

SecureFiles Performance

An Oracle White Paper
November 2007

1. INTRODUCTION	3
2. MACHINE CONFIGURATION	4
3. SECUREFILES BASIC WORKLOADS.....	5
WORKLOAD DETAILS	5
<i>Table</i>	5
<i>C/OCI</i>	5
<i>Code</i>	6
Write.....	6
Read.....	7
<i>Performance</i>	8
Java (JDBC) Thin/OCI	9
<i>Code</i>	9
Write.....	9
Read.....	10
<i>Performance</i>	10
4. SECUREFILE VS. NETWORK FILE SYSTEM.....	11
WORKLOAD DETAILS	11
<i>Test Settings</i>	11
<i>SecureFiles VS Filesystem (NFSV3 over Ext3 FS)</i>	12
<i>Read Performance</i>	13
<i>Write Performance</i>	13
<i>Hardware & Software Configuration</i>	13
5. SCALABILITY EXPERIMENTS.....	14
Oracle Multimedia (InterMedia).....	14
6. SECUREFILES BEST PRACTICES.....	15
CHUNK	15
WRITE GATHER CACHE	15
LOGGING	15
DEVICE CONTENTION	16
TABLESPACES AND DEVICES	16
DATABASE PARAMETERS	16
<i>SDU/TDU Size</i>	16
<i>Log_Buffer</i>	16
<i>Recv_Buf_size/Send_Buf_size</i>	16
KERNEL PARAMETERS	17
MONITORING	17
7. CONCLUSION	18
8. APPENDIX.....	19
MAJOR FILESYSTEMS	19
<i>NFS + EXT3</i>	19
<i>Windows NTFS</i>	20
<i>Solaris 10 ZFS (Zettabyte File System)</i>	20
<i>Solaris 7 UFS does metadata journaling</i>	20
<i>IBM AIX JFS2</i>	20
<i>Network Appliance Filer</i>	21

1. INTRODUCTION

SecureFiles is a new feature in Oracle Database 11g that is specifically engineered to deliver high performance and scalability for storing unstructured or file data inside the Oracle database. SecureFiles presents the best of both the file system and the database worlds for unstructured content. Data stored inside SecureFiles can be queried or written at performance levels comparable to that of traditional file systems while retaining the advantages of the Oracle database. SecureFiles supports advanced file system features such as Deduplication, Compression and Encryption. SecureFiles is designed as a superset of the ANSI standard LOBs and offers easy migration from old LOBs (now called “BasicFiles” in Oracle RDBMS 11gR1). For details on usage please refer to the Oracle Database 11g SecureFiles White Paper at <http://www.oracle.com/technology/products/database/securefiles/pdf/securefiles.pdf>

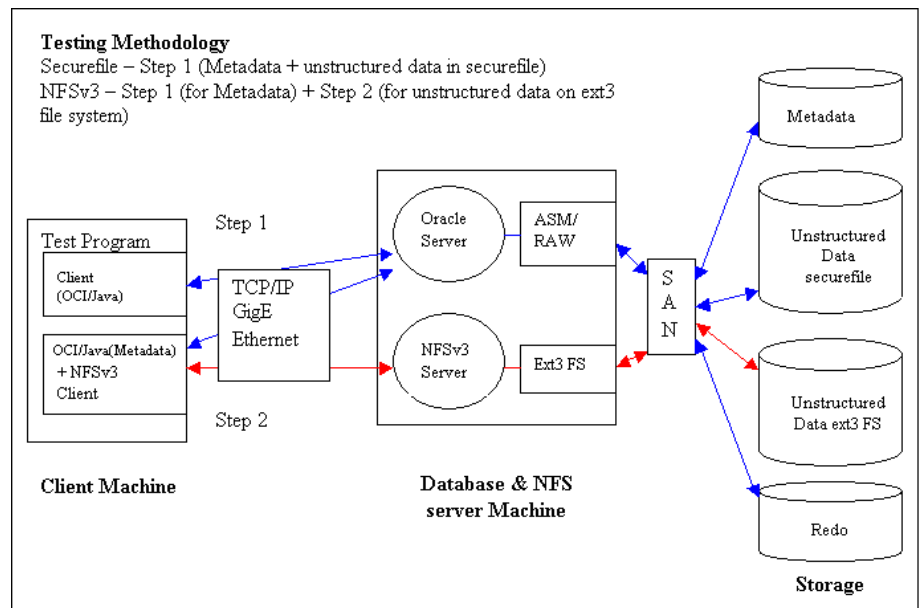
This paper presents the benchmarks and result data to demonstrate the impact of the SecureFile performance and scalability in 11gR1. We compare SecureFiles to older LOBs (Oracle Database 10g or prior) or BasicFiles as a baseline for performance using both C and JDBC. We then compare SecureFiles with the Network File System (for typical applications that currently keep metadata in the database and unstructured data in the file system that is accessed over NFSv3) and provide comparison of SecureFiles against BasicFiles for Oracle Multimedia applications as an example of compatibility. Finally, we compare SecureFiles to NFS in a WAN environment to show SecureFiles performance in a high latency environment.

The scripts used in these tests are available for download from OTN (<http://www.oracle.com/technology/products/database/securefiles/index.html>)

2. MACHINE CONFIGURATION

The test environment consisted of two computers connected by a gigabit Ethernet connection, unless otherwise noted. When we tested “server to server” performance, we used the server machine as both the client and the server, and the workload was driven through the loopback interface. The server was running Oracle RDBMS 11.1.0.6

The environment was configured as seen in the diagram below:



Server configuration

Server: Dell 2850

CPUs: 2 Intel Xeon 3.2 GHz, hyper-threaded processors with 2MB cache each

Memory: 6GB total

Disk: EMC CX700

OS: Red Hat Enterprise Linux 4.0

SAN Host adaptor: 2Gbit Fiber Channel

Network: 1 Gbit Ethernet card

Client configuration

Client: Dell 2650

CPUs: 2 Intel Xeon 2.8GHz, hyper-threaded processors with .5 Mb cache each

Memory: 6GB total

OS: Red Hat Enterprise Linux 4.0

Network: 1 Gbit Ethernet card

Storage Devices

Drives were allocated to two identically configured 2TB Raid 5 arrays. One of the units was allocated to Oracle and was managed using ASM. The other had been configured with a file system that is made available via NFS to the client machine.

Both client and server were running Red Hat Enterprise Linux Release 4. The network was configured with an MTU of 1500. The TCP stack was configured with rmem and wmem of 512k.

3. SECUREFILES BASIC WORKLOADS

Workload Details

Table

number(10) primary key

blob

The create table SQL follows:

```
create table foo (pkey number(10) not null,
                 DOCUMENT blob)
lob(DOCUMENT) store as SecureFile
FOO_DOCUMENT_LOBSEG (
  NOCACHE LOGGING RETENTION AUTO);
```

C/OCI

The test application is a simple C program using the OCI interfaces to write rows to the above table.

The application was simple. It wrote a specified number of LOBs using the LOB Locator based API available in C/OCI. The LOBs were written as a series of buffers. The transaction was committed after each LOB was complete. We tested with the default settings for SecureFiles (NOCACHE LOGGING and ENABLE STORAGE IN ROW).

For multi-stream performance, we ran the multiple copies of the test on the same machine and simply inserted/read different keys for each test client.

We measured the throughput as the bytes written to the disk divided by the total time to insert, write and commit all LOBs.

We tested the following basic workloads to show representative numbers for SecureFiles Performance:

- Write/Read 10 32MB SecureFiles using a 1MB chunksize aligned buffer committing after each SecureFile is written.
- Write/Read 10 32MB SecureFiles using a 1MB chunksize aligned buffer with space reclamation committing after each SecureFile is written.
- Write/Read 10 32MB SecureFiles using a 1MB chunksize unaligned buffer (i.e. the buffer was exactly 1MB vs. 1047800 bytes) committing after each SecureFile is written.
- Four clients each performing Write/Read 10 32MB SecureFiles using a 1MB chunksize unaligned buffer committing after each SecureFile is written.

For the reclamation testing, we first ran the exact workload once, and then deleted all rows from the table and committed. We then ran the write workload again. This insertion and deletion causes the SecureFile data space to reclaim the deleted SecureFile data as there wasn't sufficient space in the segment to allow two complete sets of writes of the dataset.

Code

Write

The C/OCI write test inserted the data using the returning clause of the SQL INSERT command and the OCILobWrite2 function with a callback:

```
static text *stmt_insertstr = (text *)"INSERT INTO
FOO VALUES (:PKEY, EMPTY_BLOB()) RETURNING document
into :LOC";

typedef struct cbctx {
    ub1 *buffer_cbctx;
    ub4  bufferlen_cbctx;
    ub8  position_cbctx;
    ub8  length_cbctx;
} cbctx;

sb4 callback(void *ctxp, void *bufp, oraub8 *lenp,
             ub1 *piece, void **changed_bufpp,
             oraub8 *changed_lenp)
{
    ub4      nbytes;
```

```

cbctx *cbctxp = (cbctx *)ctxp;
oraub8 amt = (*lenp > cbctxp->bufferlen_cbctx) ?
              cbctxp->bufferlen_cbctx : *lenp;
oraub8 rem = cbctxp->length_cbctx -
              (cbctxp->position_cbctx - 1);

*piece = OCI_NEXT_PIECE;

if (rem > amt)
{
    nbytes = amt; /* Still have more pieces to go */
}
else
{
    nbytes = rem;
    /* This is going to be the final piece */
    *piece = OCI_LAST_PIECE;
}

memcpy(bufp, cbctxp->buffer_cbctx, nbytes);

*lenp = nbytes;

cbctxp->position_cbctx += nbytes;

*changed_bufpp = NULL;
return OCI_CONTINUE;
}

/* Insert into foo values (pkey, empty_blob() ); */
CheckErr(errhp, OCISstmtExecute(svchp, stmtinserthp,
                               errhp, (ub4) 1, (ub4) 0,
                               (CONST OCISnapshot *) 0,
                               (OCISnapshot *) 0, OCI_DEFAULT));

/* Got the Locator, now start writing data to it */
ub8 ls = 0;
CheckErr(errhp, OCILobWrite2(svchp, errhp,
                             lobloc, &ls, NULL, (oraub8)1,
                             (void *)LobWriteBuf,
                             (oraub8)SendSize,
                             OCI_FIRST_PIECE,
                             (void *)&ctx, callback, (ub2) 0,
                             (ub1) SQLCS_IMPLICIT));

```

♣ Oracle recommends using the streaming API as shown above. It is the most efficient way of writing a LOB as it prevents multiple calls to the server.

Read

The C/OCI read test used the OCI LOB Locator API to get the LOB locator and then used the OCILobRead2 function with a callback to read the data from the LOB:

```

static text *stmt_sellocstr = (text *)"SELECT
DOCUMENT FROM FOO WHERE PKEY = :MYPKEY";
typedef struct cbctx
{
    ub8 length_cbctx;
    ub8 read_cbctx;
    ub4 count_cbctx;
} cbctx;
sb4 callback(dvoid *ctxp, CONST dvoid *bufp, oraub8
len, ub1 piece,
             dvoid **changed_bufpp, oraub8
*changed_lenp)
{
    cbctx *cbctxp = (cbctx *)ctxp;

    cbctxp->read_cbctx += len;
    cbctxp->count_cbctx++;

    /* We just dump the buffer on the floor */
    return OCI_CONTINUE;
}

for (i=0; i < NumRows; i++)
{
    ub8 Len = LobSize;
    ctx.length_cbctx = Len;
    ctx.read_cbctx = 0;
    ctx.count_cbctx = 0;

    /* Select statement to get Locator */
    CheckErr(errhp, OCISstmtExecute(svchp,
                                   stmtsellochp, errhp, ub4) 1,
            (ub4) 0,
            (CONST OCISnapshot *) 0,
            (OCISnapshot *) 0,
            OCI_DEFAULT));

    CheckErr(errhp, OCILobRead2(svchp, errhp, lobloc,
                                &Len, NULL, 1,
                                (void *) LobReadBuf,
                                (oraub8)ReadSize,
                                OCI_FIRST_PIECE,
                                (void *)&ctx, callback,
                                (ub2) 0, (ub1) SQLCS_IMPLICIT));
}

```

♣ Oracle recommends using the streaming API as shown above. It is the most efficient way of reading a LOB as it prevents multiple calls to the server.

Performance

The client/server system had a maximum theoretical throughput of 125MB/s caused by the single 1Gb Ethernet between the client machine and the server machine.

Realistically, the maximum throughput of the Ethernet was between 100 to 110MB/s with the many various protocols overheads.

SecureFile	Write MB/s	Read MB/s
Aligned	80	94.28
Aligned/Reclaim	79.8	94.01
Unaligned	80.4	94.02
4 Client Aligned	96.14	109.54

SecureFiles achieved 80MB/s with a single client process and 96MB/s with 4 client processes for write performance, and 94MB/s with a single client and 109MB/s with 4 clients for read performance.

It is important to note that SecureFiles doesn't have any performance degradation due to space reuse. Also there is no degradation because of unaligned writes.

Java (JDBC) Thin/OCI

The Java application was a java equivalent of the C/OCI application.

Code

Write

We tested the following variations for SecureFiles:

- One and four stream write, 512K LOB length to 32MB LOB length varying buffer size from 64k (aligned to chunksize) to 1MB (aligned to chunksize) in powers of 2 (aligned to chunksize).

The Java write application used either oci8 or thin client protocols, and the following code:

```
while (i<loops) {
    cs = (OracleCallableStatement) conn.prepareCall(
        "BEGIN insert into foo values(" + key +
        ", empty_blob()) returning document into ?;" +
        " END;");
    cs.registerOutParameter(1, OracleTypes.BLOB );
    cs.execute();
    stmt = conn.createStatement();
    myblob = cs.getBLOB(1);
    long pos = 1;
    int blen = len;
    int bpos = 0;
    while(pos < lobSize)
    {
        myblob.setBytes(pos, buffer, bpos, blen);
        pos += blen;
        bpos += blen;
        if (bpos+blen > lobSize)
        {
            blen = lobSize-bpos;
        }
    }
}
```

Read

We tested the following variations for SecureFiles:

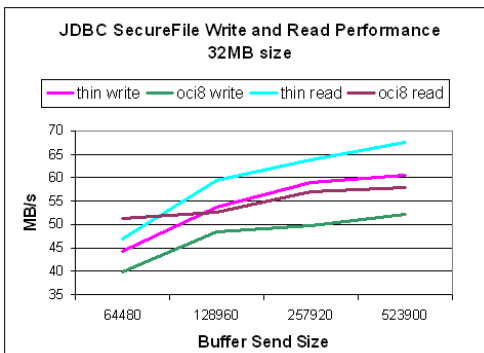
- One and four stream read, 512K LOB length to 32MB LOB length varying buffer size from 64k (aligned to chunksize) to 1MB (aligned to chunksize) in powers of 2 (aligned to chunksize).

The Java read application used either oci8 or thin client protocols, and the following code:

```
while (i<loops) {
    stmt = conn.createStatement();
    ResultSet rset = stmt.executeQuery(
        "select document from foo where pkey = "+
        key + " for update");
    while(rset.next()){
        myblob = ((OracleResultSet) rset).getBLOB(1);
        long pos = 1;
        int blen = len;
        int bpos = 0;
        if(i == 0)
        {
            lobSize=new Long((myblob.length())).intValue();
        }
        while(pos<lobSize)
        {
            buffer = myblob.getBytes(pos, blen);
            pos += blen;
            bpos += blen;
            if (bpos+blen > lobSize)
            {
                blen = lobSize-bpos;
            }
        }
    }
}
```

Performance

In comparing the two protocols, we can see that the “thin” client is faster than the “thick” client. This is due to overhead imposed by the Java/C (JNI) interface.



4. SECUREFILE VS. NETWORK FILE SYSTEM

Workload Details

The test application simulates a real world DICOM (Digital Imaging and Communications in Medicine standard) application consisting of patient metadata and digital diagnostic images (image sizes could range from 10KB to 100MB with an average of around 0.5MB). We compare the performance of SecureFiles to that of the NFSv3 file system. In both cases, metadata is stored in an Oracle database. In the case of file system, the DICOM images are stored on file servers that are accessed from a client machine over NFSv3. In the case of SecureFiles, the metadata as well as the DICOM images are stored inside an Oracle database.

A simple schema is used where we have a primary key and some metadata

File System	SecureFile
number(10) primary key	number(10) primary key
varchar2 (300) Owner	varchar2 (300) Name
varchar2 (100) PathToFile	securefile Document

Test Settings

File System	SecureFile
Client Access OCI + NFSv3 (TCP/IP)	Client Access OCI TCP/IP
8 NFSv3 Servers (nfsd's)	Database session and Oracle Background processes
Linux ext3 File System	ASM - same set of disks (as file system) were used for DATA. Separate set of disks used for Redo Logs.
Write Op - insert metadata in database then write the file to the file system server over NFSv3 using system calls open() and write()	Write Op - OCI Program uses LONGAPI to write both metadata and file to the database in one round trip.

File System	SecureFile
Read Op - Randomly pick up a row ID, get metadata for the file from the database, then read the file from the file system server over NFSv3 using system calls open() and read().	Read Op - Randomly pick up a row ID, use OCI LONGAPI to read both metadata and file in the database in one round trip. For 100MB reads uses callback mechanism and reads in 10MB pieces
sync and async options for NFSv3 and the default data=ordered setting for the linux ext3 file system	NoCache-Logging-DisableStorageInRow (FILESYSTEM_LIKE_LOGGING)
Tests are run for sizes 10KB, 100KB, 1MB, 10MB and 100MB. Uses the same commit rate as SecureFile. The file handle is opened and closed for each file.	Tests are run for the same sizes as file system. For file sizes 100M and 10M, the tests do a commit for every insert. For file sizes 1M, 100K and 10K, the tests do a commit every 100 inserts. The lob handle is opened and closed for each file insert.
For file sizes 100M, 10M and 1M, the total amount of data inserted for each file size is 100G. For file sizes 100K it is 10G and for 10k the total amount of data inserted is 1G. The client side cache is disabled by un-mounting and mounting the file system – note that client side cache is not helpful in real world scenarios where reads are random in nature.	Same as file system.

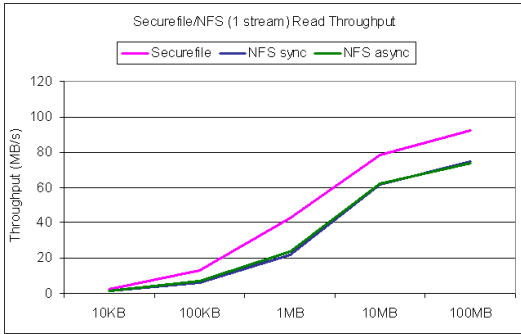
SecureFiles VS Filesystem (NFSV3 over Ext3 FS)

This section compares the performance of SecureFiles with the File System (for the workload discussed above). **Note that these throughput numbers are dependent on the network and storage setup.** Details of the setup can be found in the configuration section of this document.

In Oracle Database 11g, SecureFiles supports a new logging level, FILESYSTEM_LIKE_LOGGING, which is similar to logging available with popular file systems. When SecureFiles logging is set to this level, Oracle writes only the metadata to the redo log. This setting is similar to the metadata journaling of file systems, which reduces mean time to recovery from failures and is sufficient for crash

recovery or instance recovery. SecureFiles also supports database logging in which both the metadata and lob data are written to the redo log. This is especially useful when media recovery or standby database are required. In such cases, archive log should be turned on. In our tests, we set SecureFiles to FILESYSTEM_LIKE_LOGGING mode to keep the logging level and functionality comparable to that of the file system (ext3 data=ordered). Note that the FILESYSTEM_LIKE_LOGGING mode gives true transactional behavior, is fully crash recoverable, gives consistent reads and is multi-versioned unlike NFS+ext3.

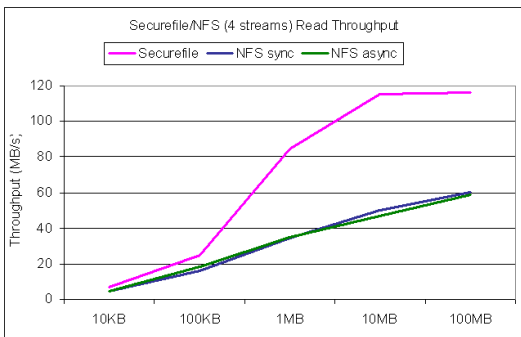
Single and multi-stream tests were done to see the performance and scalability. We have tried to use the best performing options for NFSv3 (async and rwszie of 32KB). The data=writeback options for ext3 was also tried but it was not any different from the NFSv3 async numbers. For explanation and details see the Appendix.



Read Performance

SecureFile outperforms the NFSv3 access for all sizes. Gains for the smaller file sizes are also due to reduced roundtrips where metadata and data is accessed in one roundtrip unlike the NFSv3 case where metadata and file is accessed in separate roundtrips.

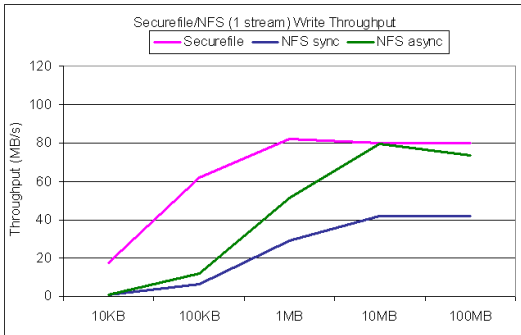
The big difference we see in the read performance for larger file sizes is because of intelligent pre-fetching, larger I/O sizes due to better contiguous space allocations and network optimizations.



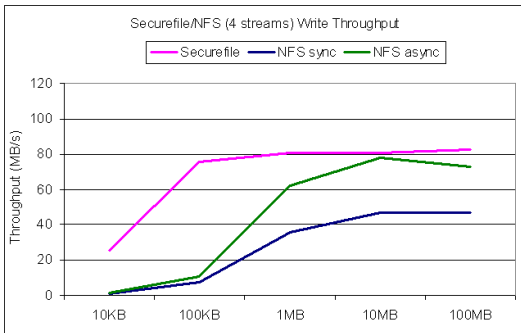
Write Performance

SecureFile outperforms the NFSv3 access for all sizes. Again as in the read case, for small file sizes the NFSv3 case has the overhead of writing metadata and file separately in two roundtrips.

This demonstrates the improvements due to the write gather cache, larger contiguous I/O and space pre-allocation optimizations.



For smaller file sizes the throughput is limited by the roundtrip overheads and the disks are not utilized completely. Increasing the number of concurrent threads would ensure increased throughput. For larger file sizes the throughput is limited by the network and I/O subsystem. In this test the read is limited by the network and writes is limited by I/O. The throughput would scale based on the hardware.



Hardware & Software Configuration

The configuration used for the tests are as follows

- Server (DB & NFSv3) - 4x16G AMD Opteron (tm) 64 Bit Processor 875 CPUs (8 cores) running at 2.2GHz, 1 MB L2, 8 * 73GB disks, 15K RPM, 1

disk used for OS, 7 disks = 3 RAID 0 (Redo Log, SYS, SYSAUX) + 4 RAID 0 128K stripe size (Data SecureFile/File System). Used ASM (1MB stripe size over the controller level hardware stripe depth of 128K) for building the database.

- Client (DB & NFSv3) - 4x8G AMD Opteron (tm) 32 Bit Processor 875 CPUs (8 cores) running at 2.2GHz, 1 MB L2
- Network - Dedicated GigE network. OS network settings (rmem 1MB, wmem 256KB)
- Operating System - Server - 2.6.9-42.0.3.ELsmp x86_64, Client - 2.6.9-22.ELsmp i686 athlon i386
- Database - 11gR1 (32 Bit)
- NFSv3 - Default settings (sync) as well as async, TCP protocol, v3, rsize=32768, wsize=32768 and 8 nfsds
- Ext3 – Default settings (data=ordered)

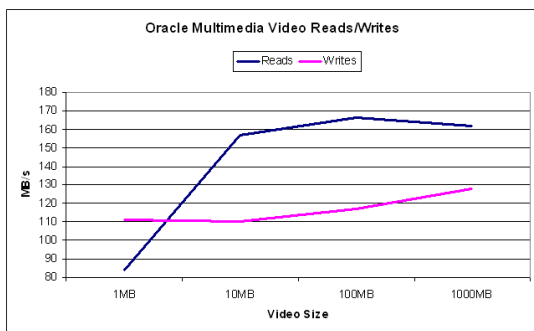
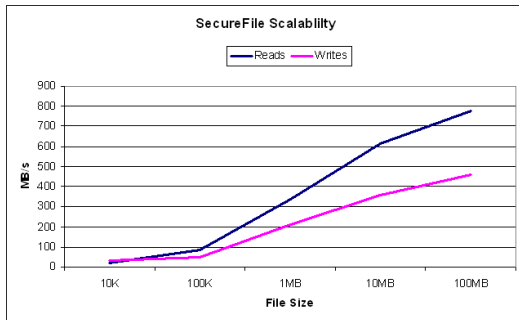
5. SCALABILITY EXPERIMENTS

We performed some tests to demonstrate scalability with SecureFiles. Given the following hardware, SecureFiles was able to provide a sustained throughput of 776 MB/sec for reads and 462MB/sec for writes (38TB/day of data ingestion).

- 4 RAC Nodes – 2x6GB Intel Xeons with Hyperthreading. HBA is 2GBs, 3 EMC CX700 connected through 2 switches, 12 LUNs spanning the 3 arrays
- OCI, 8 concurrent streams total
- SecureFiles settings NOCACHE FILESYSTEM_LIKE_LOGGING

Oracle Multimedia

Experiments were also done with Oracle Multimedia where the throughput of loading and reading videos was measured. This also showed throughput scaling with increasing file size. Throughput is only limited by the hardware constraints. For example, SecureFiles showed a 6.5x performance improvement over BasicFiles for pyramid building and a 36% improvement over BasicFiles for load and scale (down) tests on a simple 1-CPU (P4-HT@3GHz) with a single 80GB, 7200RPM disk drive.



6. SECUREFILES BEST PRACTICES

This section discusses the recommendations for getting better performance from SecureFiles and to avoid errors.

Chunk

SecureFiles, unlike BasicFiles or older LOBs, attempts to write using maximum sized contiguous allocations on disk in order to reduce fragmentation. Because of this, the `CHUNK` parameter is not used for SecureFiles. If set, SecureFiles silently ignores it, but will still return a value aligned to the data block size consistent with the `CHUNK` parameter to continue to allow existing applications to function properly without change. Oracle still recommends using a multiple of the chunk size as a guide for buffer sizes as this will result in optimal performance from SecureFiles, but it is not necessary.

Write Gather Cache

The Write Gather Cache buffers writes to a SecureFile across calls. This allows for more efficient, larger, contiguous writes irrespective of the size of the buffer the application is sending.

Oracle recommends inserting multiple rows between database commits. If the application is inserting a lot of small/medium sized lobbs, multiple row commits is recommended to utilize advantages provided by SecureFiles namely, larger IOs facilitated by WGC buffering and bulk/contiguous space allocation strategies, space pre-allocation and maintaining IO pipelines across multiple lob writes. With the multiple row commits, SecureFile is able to keep IO going across lobbs/rows, and almost all IOs have enough time to complete before they are reaped, avoiding foreground waits. *Note that the write gather cache size is not tunable.*

Logging

The default setting for SecureFiles is `NOCACHE LOGGING`. The use of a caching strategy with SecureFiles and the performance benefits derived are application dependent. In a scenario where applications do some metadata work and a lot of lob reads, setting up cache could be beneficial. However, consideration needs to be given to the lob data access pattern, lob sizes, buffer cache sizes to derive best performance. SecureFiles introduces `FILESYSTEM_LIKE_LOGGING`, which logs only the SecureFile metadata. The SecureFile data is not logged. This provides a logging mechanism similar to most journaled file systems. It also significantly reduces the amount of data written to the Redo Logs in heavy SecureFile applications. In cases where the lob data is accessed randomly or there are a lot of cache misses, the `FILESYSTEM_LIKE_LOGGING` option could result in improved throughput.

Device Contention

Oracle recommends having more physical devices (spindles) than the number of concurrent writers in the application to avoid device contention. The individual write IOs will be completely independent of each other as long as there are more stripes than concurrent writers.

For applications using LOGGING (NOCACHE LOGGING or CACHE), it is important that the device bandwidth be sufficient to allow for the logs to be written at speeds sufficient to keep up with the SecureFile data writes. Not having sufficient bandwidth on the log devices will cause SecureFile performance to suffer.

Tablespaces and devices

SecureFiles achieves best performance by being allowed to allocate large, contiguous “chunks” for data writes. This also allows efficient pre-fetching for data reads. Oracle recommends that the SecureFile segments be placed in their own tablespaces and devices to allow for maximum “chunk” allocations. By giving SecureFiles space unencumbered by smaller, single block allocations, SecureFiles can achieve larger contiguous writes and reads avoiding disk rotational and seek delays.

Database Parameters

SDU/TDU Size

Oracle recommends setting SDU and TDU to the maximum possible (32KB) to allow for improved client/server interaction for all LOB applications. This is especially important for SecureFiles as it will allow for more efficient streaming of data with the new streaming protocol.

Log_Buffer

Since the DBWriter waits to write the data until the log is 1/3rd full, the log_buffer size needs to be balanced between allowing maximum SecureFiles throughput and allowing other, smaller data to be written. Also, allowing more CPU for the LGWR processes by upping their priority is a must for SecureFiles intensive applications.

Recv_Buf_size/Send_Buf_size

These parameters should be set relatively high. In our internal tests, we’ve found that setting these 512KB has a significant impact on network throughput. This is 16x the recommended SDU/TDU size.

Kernel Parameters

Oracle has found that increasing the network buffer sizes greatly increases SecureFile throughput. Please see your OS documentation for what and how to tune these parameters.

Monitoring

The following instance stats (sysstat or AWR) can be looked at to identify potential problems as well as to identify if things are working as intended. For example, fragmentation due to space allocation or wasteful flushes can be identified using stats such as the number of flushes, the number of allocation chunks, the number of write-ops and the number of write bytes.

Statistic Name	Description
securefile allocation bytes	The number of bytes requested from the space layer in this session.
securefile allocation chunks	The number of "chunks" returned by the space layer in this session. In this case a "chunk" is a contiguous series of disk blocks.
securefile direct read bytes	Number of bytes direct read in this session. This is for NOCACHE lobs.
securefile direct write bytes	Number of bytes direct written in this session. This is for NOCACHE lobs.
securefile direct read ops	Number of times a direct read was issued to disk from SecureFiles. (This is NOT the number of LOB reads issued by the client, but a count of the low-level calls).
securefile direct write ops	Number of times a direct write was issued to disk from SecureFiles. (This is NOT the number of LOB write issued by the client, but a count of the low-level calls).
securefile inode read time	How long reads from disk took in this session. This is a very low level op.
securefile inode write time	How long writes to disk took in this session. This is a very low level op.

Statistic Name	Description
securefile inode ioreap time	Time spent reaping async ios in this session. This is a very low level op.
securefile bytes non-transformed	Number of bytes written by the WGC in this session.
securefile number of non-transformed flushes	This is an internal counter for SecureFiles as is not useful outside of Oracle RDBMS Development.
securefile number of flushes	The number of times the WCG flushed SecureFile data to disk in this session.

7. CONCLUSION

SecureFiles offer numerous technical advances and enable unstructured or file data to be managed in the database with comparable or better performance than traditional file systems. This is achieved through a number of enhancements such as, larger IOs facilitated by WGC buffering and bulk/contiguous space allocation strategies, space pre-allocation, maintaining IO pipelines across multiple lob writes, intelligent pre-fetching and network optimizations. SecureFiles scales well with higher loads and larger file sizes and can run as fast as the hardware permits. In addition, the compression and de-duplication features can provide great space savings. Given the performance we see with this innovative solution, it is time to move the unstructured data into the Oracle database.

8. APPENDIX

When comparing SecureFiles with a file system make sure it is an apples to apples comparison. Comparison needs to be done with the same logging modes (e.g. metadata journaling file system versus file system like logging in SecureFiles), the same network modes (remote NFS/CIFS access versus remote SecureFile, or local file access versus PL/SQL access of SecureFiles).

Major Filesystems

Explanations for settings for some major filesystems are listed below:

NFS + EXT3

For Linux ext3 file system the `data=ordered` is the comparable mode to SecureFiles file system like logging mode. SecureFile file system like logging mode also gives true transactional behavior, complete crash recoverability, consistent read and is multi-versioned unlike NFS+ext3.

NFS 'SYNC | ASYNC'

When set to "sync," Linux server behavior strictly conforms to the NFS protocol. This is default behavior in most other server implementations. When set to "async," the Linux server replies to NFS clients before flushing data or metadata modifying operations to permanent storage, thus improving performance, but breaking all guarantees about server reboot recovery.

Linux **ext3 'data=journal,ordered,writeback'** Default is **data=ordered**.

In `data=writeback` mode, ext3 doesn't do any form of data journaling at all only metadata journaling. Metadata can reach disk before data blocks.

In `data=ordered` mode, ext3 only officially journals metadata, but it logically groups metadata and data blocks into a single unit called a transaction. When it's time to write the new metadata out to disk, the associated data blocks are written *first*.

`data=journal` mode provides full data and metadata journaling. All new data is written to the journal first, and then to its final location. In the event of a crash, the journal can be replayed, bringing both data and metadata into a consistent state.

NFS (sync) + `data=ordered|writeback`, file will be consistent *after close successfully finishes*. However if the crash happens while data blocks were being flushed (i.e. before successful close), data blocks can be corrupted. So if an overwrite of a file was done, the data on disk can end up garbled with old and new data in some random combination.

NFS (sync + data=journaled), gives true data recoverability.

Hence data blocks on ext3 can get corrupted because ext3 does *in place* writes to a block. Oracle SecureFiles on the other hand writes out of place hence after a crash we are guaranteed to be consistent. The only true way to guarantee crash recoverability is data=journal in ext3.

Windows NTFS

Windows NTFS uses transaction logging and recovery to guarantee that the volume structure is not corrupted. NTFS does not do data journaling and only does meta-data journaling. This can cause data corruption at crash times. Programmatically, the application can specify FILE_FLAG_WRITE_THROUGH to push every write to disk or call FlushFileBuffers at close time to flush file buffers. SecureFiles supports this through the NOCACHE and READ CACHE mechanisms. There is no need for your application to explicitly manage the disk write policies as is the requirement with NTFS.

If the program does not use these interfaces directly, user data can be lost due to a system failure. If a system failure does occur, NTFS shows either the previous data, the new data, or zeros.

Solaris 10 ZFS (Zettabyte File System)

ZFS uses copy-on-write where data is always written to a new block on disk before changing the pointers to the data and committing the write. Data blocks are not journaled and only metadata changes are journaled. Thus ZFS is similar to FS_LIKE_LOGGING.

Solaris 7 UFS does metadata journaling

HP-UX JFS or VxFS provides delaylog and log modes. The log intent logging option guarantees that all structural changes to the file system are logged to disk before the system call returns to the application. This is similar to metadata journaling. VxFS provides the mincache=closesync option. In closesync mode, only files that are written during the system crash or shutdown can lose data. Any changes to a file are flushed to disk when the file is closed.

IBM AIX JFS2

AIX JFS2 File System – Does metadata journaling but does not flush data on close.

Network Appliance Filer

The Network Appliance Filer does metadata journaling and copy-on-write. For consistency it uses specialized non-volatile RAM (NVRAM) to keep a log of NFS requests it has processed since the last consistency point. As requests are kept in NVRAM, data is protected from server crashes. However loss of NVRAM can result in data loss.

ORACLE

SecureFile Performance

November 2007

Author: Scott Lynn

Contributing Authors: Vikram Kapoor, Ravi Rajamani, Niraj Srivastava, Hao Wen

Oracle Corporation

World Headquarters

500 Oracle Parkway

Redwood Shores, CA 94065

U.S.A.

Worldwide Inquiries:

Phone: +1.650.506.7000

Fax: +1.650.506.7200

oracle.com

Copyright © 2007, Oracle. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice.

This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission. Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.