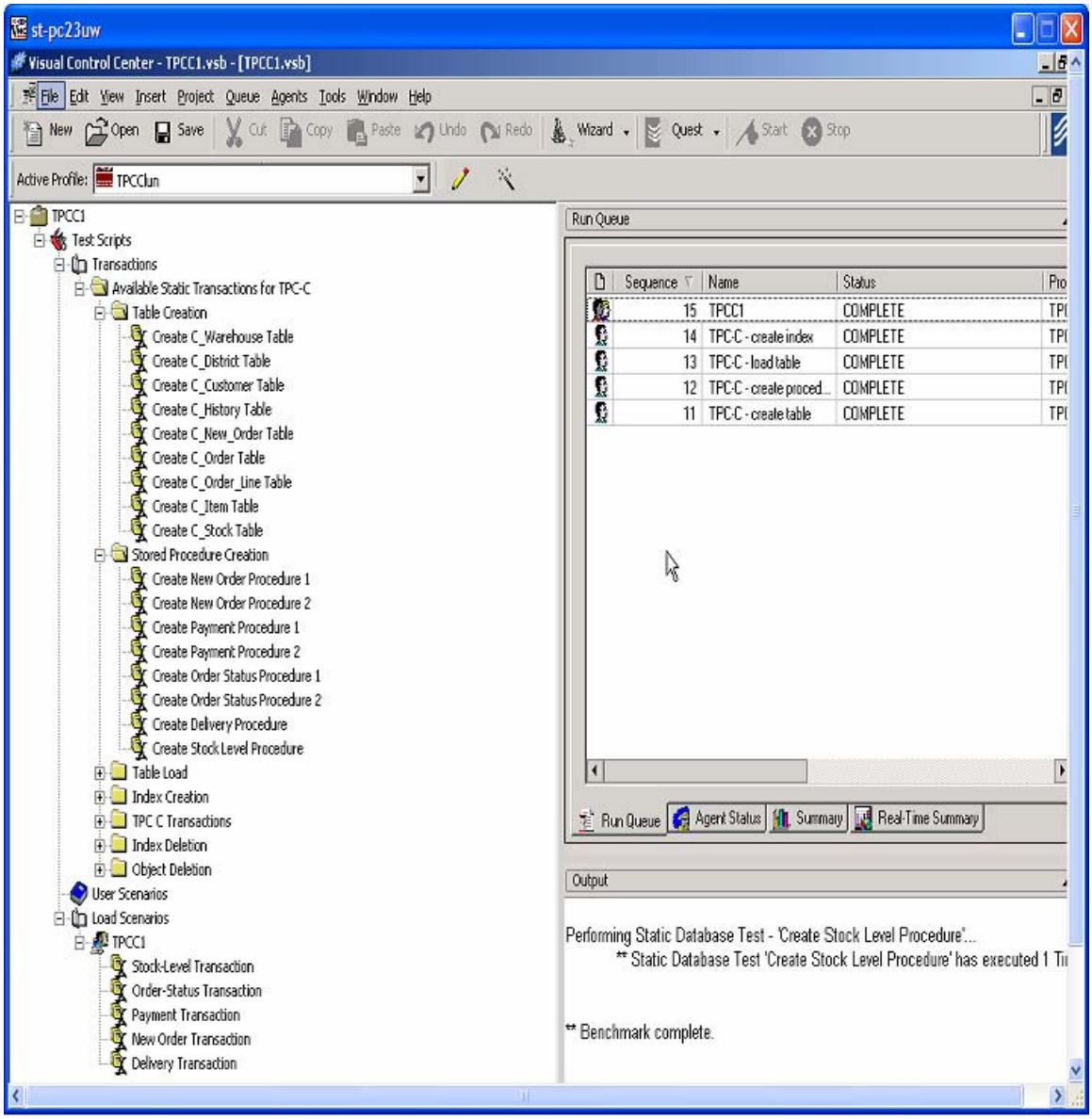


Figure 2) TPCC Window panel.

The STB tool comprises lengthy PL/SQL scripts and will not be shown in this paper.

5.0 Cloning a Database in a NAS Environment

Clone means an exact copy of the original. Cloning a database is the routine DBA task of updating test and development environments from production, and it usually involves backup and recovery processes. The clone can run as an independent instance under a new environment. It should be noted that the clone can only run under the same operating system as the production. One cannot expect, for example, the production database to run on Solaris and the clone on AIX. Moreover, the cloning process should be carried out only when traffic to the production filers is low. DBA should schedule cloning and backup prudently.

This section focuses on a non-RMAN clone under a NAS environment. The next section will discuss RMAN clone in a SAN environment.

There are two types of database backups: hot and cold. In a hot backup, the database is open and extensive transaction activities can occur. In this case, the datafiles tend to be inconsistent with respect to the checkpoint System Change Number (SCN). Archived redo logfiles are mandatory for cloning the database. In a cold backup, the database is shut down. There are no user transactions and background process that could change the system SCN. Therefore, the datafiles are checkpointed to the same SCN, and the state of the database is consistent. Most DBAs use cold backup for cloning. However, there are times that the database cannot be shut down and hot backup must be employed.

The NetApp flexible-volume cloning feature provides an extremely convenient and efficient way for database backup. A single filer command can clone a flexible volume of 200GB in about two to three minutes.

5.1 Full Database Clone with a Hot Backup

Hot backups are performed when the database is open. The database must also be operated in archive log mode. Using the production “DB1” created previously as the source database, our task is to create a clone of “DB1” named “CLONEDB1”. The clone can reside on the same machine as the source, or it can reside on a different machine.

In the cloning process, only the production datafiles and archived redo logfiles are duplicated using the NetApp filer flexible-volume clone feature. It is not necessary to duplicate the online redo logs and control files. They are recreated during the cloning process.

The following steps describe the cloning process.

Step (a):

Correctly set the environment variables `ORACLE_SID`, `ORACLE_HOME`, and `ORACLE_BASE` to reflect the clone's instance and storage configurations. If the clone and the source exist on the same machine, `$ORACLE_HOME` should be similar for both.

```
oracle> setenv ORACLE_SID CLONEDB1
oracle> setenv ORACLE_BASE /export/home/oracle9/clonebase
oracle> setenv ORACLE_HOME /export/home/oracle9/OraHome1
```

Step (b):

Create the following directories on `$ORACLE_BASE` to store the clone's system files, initialization files, alert log files, trace files and control files, etc.

```
oracle> mkdir $ORACLE_BASE/db1clone/admin
oracle> mkdir $ORACLE_BASE/db1clone/dbs
oracle> mkdir $ORACLE_BASE/db1clone/oradata
oracle> mkdir $ORACLE_BASE/db1clone/admin/bdump
oracle> mkdir $ORACLE_BASE/db1clone/admin/cdump
oracle> mkdir $ORACLE_BASE/db1clone/admin/pfile
oracle> mkdir $ORACLE_BASE/db1clone/admin/udump
```

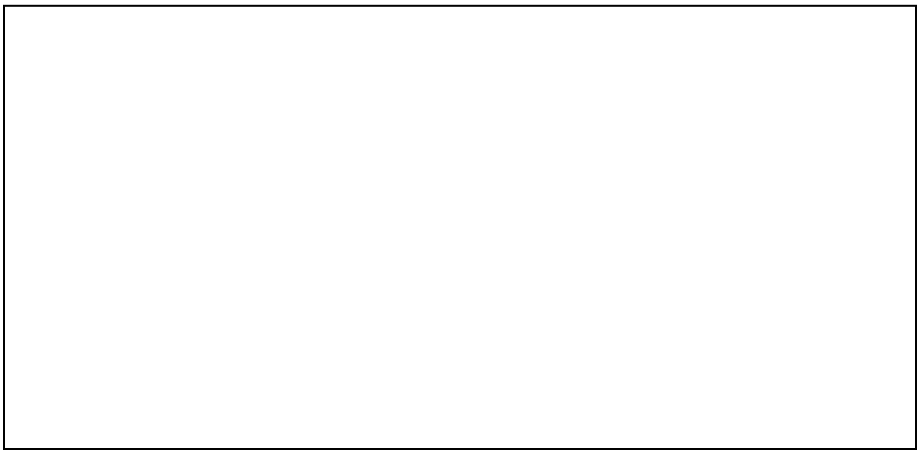

Step (c)

Create an initialization file for the clone "DB1CLONE" by simply copying the existing one from production DB1. Then modify the file with the following parameters to fit the clone environment: `db_domain, db_name, background_dump_dest, core_dump_dest, user_dump_dest, control_files, instance_name, service_names, and log_archive_dest`, etc. Name the resulting initialization file `initClonedb1.ora` and put it on `$ORACLE_BASE/db1clone/admin/pfile`.

Step (d)

Now we are ready to duplicate the production database "DB1". We need to clone the volumes containing all the DB1 datafiles and archived redo logfiles. However, keep in mind that the database is open during the cloning process and user activities are still occurring. That is, the TPCC and STB tools are still inserting and updating data on "DB1", resulting in changes of SCN on transactional commits.

To duplicate datafiles, we first find out what tablespaces exist on "DB1". Then we apply the Oracle `BEGIN BACUP` command for these tablespaces.



Once the tablespaces are found, execute the following SQL script on "DB1":

```
ALTER TABLESPACE SYSTEM BEGIN BACKUP;
ALTER TABLESPACE UNDOTBS1 BEGIN BACKUP;
ALTER TABLESPACE DRYS BEGIN BACKUP;
ALTER TABLESPACE INDX BEGIN BACKUP;
ALTER TABLESPACE TOOLS BEGIN BACKUP;
ALTER TABLESPACE USERS BEGIN BACKUP;
ALTER TABLESPACE XDB BEGIN BACKUP;
ALTER TABLESPACE USERTS BEGIN BACKUP;
```


These `BEGIN BACKUP` commands will freeze the checkpoint SCN in the datafile headers of the associated tablespaces. That is, even though there are user transactions in progress, the checkpoint SCN will remain constant until the command `END BACKUP` is issued.

Now we are ready to make a clone of DB1 on the filer. Make sure that there is enough disk space on the filer aggregates to store the clone. Check the aggregates' free spaces using the following commands before cloning begins.

```
filer> df -A -g aggr1
Aggregate                total       used      avail capacity
aggr1                    454GB       88GB      365GB      20%
aggr1/.snapshot           23GB        0GB       23GB       0%

filer> df -A -g aggr2
Aggregate                total       used      avail capacity
aggr2                    454GB      198GB      255GB      44%
aggr2/.snapshot           23GB        0GB       23GB       0%

filer> vol clone create userdataClone -b dbuserdata
Creation of clone volume 'userdataClone' has completed.

filer> vol clone create sysdataClone -b sysdata
Creation of clone volume 'sysdataClone' has completed.
```

The filer `vol clone` command basically creates new metadata for the new volume and a reference to the source volume snapshot.

As mentioned previously, the cloning process should be conducted when traffic to the filer is low. In these tests the average filer CPU usage was around 50 percent, and the cloning of 300GB of data took just a few minutes.

After the volume cloning process is completed, we end the tablespace backups with the following SQL script.

```
ALTER TABLESPACE SYSTEM END BACKUP;
ALTER TABLESPACE UNDOTBS1 END BACKUP;
ALTER TABLESPACE DRYS END BACKUP;
ALTER TABLESPACE INDX END BACKUP;
ALTER TABLESPACE TOOLS END BACKUP;
ALTER TABLESPACE USERS END BACKUP;
ALTER TABLESPACE XDB END BACKUP;
ALTER TABLESPACE USERTS END BACKUP;
```

Next, we duplicate the archived redo logfiles. Archive redo logs are mandatory for recovery of the cloned database "DB1CLONE". To get the most recent "DB1" redo log records, we archive the unarchived online redo logs. Before doing so, we first check the status of the archive process and the archive log mode.

```
SQL > SELECT ARCHIVER FROM V$INSTANCE;

ARCHIVE
-------
STARTED

SQL> SELECT LOG_MODE FROM V$DATABASE;

LOG_MODE
----------
ARCHIVELOG

SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

The `LOG CURRENT` SQL command archives the most recent online redo logfiles. We can now clone the volume containing the archived redo logfiles with the filer command:

```
filer> vol clone create archClone –b archivedlog
Creation of clone volume 'archClone' has completed.
```

In the test, we cloned about 300GB of datafiles and archived redo logfiles within a few minutes by simply using two filer `vol clone` commands. This efficiency and simplicity is a result of using NetApp Snapshot technology, which offers DBA a robust and fast backup mechanism. In addition, the flexible volume feature allows users to create a distinct container (volume) for a distinct dataset. This is extremely useful for partial backup of a database. For instance, we can create database datasets such as the Oracle binary files, datafiles, multiplexed online redo logfiles, or the archived redo logfiles and store them in separate containers. Each dataset has its own container, and the size of the container is based on the dataset space requirement. If you want to back up a particular dataset—for example, the archived redo logfiles—you need only back up the associated container. No other datasets are involved. This method obviously saves a DBA time and energy in doing backups so that she can spend her energy doing other important tasks.

Step (e):

We can now mount the previously cloned volumes "sysdataClone" and "archClone"on the Oracle server, which can be the same machine as the source database "DB1" or a different machine.

```
oracle9# mount -o hard,intr,suid,vers=3,proto=tcp,llock,forcedirectio,
rsize=32768,wsize=32768, filer:/vol/userdataClone
/export/home/db1clone/userdata
oracle9# mount -o hard,intr,suid,vers=3,proto=tcp,llock,forcedirectio,
rsize=32768,wsize=32768,filer:/vol/sysdataClone /export/home/db1clone/sysdata
oracle9# mount -o hard,intr,suid,vers=3,proto=tcp,llock,forcedirectio,
rsize=32768,wsize=32768, filer:/vol/archClone /export/home/db1clone/archlog
oracle9# mount -o hard,intr,suid,vers=3,proto=tcp,llock,forcedirectio
rsize=32768,wsize=32768, filer:/vol/log1Clone /export/home/db1clone/log1
oracle9# mount -o hard,intr,suid,vers=3,proto=tcp,llock,forcedirectio
rsize=32768,wsize=32768, filer:/vol/log2Clone /export/home/db1clone/log2
```


Note that we did not duplicate the online redo log files of the production "DB1". Instead, we created new volumes "log1Clone" and "log1Clone" to store the online redo logfiles for the clone. In fact, it is advised that you do not duplicate the online redo logfiles and control files since they may corrupt your clone if not used carefully.

Step (f):

At this point, we are done with the file backups and are ready to recover the clone "DB1CLONE". First, we recreate the clone's control files. On the production server, issue this SQL command:

```
SQL> alter database backup controlfile to trace

Database altered.
```

This command generates a trace file to be written to wherever the production "DB1" initialization parameter `USER_DUMP_DEST` is pointing. In our case, it is in `$ORACLE_HOME/DB1/admin/udump`. However, this trace file is somewhat messy and needs to be reshaped. Here is the reshaped trace file in the form of an SQL script `controlClone.sql`.

-- controlClone.sql --

```
connect / as sysdba
STARTUP NOMOUNT
pfile=/export/home/oracle9/clonebase/db1clone/admin/pfile/initDB1CLONE.ora
CREATE CONTROLFILE SET DATABASE "DB1CLONE" RESETLOGS NOARCHIVELOG
--  SET STANDBY TO MAXIMIZE PERFORMANCE
    MAXLOGFILES 5
    MAXLOGMEMBERS 3
    MAXDATAFILES 100
    MAXINSTANCES 1
    MAXLOGHISTORY 907
LOGFILE
  GROUP 1 (
    '/export/home/db1clone/log1/redo01.log'
  ) SIZE 100M,
  GROUP 2 (
    '/export/home/db1clone/log2/redo02.log'
  ) SIZE 100M
-- STANDBY LOGFILE
DATAFILE
  '/export/home/db1clone/sys/system01.dbf',
  '/export/home/db1clone/sys/undotbs01.dbf',
  '/export/home/db1clone/sys/undotbs02.dbf',
  '/export/home/db1clone/sys/undotabs03.dbf',
  '/export/home/db1clone/sys/indx01.dbf',
  '/export/home/db1clone/sys/tools01.dbf',
  '/export/home/db1clone/users/users01.dbf',
  '/export/home/db1clone/users/users02.dbf',
  '/export/home/db1clone/users/users03.dbf',
  '/export/home/db1clone/users/users04.dbf',
  '/export/home/db1clone/users/users05.dbf',
  '/export/home/db1clone/users/users06.dbf',
  '/export/home/db1clone/users/users07.dbf',
  '/export/home/db1clone/users/users08.dbf',
  '/export/home/db1clone/users/users09.dbf',
  '/export/home/db1clone/users/users10.dbf'
CHARACTER SET WE8ISO8859P1;
```

Execute `controlClone.sql` in the clone's instance. This will create new control files for the clone. The files are stored on the directory specified by the clone's initialization parameter `control_files`.

```
SQL> @controlClone.sql
Connected to an idle instance.

ORACLE instance started.

Total System Global Area  135352820 bytes
Fixed Size                   455156 bytes
Variable Size             109051904 bytes
Database Buffers           25165824 bytes
Redo Buffers                 679936 bytes

Control file created.
```

Step (g):

In Step (e), we archived the most recent online redo logfiles of the production database "DB1". However, traffic to the database is still continuing. Users' records keep filling up the active online redo logfiles. Therefore, it is a good idea to check which online redo logfiles have not been archived. Use the following SQL command on the DB1 instance to achieve that:


```
SQL> select a.group# "GROUP", a.member, b.status
"STATUS", b.archived from v$logfile a, v$log b where
a.group#=b.group# and b.archived='NO';
SQL> /

     GROUP
----------
MEMBER
------------------------------------
STATUS           ARC
---------------- ---
         3
/export/home/STB1/log_1/redo03a.log
CURRENT          NO

         3
/export/home/STB1/log_2/redo03b.log
CURRENT          NO
```

The files `redo03a.log` and `redo03b.log` have not been archived. Copy them to the cloned database "DB1CLONE" archive logs directory for later use. In addition, if there is any new archived logfile generated after the snapshot was created as a result of the volume clone (Step d), copy it to the archived logs directory as well.

Step (h)

Now recover the clone "DB1CLONE". In the clone instance, issue this command:

```
SQL> recover database using backup controlfile
```

This command will ask you to enter the archived log file names one by one for recovery. You can select the `AUTO` option. When the archived logs are done, choose the `filename` option and enter the previously copied files `redo03a.log` and `redo3b.log`. The following message indicates the step:

```
Specify log: {<RET>=suggested | filename | AUTO | CANCEL}
/export/home/STB1/clone/archlog/redo03b.log
Log applied.
Media recovery complete.
```

Now that the recovery process for the clone "DB1CLONE" is complete, open it with the `resetlogs` option:

```
sql> alter database open resetlogs;

Database altered.
```


The clone "DB1CLONE" is now ready for use. It is recommended that you back up the clone immediately since the archived redo logfiles are now reset. They cannot be used again to recreate the clone.

The above database cloning procedures are shown step by step for clarification. In general, all the steps can be automated in a single script.

5.2 Partial Database Clone with a Hot Backup

The previous section describes a way to clone an entire database. However, DBA may sometimes want to clone just a single user tablespace from production for maintenance. This section describes how the task is performed using the NetApp filer flexible-volume clone feature.

As an example, our task is to duplicate the production database "DB1" tablespace "USERS" stored on the filer volume "dbuserdata". Here are the steps:

(a) Issue this SQL command in the DB1 instance:

```
SQL> alter tablespace users begin backup;
Tablespace altered.
```

(b) Make a clone of the production volume “dbuserdata”, which is 40GB in size, and name the cloned volume “userClone”. The cloning process usually takes about two to four minutes depending on filer traffic. This is the only step you need to do on the filer:

```
filer> vol clone create userClone –b dbuserdata

Creation of clone volume 'archClone' has completed.
```

(c) End the backup process:

```
SQL> alter tablespace users end backup;
Tablespace altered.
```

(d) Now the cloned volume can be mounted onto the Oracle server to replace the original. After mounting, perform a media recovery on the cloned tablespace:

```
SQL> recover tablespace users;
SQL> alter tablespace users online;

Tablespace altered.
```


5.3 Full Database Clone with a Cold Backup

The previous sections described cloning a production database using a hot backup, in which the database is open. In a cold backup, the database is shut down and no activity is in progress, so the database is in a consistent state.

From the filer point of view, there is no difference between a hot backup and a cold backup. The filer clones only the flexible volumes that compose the database. Therefore, the steps to create a cloned database discussed in Sec. 5.1 for a hot backup are still valid for a cold backup, except that those SQL commands with `BEGIN BACKUP` and `END BACKUP` are no longer necessary.

In general, cloning database using a cold backup is faster and more consistent than using a hot backup. Therefore, if possible, cloning database using a cold backup is highly recommended.

5.4 Partial Database Clone with a Cold Backup

To duplicate a volume containing multiple tablespaces and datafiles on the filer, use the following filer command:

```
filer> vol clone create userdataClone – b dbuserdata
```

After creating the cloned volume "userdataClone", you can mount it to the Oracle server at the same mount point of the original volume "dbuserdata". Then, start up the database. No difference should be seen between the cloned and the original volumes.

```
Oracle9> mount filer:/vol/userdataClone /export/home/DB1/user_data
Oracle9> sqlplus "sys/passwd as sysdba"
SQL> startup pfile=/export/home/OraHome1/admin/DB1/pfile/initDB1.ora;
```


5.5 Recommendations for Cloning

It is important to note that cloning a database from production on the same filer requires prudent procedures. Carelessness may result in corrupting the production. Therefore, it is recommended that, if possible, the clone be placed on a different host and named with different paths for system files, control files, datafiles, and redo logfiles from production.

6. Cloning Database in a SAN Environment Using RMAN

NetApp technical reports related to LUN cloning and backup/recovery of Oracle9i database using a NetApp filer in SAN environment were published in Refs [6,7]. This paper focuses on database cloning using the NetApp flexible-volume cloning feature and RMAN.

6.1 Creating Aggregates, Flexible Volumes, and LUNs

First create aggregates and flexible volumes on the filer. The steps are similar to those described in Section 4.1.

```
filer> aggr create aggr1 30
filer> aggr create aggr2 13
filer> vol create userdata aggr1 500g
filer> vol create archivedlog aggr1 300g
filer> vol create log1 aggr2 50g
filer> vol create log2 aggr2 50g
filer> vol create sysdata aggr2 100g
```

Then create LUNs on the flexible volumes to hold the Oracle user datafiles, system datafiles, archived redo logfiles, and the online redo logfiles. In this test, a single LUN is created to hold multiple files.

```
filer> lun create –s 200g –t solaris /vol/userdata/u_data
filer> lun create –s 200g –t solaris /vol/archivedlog/arch
filer> lun crate –s 60g –t solaris /vol/sysdata/s_data
filer> lun create –s 25g –t solaris /vol/log/log_1
filer> lun create –s 25g –t solaris /vl/log/log_2
```

After the LUNs are created, map them to the Oracle server (initiator group).

```
filer> lun map /vol/userdata/u_data oracle9 0
filer> lun map /vol/sysdata/s_data oracle9 1
filer> lun map /vol/log/log_1 oracle9 2
filer> lun map /vol/log/log_2 oracle9 3
filer> lun map /vol/archivedlog/arch oracle9 4
```

Now that LUNs have been created and mapped to the Oracle server “oracle9”, we can configure the server to utilize these resources. The tasks to configure the server—modifying the `/kernel/drv/sd.conf` for new LUN mappings and creating new partition tables—are well documented in Ref. [5] and will not be discussed here.

On the Oracle server, a new raw disk partition will be created for each LUN mapped. Next, create a file system on each partition by executing the following shell script:

```
#!/usr/bin/sh
echo y | newfs /dev/rdsk/c5t0d0s6
echo y | newfs /dev/rdsk/c5t0d1s6
echo y | newfs /dev/rdsk/c5t0d2s6
echo y | newfs /dev/rdsk/c5t0d3s6
echo y | newfs /dev/rdsk/c5t0d4s6
```

Add the following entries to the Oracle server system file `/etc/vftab` to enable automatic remount at host reboot.

```
/dev/dsk/c5t0d0s6 /dev/rdsk/c5t0d0s6/ /export/home/lun/userdata
ufs 2 yes forcedirectio,onerror=lock
/dev/dsk/c5t0d1s6 /dev/rdsk/c5t0d1s6//export/home/lun/sysdata
ufs 2 yes  forcedirectio,onerror=lock
/dev/dsk/c5t0d2s6 /dev/rdsk/c5t0d2s6/ /export/home/lun/log1 ufs
2  yes forcedirectio,onerror=lock
/dev/dsk/c5t0d3s6 /dev/rdsk/c5t0d3s6/ /export/home/lun/log2 ufs
2 yes forcedirectio,onerror=lock
/dev/dsk/c5t0d4s6 /dev/rdsk/c5t0d4s6/ /export/home/lun/archlog
ufs 2 yes forcedirectio,onerror=lock
```


After the file systems have been created for the LUNs, mount them and change the file ownership.

```
oracle9> mount /export/home/lun/userdata
oracle9> mount /export/home/lun/sysdata
oracle9> mount /export/home/lun/log1
oracle9> mount /export/home/lun/log2
oracle9> mount /export/home/lun/archlog
oracle9> chown -R oracle:dba /export/home/lun/userdata
oracle9> chown -R oracle:dba /export/home/lun/sysdata
oracle9> chown -R oracle:dba /export/home/lun/log1
oracle9> chown -R oracle:dba /export/home/lun/log2
oracle9> chown -R oracle:dba /export/home/lun/archlog
```

6.2 Creating Production Database on Filer LUNs

The procedures to create a production database “DB1” described in Section 4.2 can be applied here. The mount points created for the filer LUNs discussed in section 6.1

are used to hold the database user datafiles, system datafiles, online redo logfiles, and archived redo logfiles.

After the production database has been created, we insert data in it by generating transactions using TPCC and STB tools, similar to those described in section 4.3.

6.3 Procedures for Cloning a Database in an FCP Environment

We are ready to clone the production database created previously. Here are the steps.

Step (a):

On the filer, clone the flexible volumes that comprise the production datafiles and redo archived logfiles. Make sure that appropriate procedures have been performed for a hot backup. Refer to Section 5.1 for details on cloning from hot backups. For cloning from cold backups, simply shut down the database instance and then, as good practice, perform the UNIX command `sync` to flush any file system buffering:

```
filer> vol clone create userdataCLONE -b userdata
filer> vol clone create sysdataCLONE -b sysdata
filer> vol clone create archlogCLONE -b archlog
```


Step (b):

The filer `vol clone` operations in the previous step not only create cloned volumes but also create cloned LUNs that the volume contains, as indicated below:

```
filer> lun show
/vol/userdata/u_data    100g (107374182400)  (r/w, online, mapped)
/vol/sysdata/s_data      60g (64424509440)   (r/w, online, mapped)
/vol/archlog/arch       200g (214748364800)  (r/w, online, mapped)
/vol/log1/log1           25g (26843545600)   (r/w, online, mapped)
/vol/log2/log2           25g (26843545600)   (r/w, online, mapped)
/vol/sysdataCLONE/s_data     60g (64424509440)   (r/w, offline, mapped)
/vol/userdataCLONE/u_data    100g (107374182400)  (r/w, offline, mapped)
/vol/archlogCLONE/arch     200g (214748364800)  (r/w, offline, mapped)
/vol/log1CLONE/log1          25g (26843545600)   (r/w, offline, mapped)
/vol/log2CLONE/log2          25g (26843545600)   (r/w, offline, mapped)
```

Notice that the cloned LUNs are created offline and mapped to the same igroup with identical raw device IDs as the original LUNs. We need to reassign a new device ID

to each cloned LUN and put it online. In the following example, we unmap each cloned LUN from the igroup “oracle9” and then map it again with a new device ID.

```
filer> lun unmap /vol/userdataCLONE/u_data oracle9
filer> lun map /vol/userdataCLONE/u_data oracle9 5
filer> lun online /vol/userdataCLONE/u_data
filer> lun unmap  /vol/sysdataCLONE/s_data oracle9
filer> lun map /vol/sysdataCLONE/s_data oracle9 6
filer> lun online /vol/sysdataCLONE/s_data
filer> lun unmap /vol/log1CLONE/log1 oracle9
filer> lun map /vol/log1CLONE/log1 oracle9 7
filer> lun online /vol/log1CLONE/log1
filer> lun unmap /vol/log2CLONE/log2 oracle9
filer> lun map /vol/log2CLONE/log2 oracle9 8
filer> lun online /vol/log2CLONE/log2
filer> lun unmap /vol/archlogCLONE/arch oracle9
filer> lun map /vol/archlogCLONE/arch oracle9 9
filer> lun online /vol/archlogCLONE/arch
```

Step (c):

On the Oracle server “oracle9”, we should be able to see that a new disk device is created for each cloned LUN:


```
oracle9.lab.netapp.com:r sanlun lun show
     filer:        lun-pathname         device filename         adapter         lun size
lun state
filer:/vol/userdata/u_data    /dev/rdsk/c5t0d0s2      lpfc0    100.0g (107374181888)
GOOD
filer:/vol/sysdata/s_data     /dev/rdsk/c5t0d1s2      lpfc0     60.0g (64424508928)
GOOD
filer: /vol/log1/log1         /dev/rdsk/c5t0d2s2      lpfc0     25.0g (26843545088)
GOOD
filer: /vol/log2/log2         /dev/rdsk/c5t0d3s2      lpfc0     25.0g (26843545088)
GOOD
filer:/vol/archlog/arch       /dev/rdsk/c5t0d4s2      lpfc0    200.0g (214748364288)
GOOD
filer:/vol/userdataCLONE/u_data   /dev/rdsk/c5t0d5s2      lpfc0    100.0g (107374181888)
GOOD
filer:/vol/sysdataCLONE/s_data    /dev/rdsk/c5t0d6s2      lpfc0     60.0g (64424508928)
GOOD
filer:/vol/log1CLONE/log1     /dev/rdsk/c5t0d7s2      lpfc0     25.0g (26843545088)
GOOD
filer:/vol/log2CLONE/log2     /dev/rdsk/c5t0d8s2      lpfc0     25.0g (26843545088)
GOOD
filer:/vol/archlogCLONE/arch     /dev/rdsk/c5t0d9s2      lpfc0    200.0g (214748364288)
GOOD
```

Now we mount the cloned LUNs, which hold the cloned database, on the server:

```
oracle9# mount /dev/dsk/c5t0d5s6/ /export/home/lun/STB1Clone/userdata
oracle9# mount /dev/dsk/c5t0d6s6/ /export/home/lun/STB1Clone/sysdata/
oracle9# mount /dev/dsk/c5t0d7s6/ /export/home/lun/STB1Clone/log1
oracle9# mount /dev/dsk/c5t0d8s6/ /export/home/lun/STB1Clone/log2
oracle9# mount /dev/dsk/c5t0d9s6/ /export/home/lun/STB1Clone/archlog/
```

Then on the server, create the appropriate directories for Oracle system files, the initialization file, and control files for the clone, similar to those described in Section 5, Steps (b), (c) and (f). If you are running the clone and production on the same machine, make sure that the two database names, redo logfile, and control file locations are different.

Step (e):

After the clone database's control files and initialization file have been created, we can use RMAN to recover the database from the clone archived redo logfiles:

```
oracle9> rman target sys/passwd@cloneDB

Recovery Manager: Release 9.2.0.1.0 - Production

Copyright (c) 1995, 2002, Oracle Corporation.  All rights reserved.

connected to target database (not started)

RMAN> startup mount

Oracle instance started
database mounted

Total System Global Area     135352820 bytes

Fixed Size                     455156 bytes
Variable Size                109051904 bytes
Database Buffers              25165824 bytes
Redo Buffers                    679936 bytes

RMAN> recover database until sequence 6 thread 1;

Starting recover at 04-NOV-04
using target database controlfile instead of recovery catalog
allocated channel: ORA_DISK_1
channel ORA_DISK_1: sid=11 devtype=DISK

starting media recovery
media recovery complete

Finished recover at 04-NOV-04

RMAN> alter database open resetlogs;

database opened
```

At this stage, the multiplexed online redo logfiles have been created at the locations specified in the control files. Note that in the RMAN command `recover database until`, `sequence 6` is used in the present test. You should check this sequence number in your archived redo logfile directory before using it.

Step (f):

Now the clone database has been created. It is important to note that the above duplication process will result in the production and clone having an identical DBID. Thus, changing the clone's DBID is recommended before using it, especially if it is running on the same host as the production and the RMAN recovery catalog is used for backup and recovery. The DBNEWID utility provided by Oracle is meant for this purpose and gives the clone a new DBID.

Note that RMAN also provides the command `duplicate` to clone a database. Basically the `duplicate` command performs the following tasks: (1) determining the nature and locations of database backups; (2) allocating an auxiliary channel for the auxiliary instance; (3) restoring the datafiles and archived redo logs to the auxiliary instance; (4) building a new auxiliary control file; (5) performing an incomplete recovery for the clone; (6) resetting the DBID for the clone; and (7) opening the clone with `resetlogs`. However, by using the NetApp flexible-volume cloning feature as described above you can avoid the lengthy backup and restore processes that the RMAN command `duplicate` requires. This significantly saves a DBA's time and energy.

7. Verifications of the Cloned Database

This section describes the methods used to validate the cloned database. The methods include (a) verifying the data generated by TPCC and STB, (b) using the Oracle tool `db_verify` to check the structural integrity of the database, and (c) using the NetApp filer commands `WAFL_check` and `aggr wafliron` to check volume integrity.

7.1 STB and TPCC Verifications

The user data in the database are essentially generated by the STB and TPCC tools, as described in Section 4.3. STB generates known data in the production database "DB1". These data are then duplicated in the cloned database. To validate the cloned data, we retrieve the known data from the clone and compare them with the original to make sure they match. Similarly, TPCC creates data for various tables, indexes, and procedures, etc. The same TPCC transactions are run on the clone to make sure that the data objects are intact.

7.2 Verifications with NetApp `WAFL_check`, `aggr_wafliron`, and Oracle `db_verify`

To check the data integrity of the NetApp filer aggregates and volumes that comprise the cloned database, NetApp filer commands "`WAFL_check`" and "`aggr_wafliron`" are used. The former command is run at filer reboot, while the latter is run when the aggregate is online.

```
filer> aggr wafliron start aggr1
```

Oracle also provides the `db_verify` utility to check the structural integrity of the database files for corruption. The following Perl script performs the `db_verify` operations on all data files with names ended with `.dbf` in a directory. It also prints the `db-verify` results to a file.

```
#!/usr/bin/perl
#####################################################################
#File:dbverify.pl
#This script is used to verify Oracle database datafiles
#Usage: perl db_verify.pl --dir=directory_of_the_datafiles
#####################################################################
use strict;
use Getopt::Long;
use IO::Handle;
my $dir;
my $status;
GetOptions ('dir=s' => \$dir,);
chomp ($dir);
#Create an output file
open (IFP, ">db_verify.txt") or die "Can't open file:$!\n";
IFP->autoflush(1);
my @dbf_list = `ls -alt $dir`;
# Do db_verify on all the DB data files and print the results
for (my $i =0; $i < @dbf_list; $i++) {
   if ($dbf_list[$i] =~ /\s+(\S+\.dbf)/) {
       print IFP "\n\nVerify $1\n";
       $status = `dbv file=$dir/$1 blocksize=8192 2>&1`;
       print IFP "$status";
   }
}
close (IFP);
```

.

The following shows the `db-verify` output of a single `.dbf` file.

```
Verify users01.dbf

DBVERIFY: Release 9.2.0.1.0 - Production on Fri Aug 27 10:56:53 2004

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.

DBVERIFY - Verification starting : FILE =
/export/home/STB1/clone/users/users01.dbf


DBVERIFY - Verification complete

Total Pages Examined         : 25600
Total Pages Processed (Data) : 314
Total Pages Failing   (Data) : 0
Total Pages Processed (Index): 126
Total Pages Failing   (Index): 0
Total Pages Processed (Other): 20
Total Pages Processed (Seg)  : 0
Total Pages Failing   (Seg)  : 0
Total Pages Empty            : 25140
Total Pages Marked Corrupt   : 0
```

8. Conclusion

The NetApp flexible-volume cloning feature provides DBA a fast and robust way to clone a production database. The clone feature and its associated Snapshot technology offers time and space efficiency in database backup and recovery operations. The flexible volume feature allows users to create a distinct container for a distinct dataset. This greatly simplifies backup and cloning of a partial database. The procedures described in this paper provide a way to meet the cloning objective.

9. Caveats

The tests described in this paper were conducted in the System Test Lab. Cloning of a Oracle9i database was done on Sun Solaris 8 and a filer F880 platform loaded with the Anchorsteam release. NetApp has not tested this configuration with all the combinations of hardware and software options available. There may be significant differences in your configuration that will change the procedures necessary to accomplish the objectives outlined in this paper. If you find that any of these procedures do not work in your environment, please contact the author immediately.

10. Technical Reference:

[1] Jerry Liu, Jeff Browning and Tim Moore, *Oracle8 for UNIX: Providing Disaster Recovery with NetApp SnapMirror Technology,* NetApp TR 3057, 8/2000.
[2] Customer Simulation Lab managed by System Test Team. Contact: Francis Hong.
[3] Benchmark Factory for Oracle, Quest Corp.: www.quest.com/bmfo.
[4] PL/SQL codes largely developed by Shou-Wen Chen of the System Test Team.
[5] Brian Casper, *Oracle9i for UNIX: Integrating with a NetApp Filer in a SAN Environment,* NetApp TR 3207, 12/2002.
[6] Richard Jooss and Brian Casper, *Oracle9i for UNIX: Backup and Recovery Using a NetApp Filer in a SAN Environment,* NetApp TR 3210, 04/2004.
[7] Toby Creek, *Application for Writeable LUNs and LUN Cloning in Oracle Environments,* NetApp TR 3266, 6/2003.
[8] Bruce Clarke and Sankar Bose, *Using Oracle with Multiprotocol Filer,* NetApp TR 3203.
[9] Oracle Simulator, User's Guide, Release 6.0, 1/2004
[10] Jeff Browing, *Oracle8 for UNIX: Integration with a NetApp Filer,* NetApp TR 3047.

Revision History

Date	Name	Description

1/2005	Sunny Ng	Creation
5/25/2005	Mel Shum	Updated Aggregates to conform to best practices