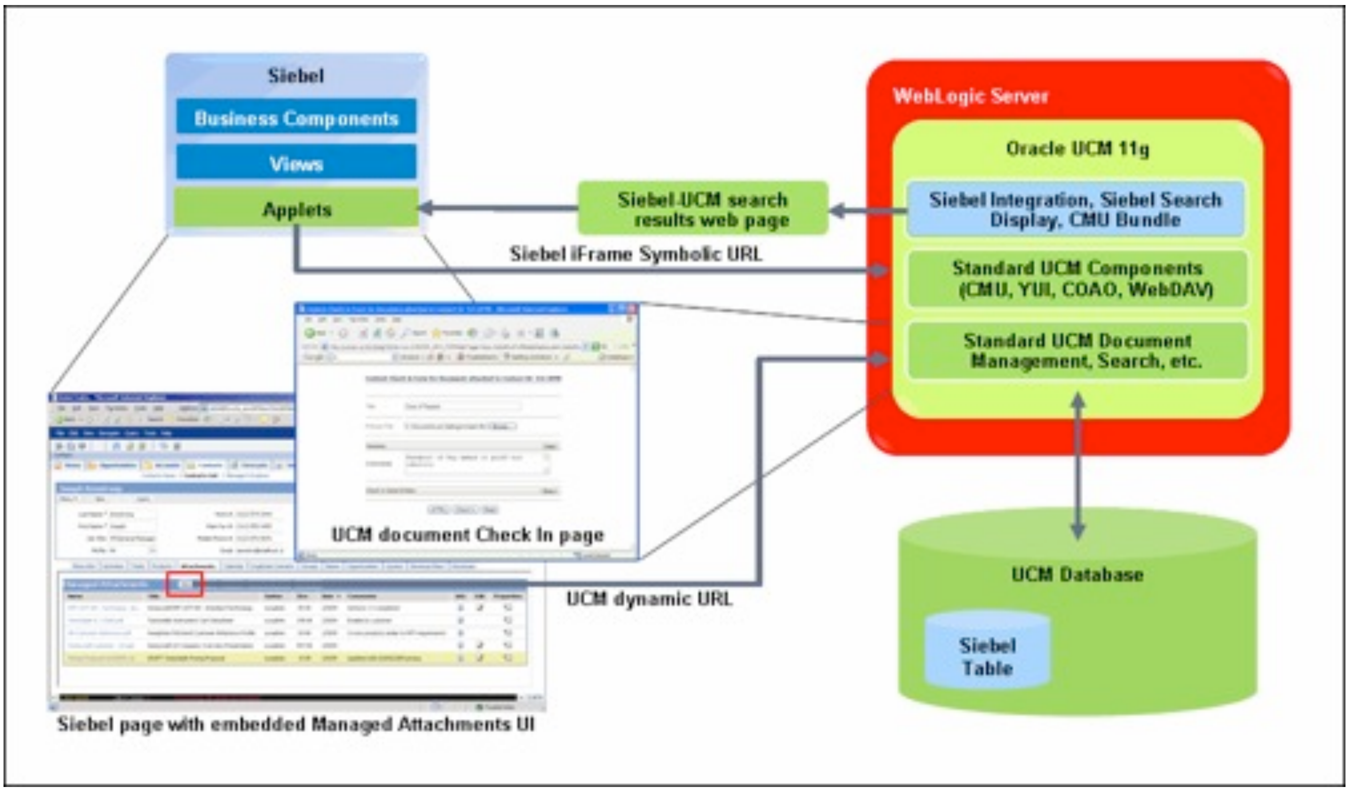
Implementing SecureFile lobs with WebCenter Content,

Lessons learned while pushing the limits of SecureFile lobs

13 januari 2013, Jacco H. Landlust <jacco.landlust@idba.nl>

Preface

In December 2010 I was asked to become the DBA for a project within a financial to upgrade their Siebel Document store. This Siebel Document Store was an Oracle 10g UCM system (based on the original Stellent product) which was interfaced through an Siebel Adapter. The Siebel Adapter for Oracle UCM allows Siebel CRM users to scan, attach, store, and retrieve attachments stored in an Content Server repository. Oracle UCM documents are displayed as managed attachments to Siebel entities in a customizable iFrame within the Siebel application. This Siebel adapter was developed by this financial in coherence with Oracle and is publicly available[1].


Siebel
Business Components
Views
Applets
Siebel-UCM search results web page
Siebel iFrame Symbolic URL
WebLogic Server
Oracle UCM 11g
Siebel Integration, Siebel Search Display, CMU Bundle
Standard UCM Components (CMU, YUI, COAO, WebDAV)
Standard UCM Document Management, Search, etc.
UCM document Check In page
UCM dynamic URL
UCM Database
Siebel Table
Siebel page with embedded Managed Attachments UI

This Siebel Document Store consists out of a lot of moving parts, but for this paper only the 10g UCM part is relevant and therefore only the UCM part will discussed. For those unknown to content management system, a UCM system typically has the following components:

1. Document metadata store, storing keywords and metadata of documents
2. Store for the documents, usually a large NAS or something alike
3. Application Servers performing the business logic (check-in of new documents & revisions, search & retrieve of stored documents)

The 10g UCM product as Oracle was selling it back then, was much alike the original Stellent product that was acquired in 2006 [2].
For the 10g UCM system that was the start point of the project the following components were in place:

- Document metadata stored in an 150GB Oracle 10.2.0.4 extended RAC database with three instances (one preferred and two available instances[3]) and ASM normal rundancy running on OEL 5 on shared hardware

1 http://docs.oracle.com/cd/E14571_01/doc.1111/e10724/c06_axf.htm#CHDBJCHG

2 http://www.oracle.com/us/corporate/acquisitions/stellent/index.html

3 http://docs.oracle.com/cd/B19306_01/rac.102/b28759/configwlm.htm

- Application servers running on 14 AIX nodes that had a massive (20TB) GPFS[4] used as document store containing some 60.000.000 documents

These components had the following characteristics:

- 65.000 users, in peak hours (10 AM until 4 PM) 35.000 online concurrent users that are retrieving and checking-in documents
- Retrieving of documents is based on a one by one basis (one file, based on selection in metadata or by document ID)
- Several high speed scanners that digitally deliver all incoming written communication (mail) to the system (on average 40.000 documents during weekdays, al scanned within 2 hours)
- On average 400.000 new documents per week, resulting in 250GB of data on GPFS
- All documents that are loaded that should be kept for legislation will be converted to PDF/A too, so both the original document as well as the converted document are kept

This UCM 10g system was performing really good, up until a certain point. The bottleneck in the system was introduced by Stellent to ensure that UCM was a database independent product: UCM did not use sequences but a counter table to assign IDs. As you can imagine the database suffered from constant row lock contention when more and more users where added to the system, sometimes resulting into a complete system hang when the high speed scanners where adding documents to the system automatically.

Apart from this bottleneck there was a technical risk in backups. Having a metadata repository and also a filesystem with documents makes it really hard to create a consistent backup. When in some earlier occasion a backup had to be restored the filesystem and metadata database became out of sync. This resulted in a six week rescue operation by the development team to identify the missing documents on filesystem. The heat from the business to the IT department because of this was enormous, demanding for a guarantee this would never happen again.

The bottleneck, the risk of inconsistency between metadata database and filesystem along with the ending support for the 10g product, made a good driver to upgrade the UCM system to a solution where these issues would disappear.

In the 11g version of Oracle UCM, or more specific Oracle Enterprise Content Management; UCM Patchset 2 (11.1.1.3) the counter table bottleneck was removed by Oracle development by introducing sequences. This was all before the product was rebranded to Oracle WebCenter Content. Anyway, upgrading to 11g mend upgrading to an Oracle Fusion Middleware product, based on Oracle WebLogic and the full Fusion Middlware stack with all its caveats. The 11g version of the product also supported usage of SecureFile lobs in the Oracle database (when an Oracle 11g database is used) as document store along with the metadata[5]. With this solution all data is stored in the Oracle database and therefore all the "usual" benefits of archivelogs, point in time recovery and what have you <insert marketing slide here> from the Oracle database are available. So it was agreed that Oracle UCM 11g was the way to go.

Since the project expected a lot of middleware work, they figured a DBA with lots of middleware experience like me would be the ideal candidate to guide the project to the newer release. And this is where I came on board with the development team. Also this is the starting point of this paper, where only the SecureFile lobs part of the project will be discussed.

4 http://www-03.ibm.com/systems/software/gpfs/

5 As of today it is only supported to use one (1) database as store for both metadata and documents. It is not supported to store metadata in one database and documents in some other database in secure files.

Introducing SecureFile lobs

Basic knowledge about lobs is expected for this paper. Tanel Põder has a great presentation about lobs: http://www.slideshare.net/tanelp/oracle-lob-internals-and-performance-tuning , therefore basiclob or generic features will not be discussed again.

Although this paper is not intended to be a SecureFile LOB overview, a short introduction of the topic should be added in my views. This chapter is a summary of the Oracle documentation about SecureFile lobs. [6]

Oracle Database 11g introduced SecureFile lobs to eliminate the distinction between structured and unstructured content storage. SecureFile lobs is a new re-architecture featuring entirely new disk formats, network protocol, space management, redo and undo formats, buffer caching, and I/O subsystem. SecureFile lobs delivers substantially improved performance along with optimized storage for unstructured data inside the Oracle database. LOBs from Oracle Database 10g and prior releases are still supported and will now be referred to as 'BasicFiles'.

SecureFile features are available with the compatibility in the database set to 11.1 or higher. LOBs from 10g and prior releases are still supported under 11.1 compatibility and keyword 'BASICFILE' is used to refer to such LOBs. In 11gR1, 'BASICFILE' is the default storage LOB storage type. Keyword ' SECUREFILE' is used for specifying SecureFile lobs. The default LOB storage can be changed using the db_securefile parameter.

db_securefile=[ALWAYS | FORCE | PERMITTED | NEVER | IGNORE]

ALWAYS – Always attempts to create all LOBs as SECUREFILE LOBs
FORCE - all LOBs created in the system will be created as SECUREFILE LOBs.
PERMITTED – Allows LOBs that use SecureFile lobs.
NEVER – Never Allows LOBs that use SecureFile lobs, they will always be BASICFILE LOBs
IGNORE – Ignore errors that otherwise would be raised by forcing LOBs as SECUREFILE LOBs

SecureFile lobs requires tablespaces managed with ASSM. With SecureFile lobs, Oracle has introduced numerous architectural enhancements for greatly improved performance, scalability, efficient storage and easier manageability

Write-Gather Cache (WGC)

SecureFile lobs uses a new cache that buffers data typically up to 4MB[7] during write operations before flushing or committing to the underlying storage layer. This buffering of in-flight data allows for large contiguous space allocation and large disk I/O. Write performance is greatly improved due to reduced disk seek costs. The WGC is allocated from the buffer cache and maintained on a per-transaction basis. Only one WGC is used for all SecureFile lobs in a transaction. Oracle automatically determines if the WGC needs to be flushed to disk before the 4MB limit is reached. A 'commit' by the user also causes flushing of the WGC to disk

Space Management

Space management for SecureFile lobs is optimized for storing file data inside the Oracle database. An intelligent space manager provides fast allocation of large, contiguous disk blocks and efficient deletion by automatically reclaiming freed space. Space is also pre-allocated based on heuristics of recent demand and this ensures no stalling and consequently no impact on foreground performance. Space management is designed to be self-adaptable and uses new in-memory statistics for efficiently memory and space

[6] Most of the text in this chapter is a citation of the Oracle Database 11g: SecureFile lobs paper : http://www.oracle.com/technetwork/database/features/secure-files/SecureFile lobs-paper-2009-160970.pdf

7 Non tunable buffer

allocation. SecureFile lobs segments require tablespaces managed with automatic segment space management (ASSM).

Reduced Fragmentation

SecureFile lobs, unlike BasicFiles or older LOBs, uses a dynamic CHUNK (one or more Oracle blocks) size setting to maximize contiguous allocations on disk and reduce fragmentation. The user-specified CHUNK value is used as a suggestion along with other factors such as size of the SecureFile, availability of space in the segment to determine the optimal CHUNK size. A fixed CHUNK size setting causes internal fragmentation if it is too large and poor write performance if it is too small. With a dynamic CHUNK size setting, large regions of disk can be allocated and deallocated in a single operation. SecureFile lobs is thus designed from the ground up to be a high performance and storage efficient solution for unstructured or file data of all sizes.

Intelligent Pre-fetching

Read performance is enhanced by read-ahead or pre-fetching data from disk while data is sent over the network. SecureFile lobs keep track of access patterns and intelligently pre-fetches data before the request is actually made. Read latency is reduced by the overlap of the network roundtrip with the disk I/O for the pre- fetched data and throughput is greatly increased.

No LOB Index Contention

Older LOBs use a global per-segment B+ tree index to maintain the inodes, which are data structures that map the logical offsets to the physical disk blocks. This LOB index is used both for access navigation and for managing freed or deleted blocks. This causes contention and significant performance degradation in highly concurrent environments. Unlike older LOBs, SecureFile lobs do not use the LOB index to manage such meta-data. Instead, they use "private" meta-data blocks that are co-located with the data blocks in the LOB segment. This design removes the LOB index contention that has plagued BasicFile LOBs and greatly improves performance, especially under real world insert-delete-reclaim situations.

No High water mark contention

Reclamation of space is done in the background and all the space belonging to a freed SecureFile is reclaimed at once. Thus SecureFile lobs do not suffer from the high water mark contention during space reclamation unlike older LOBs.

Easier Manageability

SecureFile lobs feature self-tuning and intelligent space and memory management algorithms and consequently require lesser number of user-tuned parameters. Specifically, FREEPOOLS, FREELISTS, FREELIST GROUPS & PCTVERSION no longer need to be specified and are ignored for SecureFile lobs. Not only are some of these parameters difficult to tune for unpredictable spikes in workloads, but also cannot be changed online. SecureFile lobs maintains internal statistics to self-tune the space management algorithms and ensures high performance and scalability under diverse workloads.

SecureFile licensable features

Three new features in SecureFile lobs - Deduplication, Compression and Encryption - can be setup independently or as a combination of one or more features. If all three features are turned on, Oracle will perform deduplication first and then compression followed by encryption. These features have to be licensed before you can use them.

Deduplication

This feature eliminates multiple, redundant copies of SecureFile lobs data and is completely transparent to applications. Oracle automatically detects multiple, identical SecureFile lobs data and stores only one copy, thereby saving storage space. Deduplication not only simplifies storage management but also results in significantly better performance, especially for copy operations. Duplicate detection happens within a LOB segment. The lob_storage_clause allows for specifying deduplication at a partition level. However, duplicate detection does not span across partitions or subpartitions for partitioned SecureFile columns.

The DEDUPLICATE keyword is used to turn on deduplicate checking for SecureFile lobs. Oracle uses a secure hash index to detect duplicate SecureFile data and stores a single copy for all identical content. Deduplication is transparent to applications, reduces storage and simplifies storage management. KEEP_DUPLICATES is used to turn off deduplication and retain duplicate copies of SecureFile data. Deduplication is turned off by default.

Compression

Oracle automatically detects if SecureFile data is compressible and will compress using industry standard compression algorithms. If the compression does not yield any savings or if the data is already compressed, SecureFile lobs will automatically turn off compression for such LOBs. Compresion not only results in significant savings in storage but also improved performance by reducing I/O, buffer cache requirements, redo generation and encryption overhead. Compression is performed on the server-side and allows for random reads and writes to SecureFile data. SecureFile provides for varying degrees of compression: MEDIUM (default) and HIGH, which represent a tradeoff between storage savings and latency.
The COMPRESS keyword is used to turn on server-side SecureFile compression. SecureFile compression is orthogonal to table or index compression. Setting table or index compression does not affect SecureFile compression or vice versa. . For partitioned tables, the lob_storage_clause is used for specifying compression at a partition level.

Encryption

In 11g, Oracle has extended the encryption capability to SecureFile lobs and uses the Transparent Data Encryption (TDE) syntax. The database supports automatic key management for all SecureFile columns within a table and transparently encrypts/decrypts data, backups and redo log files. Applications require no changes and can take advantage of 11g SecureFile lobs using TDE semantics.

The ENCRYPT and DECRYPT keywords are used to turn on or turn off SecureFile encryption and optionally select the encryption algorithm to be used. Encryption is performed at the block level and all SecureFile lobs in the specified column are encrypted. Column-level and tablespace-level encryption for SecureFile lobs must not be turned on at the same time because encryption comes at a non-trivial CPU cost; and one of them is unnecessary in this case.

Identified tests

Since SecureFile lobs introduced all these new features, it seemed smart to perform extensive testing on these features before running production on it. When first looking into the topic the licensable features seemed most interesting (compression and deduplication) but when investigating SecureFile LOBs more it showed that features like filesystem like logging and space management where just as interesting to investigate.

In this paper I focus the testing on the features that were actually used in the project. Whenever some feature gave issues in the production setup this will be mentioned.

Test Environment Description

All tests in this document are performed on a VM's which was deployed in VirtualBox on a laptop. This Laptop is a med-2010 MacBook Pro with an 2,66 Ghz Intel Core i7, 8 GB RAM and a 256GB apple SSD.

Virtual Machine

The VM running the database has 2 cores, 4 GB RAM and 3 virtual disks:

- OS disk, storing the OS, 12 GB thin provisioned
- u01 disk, storing the Oracle software, 50 GB thin provisioned
- ASM disk, storing the database files, 40 GB fully allocated.

In the VM 64-bit OEL version 5.8 is installed running a kernel 2.6.39-300.17.3.el5uek. The Oracle software installed is GI and RDBMS, both patched to level 11.2.0.2.6 [8]. The database configured uses ASM external redundancy[9].

The tmpfs is set at 3GB (3072m) to allow larger settings of MEMORY_TARGET [10].

Database

The database used for all tests is running with the following non-default parameters:

```
SQL> SELECT name, value
  2    FROM v$parameter
  3*  WHERE isdefault='FALSE'
SQL> /

NAME                           VALUE
------------------------------ --------------------------------------------------
processes                      150
memory_target                  2147483648
control_files                  +DATA/demodb/controlfile/current.256.801100055,
                               +DATA/demodb/controlfile/current.257.801100057
db_block_size                  8192
compatible                     11.2.0.0.0
db_create_file_dest            +DATA
db_recovery_file_dest          +DATA
db_recovery_file_dest_size     10G
undo_tablespace                UNDOTBS1
remote_login_passwordfile      EXCLUSIVE
db_domain                      area51.local
dispatchers                    (PROTOCOL=TCP) (SERVICE=DEMODBXDB)
audit_file_dest                /u01/app/oracle/admin/DEMODB/adump
audit_trail                    DB
db_name                        DEMODB
open_cursors                   300
diagnostic_dest                /u01/app/oracle

17 rows selected.
```

8 See appendix for the output of opatch lsinventory as the setup was when starting the tests

9 See appendix for configuration settings of ASM

10 See http://www.oracle-base.com/articles/11g/automatic-memory-management-11gr1.php for more information about memory_target and how to resize the tmpfs

As you can see this is a vanilla database configuration, no special parameters are set. The database is running in archivelog mode.

If you are actually going to use the database for production purposes you might want to consider setting the processes parameter according to your specific needs. See the JDBC datasources document in this blogpost for references: http://oraclemva.wordpress.com/2012/11/17/configuring-fusion-middleware-jdbc-data-sources-correctly/ .

The database is created with characterset AL32UTF8, which is listed as requirement in the repository creation utility documentation for Fusion Middleware[11].

Repository

When you want to run a WebCenter Content system, you need to create the repository for the system using Oracle's Repository Creation Assistant (RCU)[12]. The details of RCU itself are out of scope for this document, but the tool does have some caveats that are relevant for SecureFile lobs.

If you do not change any settings, by default the RCU will create two tablespaces for your. repository If you choose the prefix DEMO for your domain, the same prefix will be used for the tablespaces of the WebCenter Content repository. This results in two tablespace that are called are called DEMO_ODC[13] and DEMO_OCS_TEMP. The latter is a temporary tablespace. Both tablespaces use an 8kb blocksize and have a maximum size of 2GB.

```
  1  SELECT tablespace_name, block_size, max_size, bigfile,
  2         segment_space_management
  3    FROM dba_tablespaces
  4*  WHERE tablespace_name like 'DEMO%'
SQL> /

TABLESPACE_NAME                BLOCK_SIZE   MAX_SIZE BIG SEGMEN
------------------------------ ---------- ---------- --- ------
DEMO_OCS                             8192 2147483645 NO  AUTO
DEMO_OCS_TEMP                        8192 2147483645 NO  MANUAL

2 rows selected.
```

For small scale testing this configuration could be sufficient, since the setup can be fairly small. When you are planning to store terabytes of data you might want to switch to bigfile tablespaces to minimize the number of datafiles on the system and increase manageability. Running a 100 TB database with smallfile datafiles would imply over 6000 datafiles, a horrific scenario for most administrators. Therefore it is advisable to use bigfile tablespaces. This paper does not investigate the existence of any performance differences between smallfile and bigfile tablespaces.

11 http://docs.oracle.com/cd/E29505_01/doc.1111/e15722/db.htm#CIABGEBB

12 http://docs.oracle.com/cd/E12839_01/doc.1111/e14259/rcu.htm

13 OCS is for Oracle Content Server

The following statement was used to create a tablespace for testing SecureFile lobs:

```
  1  CREATE BIGFILE TABLESPACE hotsos_demos
  2    DATAFILE SIZE 100M AUTOEXTEND ON MAXSIZE unlimited
  3    EXTENT MANAGEMENT LOCAL
  4*   SEGMENT SPACE MANAGEMENT AUTO
SQL> /

Tablespace created.
```

As you can see the tablespace has ASSM, this is mandatory for SecureFile lobs. Amongst this document scripts are provided in an archive. The setup.sql script creates the user and grants privileges that are used during the tests described in this document.

The table used for storing content in the database, called filestorage, is created from WebCenter content automatically when you configure WebCenter Content to use a database store for it's content. Since setting up a full WebCenter Content system is beyond the scope of this paper, the DDL for the table is provided here:

```
CREATE TABLE FILESTORAGE
   (    DID             NUMBER(*,0) NOT NULL ENABLE,
        DRENDITIONID    VARCHAR2(30 CHAR) NOT NULL ENABLE,
        DLASTMODIFIED   TIMESTAMP (6),
        DFILESIZE       NUMBER(*,0),
        DISDELETED      VARCHAR2(1 CHAR),
        BFILEDATA       BLOB,
        CONSTRAINT PK_FILESTORAGE PRIMARY KEY (DID, DRENDITIONID)
   )
  TABLESPACE DEMO_OCS
/
```

The BFILEDATA BLOB is created as a basic lob by default. This is caused by the default setting of the db_securefile[14] parameter: PERMITTED . If the setting of db_securefile was set to ALWAYS the BFILEDATA BLOB would have been created as SecureFile blob.

For testing SecureFile lobs not all functionality is needed. Therefore the columns that are not relevant for the tests are removed from the filestorage table. This leads to the following test table that is the core test table for all tests described in this paper:

```
SQL> CREATE TABLE filestorage
  2  ( did          NUMBER NOT NULL,
  3    insert_date  DATE DEFAULT SYSDATE NOT NULL,
  4    bfiledata    BLOB,
  5    CONSTRAINT pk_filestorage PRIMARY KEY (did)
  6  )
  7  TABLESPACE hotsos_demos
  8  LOB (bfiledata) STORE AS SECUREFILE
  9    (
 10      TABLESPACE hotsos_demo_lobs DISABLE STORAGE IN ROW NOCACHE
 11    )
 12  /

Table created.
```

Every test case has it's own specific small changes in the SecureFile lobs setup, a new create statement is listed with every test. This main statement is just to give an some general idea of what the filestorage table looks like.

14 http://docs.oracle.com/cd/E11882_01/server.112/e25513/initparams068.htm#REFRN10290

SecureFile lob specific tests

This chapter describes the actual tests that were performed to investigate the usability of SecureFile lobs as storage in a WebCenter Content environment.

ASM diskgroup compatibility

If you advance disk group compatibility, then you could enable the creation of files that are too large to be managed by a previous Oracle database release. You need to be aware of the file size limits because replicated sites cannot continue using the software from a previous release to manage these large files.

The table below shows the maximum ASM file sizes supported for `COMPATIBLE.RDBMS` settings with a 1 MB AU_SIZE[15]. Note that memory consumption is identical in all cases: 280 MB. For Oracle Database 11g, the number of extent pointers for each size is 16800, with the largest size using the remainder. The RDBMS instance limits file sizes to a maximum of 128TB.

For Oracle ASM in Oracle Database 11g, 10.1 is the default setting for the COMPATIBLE.RDBMS attribute when using the SQL CREATE DISKGROUP statement, the ASMCMD mkdg command, ASMCA Create Disk Group page, and Oracle Enterprise Manager Create Disk Group page.

Redundancy	COMPATIBLE.RDBMS=10.1	COMPATIBLE.RDBMS=11.1
External	16TB	140PB
Normal	5.8TB	23PB
High	3.9TB	15PB

The table shows that Oracle Database 10g can only support a file size of up to 16 TB for external redundancy. If you advance the RDBMS compatibility to`11.1`, then a file can grow beyond 35 TB.

When the test setup was created this parameter compatible.rdbms parameter wasn't checked, so when the test setup reached 16 TB in its datafile the following error was thrown:

```
ORA-15095: reached maximum ASM file size (16384 GB)
```

You can set the diskgroup attribute compatible.rdbms to a value greater than 10.1 with alter diskgroup command (v$asm_attribute), but the file size limit will be the old limit for the existing files. The new value of compatible.rdbms will be effective for the new files only.

The workaround suggested by Oracle Support is to move the file to an other diskgroup with a compatible.rdbms set to a value greater than 10.1 . Since we only ran into this issue on our testing environment, we did not actually test if moving the file solves the issue. This was mainly caused by a lack of diskspace.

15 http://docs.oracle.com/cd/E11882_01/server.112/e18951/asmdiskgrps.htm#CHDCGGED

I am very sorry to say that we hit this problem while building our load-test environment because it shows that we didn't check all parameters needed. I'm also very happy that we hit this misconfiguration because that resulted in not hitting this in the production setup. The production setup grows approximately 1 to 1.5 TB per week and is setup on ASM normal redundancy. That would mean approximately six week running production load before the system would have come to a grinding stop.

If you happen to reach the max datafile size and you are using a block change tracking file, you will also encounter this error :

```
ORA-600: internal error code, arguments: [krccfl_bitmap_too_small], [19],
[4294340465], [4], [4366], [4366], []
```

This ORA-600 is documented at My Oracle Support (bug 11703092) but was suspended.

Impact of blocksize

Selecting an appropriate blocksize is relevant when sizing a database. When a blocksize is chosen that is too large, you might end up with lots of wasted space. So in general there is no statement to make about what is the best blocksize for your system, it al depends. If you have sample data available you could load the data into a table and check the blocksize by running a query like[16]:

```
SQL> SELECT SUM(blocks_4)*4/1024 AS size_in_mb_4k_bl,
  3         SUM(blocks_8)*8/1024 AS size_in_mb_8k_bl,
  4         SUM(blocks_16)*16/1024 AS size_in_mb_16k_bl,
  5         SUM(blocks_32)*32/1024 AS size_in_mb_32k_bl
  6    FROM ( SELECT did,
  7                  CEIL(DBMS_LOB.GETLENGTH(bfiledata)/(4*1024)) AS blocks_4,
  8                  CEIL(DBMS_LOB.GETLENGTH(bfiledata)/(8*1024)) AS blocks_8,
  9                  CEIL(DBMS_LOB.GETLENGTH(bfiledata)/(16*1024)) AS blocks_16,
 10                  CEIL(DBMS_LOB.GETLENGTH(bfiledata)/(32*1024)) AS blocks_32
 11             FROM filestorage
 12*        )
SQL> /

SIZE_IN_MB_4K_BL SIZE_IN_MB_8K_BL SIZE_IN_MB_16K_BL SIZE_IN_MB_32K_BL
---------------- ---------------- ----------------- -----------------
      7107.21875         7375.875        7887.29688         9196.6875
```

This gives you a rough idea of the total storage that you would need when the database is setup in a certain blocksize. Please be aware that this query gives an significant hit on your database, you might want to think about when to run this query.

When you are migrating from a UCM 10g system to a WebCenter Content 11g system with a database storage provider, or when you are considering adding a database storage provider in WebCenter Content 11g, you can use internal data from UCM / WebCenter Content to calculate the optimal blocksize.

While checking in documents into WebCenter Content (or UCM 10g) information is stored in the documents table. Column dfilesize stores the actual filesize of the file. By querying this table you can look for an optimal blocksize before loading data into the database. You have to be aware though that this is the uncompressed size.

16 data in this query is the result of the last deduplication test

Partitioning

Lobs and SecureFile lobs are similar to regular tables regarding partitioning. Especially when using a large setup, partitioning can ease backup and recovery, e.g. by usage of read only tablespaces for “old” data. Performance-wise concurrency issues can be avoided by introducing hash partitions.

Partitioned lobs are fully supported, as long as all lob segments partitions have the same blocksize. The blocksize of the lobsegment can differ from the table blocksize though.

Deduplication works within the partition, therefore partitioning will make inserts/updates faster when using deduplication. Duplicates in an other partition will not be found though, and therefore you will possibly store more physical data in your database than you would predict.

For the production setup of the Siebel Document Store a created_data column was introduced on the filestorage table. This allowed for interval partitions based on weekly basis. Next to these interval partitions a subpartition template on a hash of the primary key was introduced to ensure that we can do the maximum number of concurrent inserts.

```
CREATE TABLE filestorage
   (    did           NUMBER               NOT NULL,
        drenditionid  VARCHAR2(30)         NOT NULL,
        dadddate      DATE DEFAULT SYSDATE NOT NULL,
        dlastmodified TIMESTAMP(6),
        dfilesize     NUMBER,
        disdeleted    VARCHAR2(1),
        bfiledata     BLOB,
        CONSTRAINT pk_filestorage PRIMARY KEY (did, drenditionid)
   )
  TABLESPACE hotsos_demos
  LOB (bfiledata) STORE AS SECUREFILE filestorage_bfiledata
    (TABLESPACE hotsos_demo_lobs DISABLE STORAGE IN ROW NOCACHE COMPRESS MEDIUM
KEEP_DUPLICATES)
  PARTITION BY RANGE (dadddate) INTERVAL (numtodsinterval(7,'DAY'))
  SUBPARTITION BY HASH (did,drenditionid)
    SUBPARTITION TEMPLATE (
      SUBPARTITION P_UCM_LOBS1 LOB (bfiledata) STORE AS SECUREFILE
"FS_BFILEDATA1" (TABLESPACE hotsos_demo_lobs1) TABLESPACE hotsos_demo_lobs1,
      SUBPARTITION P_UCM_LOBS2 LOB (bfiledata) STORE AS SECUREFILE
"FS_BFILEDATA2" (TABLESPACE hotsos_demo_lobs2) TABLESPACE hotsos_demo_lobs1,
      SUBPARTITION P_UCM_LOBS3 LOB (bfiledata) STORE AS SECUREFILE
"FS_BFILEDATA3" (TABLESPACE hotsos_demo_lobs3) TABLESPACE hotsos_demo_lobs1,
      SUBPARTITION P_UCM_LOBS4 LOB (bfiledata) STORE AS SECUREFILE
"FS_BFILEDATA4" (TABLESPACE hotsos_demo_lobs4) TABLESPACE hotsos_demo_lobs1
    )
  ( PARTITION P_20121120 VALUES LESS THAN (TO_DATE('2012-11-20','YYYY-MM-
DD') ) )
/
```

Setting up the filestorage table like this enables moving tablespaces around to other diskgroups, possibly on other storage devices, and setting tablespaces to read only. Although moving might not be the correct word. If you would try to move a partiton:

```
SQL> ALTER TABLE filestorage
  2    MOVE PARTITION p_medium TABLESPACE backup_ts
  3  /
  MOVE PARTITION p_medium TABLESPACE backup_ts
                 *
ERROR at line 2:
ORA-14257: cannot move partition other than a Range, List, System, or Hash
partition
```

Trying to move a subpartition causes an ORA-00600 error:

```
SQL> ALTER TABLE filestorage
  2   MOVE SUBPARTITION SYS_SUBP6030925 TABLESPACE ucm_lobs_2012_1
  3   LOB (bfiledata) STORE AS LOGSEGMENT (TABLESPACE ucm_lobs_2012_1);
  4  /

alter table filestorage move subpartition SYS_SUBP6030925 tablespace
ucm_lobs_2012_1 lob (BFILEDATA) store as lobsegment (tablespace ucm_lobs_2012_1)
                      *
ERROR at line 1:
ORA-00603: ORACLE server session terminated by fatal error
ORA-00600: internal error code, arguments: [kkpod nextFrag], [10], [20], [1],
[1], [93891], [], [], [], [], [], []
Process ID: 5003
Session ID: 1791 Serial number: 17847
```

This ORA-600 error is caused by a bug:

```
Bug 12325317  ORA-600 [kkpod nextFrag] on ALTER TABLE .. MOVE SUBPARTITION on
INTERVAL partitioned table
```

There is no workaround for this bug for existing partitions, but a patch is available: 12325317. To apply this patch you have to upgrade to at least 11.2.0.2.8. If you don't want to patch or upgrade, you could alter the subpartition template in such a way that it will store new subpartitions in some other tablespace. This is manual maintenance though.

Shared I/O Pool

The database uses the shared I/O pool to perform large I/O operations on SecureFile LOBs. The shared I/O pool uses shared memory segments. If this pool is not large enough or if there is not enough memory available in this pool for a SecureFile LOB I/O operation, Oracle uses a portion of PGA until there is sufficient memory available in the shared I/O pool. Hence it is recommended to size the shared I/O pool appropriately by monitoring the database during the peak activity. Relevant initialization parameters:_shared_io_pool_size and _shared_iop_max_size. [17]

When you do not use SecureFile lobs that parameter defaults to 0. The parameter used to default to 6.25% of the buffer_cache size. From 11.2.0.2 onwards the parameter is derived by 25% of the buffer_cache size. The value is automatically setup upon the moment you insert your first SecureFile lob. The maximum size of the shared_io_pool size is 512MB which is defined by _shared_iop_max_size:

```
SQL> SELECT nam.ksppinm as name, val.KSPPSTVL as value,
  2         nam.ksppdesc  as description
  3    FROM x$ksppi nam,
  4         x$ksppsv val
  5   WHERE nam.indx = val.indx
  6*    AND UPPER(nam.ksppinm) LIKE '%SHARED_IOP_MAX%'
SQL> /

NAME                           VALUE      DESCRIPTION
------------------------------ ---------- ------------------------------------
_shared_iop_max_size           536870912  maximum shared io pool size

1 rows selected.
```

In a new database that never would have used SecureFile lobs you could see SGA info like this:

17 https://blogs.oracle.com/mandalika/entry/oracle_rdbms_generic_large_object

```
SQL> SELECT *
  2    FROM v$sgainfo
   3* /

NAME                                  BYTES RES
-------------------------------- ---------- ---
Fixed SGA Size                      2233336 No
Redo Buffers                        5541888 No
Buffer Cache Size                 444596224 Yes
Shared Pool Size                  176160768 Yes
Large Pool Size                     4194304 Yes
Java Pool Size                      4194304 Yes
Streams Pool Size                         0 Yes
Shared IO Pool Size                       0 Yes
Granule Size                        4194304 No
Maximum SGA Size                 1068937216 No
Startup overhead in Shared Pool    79691776 No
Free SGA Memory Available         432013312

12 rows selected.
```

So if I would create the filestorage table and insert one lob (not an empty lob!), the Shared IO Pool is set:

```
SQL> CREATE TABLE filestorage
  2  ( did              NUMBER NOT NULL,
  3    bfiledata        BLOB,
  4    CONSTRAINT pk_filestorage_l PRIMARY KEY (did)
  5  )
  6  TABLESPACE hotsos_demos
  7  LOB (bfiledata) STORE AS SECUREFILE
  8    (
  9      TABLESPACE hotsos_demo_lobs DISABLE STORAGE IN ROW NOCACHE
 10*   )
/

Table created.
```

```
SQL> DECLARE
  2    l_file        BFILE;
  3    l_blob        BLOB;
  4  BEGIN
  5
  6    -- Open the file
  7    l_file := BFILENAME('FILES', 'largest.pdf');
  8    DBMS_LOB.OPEN(l_file, DBMS_LOB.LOB_READONLY);
  9
 10    INSERT INTO filestorage (did, bfiledata )
 11      VALUES ( 1, empty_blob() )
 12      RETURNING bfiledata INTO l_blob;
 13
 14    DBMS_LOB.OPEN(l_blob, DBMS_LOB.LOB_READWRITE);
 15    DBMS_LOB.LOADFROMFILE(DEST_LOB => l_blob,SRC_LOB => l_file,AMOUNT   =>
DBMS_LOB.GETLENGTH(l_file));
 16    DBMS_LOB.CLOSE(l_blob);
 17
 18    DBMS_LOB.CLOSE(l_file);
 19
 20  END;
 21  /

PL/SQL procedure successfully completed.
```

I can find that memory has been resized:

```
SQL> SELECT component, min_size, max_size, current_size, last_oper_type
  2    FROM v$sga_dynamic_components
  3*  WHERE last_oper_type != 'STATIC'
SQL> /

COMPONENT                   MIN_SIZE   MAX_SIZE CURRENT_SIZE LAST_OPER_TYP
------------------------- ---------- ---------- ------------ -------------
DEFAULT buffer cache       335544320  444596224    335544320 SHRINK
Shared IO Pool                     0  109051904    109051904 GROW

2 rows selected.
```

The automatic memory management takes care of this Shared IO Pool setting. Some administrators like a little more control on memory management themselves. This can be done by setting a value for, in this case, db_buffer_cache that will be treated as a minimum value.

So on a brand new database with no memory_target but a sga_target (768M) and pga_aggregate_target (256M) set, and if I would set the buffer cache to a value:

```
SQL> SELECT *
  *    FROM v$sgainfo
  3* /

NAME                                  BYTES RES
-------------------------------- ---------- ---
Fixed SGA Size                      2230768 No
Redo Buffers                        6746112 No
Buffer Cache Size                 570425344 Yes
Shared Pool Size                  213909504 Yes
Large Pool Size                     4194304 Yes
Java Pool Size                      4194304 Yes
Streams Pool Size                         0 Yes
Shared IO Pool Size                       0 Yes
Granule Size                        4194304 No
Maximum SGA Size                  801701888 No
Startup overhead in Shared Pool    83886080 No
Free SGA Memory Available                 0

12 rows selected.

SQL> ALTER SYSTEM SET db_cache_size = 570426368;

System altered.
```

Create the filestorage table and insert a record just like before and query v$sga_dynamic_components:

```
SQL> SELECT component, min_size, max_size, current_size, last_oper_type
  2    FROM v$sga_dynamic_components
  3   WHERE last_oper_type != 'STATIC'
  4  /

COMPONENT                        MIN_SIZE   MAX_SIZE CURRENT_SIZE LAST_OPER_TYP
------------------------------ ---------- ---------- ------------ -------------
shared pool                     209715200  213909504    209715200 SHRINK
large pool                              0    4194304      4194304 GROW
DEFAULT buffer cache            570425344  574619648    574619648 GROW

3 rows selected.
```

I see that the Shared IO Pool is not set automatically. This makes you wonder what happens if you modify the size of your buffer cache after your Shared IO Pool already is configured.

```
SQL> SELECT *
  2    FROM v$sgainfo
  3  /

NAME                                  BYTES RES
-------------------------------- ---------- ---
Fixed SGA Size                      2228200 No
Redo Buffers                       55283712 No
Buffer Cache Size                 771751936 Yes
Shared Pool Size                  520093696 Yes
Large Pool Size                    16777216 Yes
Java Pool Size                     16777216 Yes
Streams Pool Size                  16777216 Yes
Shared IO Pool Size               184549376 Yes
Granule Size                       16777216 No
Maximum SGA Size                 2137886720 No
Startup overhead in Shared Pool    83886080 No
Free SGA Memory Available         738197504

12 rows selected.
```

```
SQL> ALTER SYSTEM SET db_cache_size = 771752960;

System altered.

  1  SELECT component, min_size, max_size, current_size, last_oper_type
  2    FROM v$sga_dynamic_components
  3*  WHERE last_oper_type != 'STATIC'
SQL> /

COMPONENT                MIN_SIZE    MAX_SIZE  CURRENT_SIZE LAST_OPER_TYP
----------------------- ----------- --------- ------------ -------------
shared pool              318767104   520093696 318767104    SHRINK
DEFAULT buffer cache     587202560   788529152 788529152    GROW

2 rows selected.
```

Inserting a lob doesn't change this. So this means that even you start manually managing your db_cache_size you also would have to manager your Shared IO Pool (_shared_io_pool) size.

When a session is unable to find free memory in the Shared IO Pool, PGA memory would be used. To see PGA memory allocations you can use the V$SECUREFILE_TIMER view[18] which gets an entry each time memory is allocated out of the PGA. These timer values are collected per session.

When you are not using automatic memory management you would have to manage the Shared IO Parameter yourself. , Personally unless there are real memory issues (e.g. ORA-4031) I would not recommend tuning the Shared IO Pool settings myself, especially because you would have to tune them via underscore parameters:

```
SQL> SELECT nam.ksppinm as name, val.KSPPSTVL as value,
  2         nam.ksppdesc  as description
  3    FROM sys.x$ksppi nam,
  4         sys.x$ksppsv val
  5   WHERE nam.indx = val.indx
  6*    AND UPPER(nam.ksppinm) LIKE '%SHARED_IO%'
SQL> /

NAME                           VALUE      DESCRIPTION
------------------------------ ---------- ------------------------------------
__shared_io_pool_size          0          Actual size of shared IO pool
_shared_io_pool_size           0          Size of shared IO pool
_shared_iop_max_size           536870912  maximum shared io pool size
_shared_io_pool_buf_size       1048576    Shared IO pool buffer size
_shared_io_pool_debug_trc      0          trace kcbi debug info to tracefile
_shared_io_set_value           FALSE      shared io pool size set internal value
                                          - overwrite zero user size
6 rows selected.
```

18 http://docs.oracle.com/cd/E11882_01/server.112/e25513/dynviews_3002.htm#REFRN30511

Caching

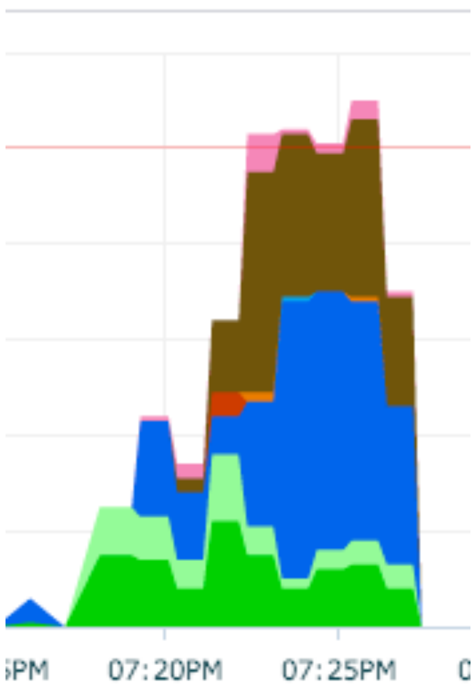
Enable caching except for cases where caching LOBs would severely impact performance for other online users, by forcing these users to perform disk reads rather than cache hits[19]. Consider bypassing the database buffer cache (NOCACHE) if large number of LOBs are stored and not expected to be retrieved frequently. This strategy is the same for basic lobs and for SecureFile lobs. The default value is nocache.

Because of the high currency of users in the Siebel Document System all retrieving different documents it is not likely that caching will benefit performance. By default 25% of the buffer cache is already used for SecureFile lob operations (Shared IO Pool). Spending more seemed useless. I did not test this statement because I could not setup a test environment that was production like for this purposes.

When configuring caching you should not forget that CACHE and NOLOGGING are not supported together. CACHE and FILESYSTEM_LIKE_LOGGING are not supported together either [20].

Redo logs & log_buffer

Default a 11.2.0.2 database a 3 redo log groups with each a 100MB logfile. This is insufficient for any type of large fast-loading environment. A typical view on the performance page for a default configured database would look like this:



Zooming in on the brown color, would show two main wait events:

- log buffer space
- log file sync (checkpoint incomplete)

This suggests that the size of the log_buffer should be adjusted to gain a higher performance.

19 http://www.oracle.com/technetwork/articles/lob-performance-guidelines-128437.pdf

20 http://docs.oracle.com/cd/B28359_01/appdev.111/b28393/adlob_smart.htm#autoId8

The current value of the log_buffer in my test environment is:

```
SQL> SELECT value, isdefault
  2    FROM v$parameter
  3*  WHERE name = 'log_buffer'
SQL> /

VALUE                ISDEFAULT
-------------------- ---------
4702208              TRUE

1 row selected.
```

The value of 4702208 is defined automatically for this system. The log buffer size depends on the number of redo strands in the system and on the size of the SGA, CPU count, and the OS on 32 or 64 bit. One redo strand is allocated for every 16 CPUs and has a default size of 2 MB. Oracle allocates a minimum of 2 redo strands per instance. When the log buffer size is not specified, any remaining memory in the redo granules is given to the log buffer[21].

> "The concept of a strand is to ensure that the redo generation rate for an instance is optimal and that when there is some kind of redo contention then the number of strands is dynamically adjusted to compensate. The initial allocation for the number of strands depends on the number of CPU's and is started with 2 strands with one strand for active redo generation.
>
> For large scale enterprise systems the amount of redo generation is large and hence these strands are *made active* as and when the foregrounds encounter this redo contention (allocated latch related contention) when this concept of dynamic strands comes into play.
>
> Oracle 10g has some major changes in the mechanisms for redo (and undo), which seem to be aimed at reducing contention. Instead of redo being recorded in real time, it can be recorded 'privately' and pumped into the redo log buffer on commit." [22]

To check the number of strands we need to look for the _log_parallelism_max parameter:

```
SQL> SELECT nam.ksppinm as name, val.KSPPSTVL as value,
  2         nam.ksppdesc  as description
  3    FROM sys.x$ksppi nam,
  4         sys.x$ksppsv val
  5   WHERE nam.indx = val.indx
  6*    AND UPPER(nam.ksppinm) LIKE '%LOG_PARALLE%'
SQL> /

NAME                      VALUE     DESCRIPTION
------------------------- --------- -----------------------------------
_log_parallelism_max      2         Maximum number of log buffer strands
_log_parallelism_dynamic  TRUE      Enable dynamic strands

2 rows selected.
```

21 http://docs.oracle.com/cd/E11882_01/server.112/e25513/initparams131.htm#REFRN10094

22 See " Alert Log Messages: Private Strand Flush Not Complete [ID 372557.1]

The actual size of the buffers is shown in x$kcrfstrand, resulting in the default value of 4702208:

```
SQL> break on report
SQL> compute sum of strand_size_kcrfa on report
SQL> SELECT indx,strand_size_kcrfa
  2    FROM x$kcrfstrand
  3*  WHERE last_buf_kcrfa != '00'
SQL> /

      INDX STRAND_SIZE_KCRFA
---------- -----------------
         0           2351104
         1           2351104
           -----------------
sum                  4702208

2 rows selected.
```

If you want to check if the private strands are causing the waitevent, you can query x$ktcxb to find out:

```
SQL> SELECT DECODE(BITAND(ktcxbflg, 4096),0,1,0) used_private_strand, COUNT(*)
  2    FROM x$ktcxb
  3   WHERE BITAND(ksspaflg, 1) != 0
  4     AND BITAND(ktcxbflg, 2) != 0
  5   GROUP BY BITAND(ktcxbflg, 4096)
  6  /

USED_PRIVATE_STRAND   COUNT(*)
------------------- ----------
                  0          1

1 row selected.
```

No private strand is used, so luckily no fiddling with hidden parameters to improve throughput. Just enlarging the value for log_buffer should improve performance.

According to documentation the log_buffer ranges between 2MB and 256MB[23] . Normally when I want to resize the log_buffer I query v$instance_recovery to find the optimal size:

```
SQL> SELECT optimal_logfile_size
  2*   FROM v$instance_recovery
SQL> /

OPTIMAL_LOGFILE_SIZE
--------------------


1 row selected.
```

This returns null because fast_start_mttr_target is not set on my testsystem. So that doesn't help me.

23 http://docs.oracle.com/cd/E11882_01/server.112/e25513/initparams131.htm#REFRN10094

One could make a case of always setting the log_buffer to 256MB, because the logbuffer is flushed upon every commit anyway. The only penalty is memory loss, but that gives you the capability to handle large transactions during peak operations in return. Especially in a system like a WebCache content system the burst (peak load) can be enormous. Therefore I push the log_buffer to 256MB.

```
ALTER SYSTEM SET log_buffer=268435456 SCOPE=spfile
```

You do have to restart the database to make this parameter active.

When raising the log_buffer size, you should also check the size of the redo logs.

```
  1  SELECT group#, bytes/1024/1024 as size_in_mb
  2*   FROM v$log
SQL> /

    GROUP# SIZE_IN_MB
---------- ----------
         1        100
         2        100
         3        100

3 rows selected.
```

WIth a 100MB size (default) for the redo logs, the log_buffer fills the redo log over 33%. This will inflict a logswitch automatically and therefore log file switch contention is eminent. To prevent this the logs are recreated with a 1024MB size. The old redo logs are dropped:

```
ALTER DATABASE ADD LOGFILE GROUP 10 SIZE 1024M
/
ALTER DATABASE ADD LOGFILE GROUP 11 SIZE 1024M
/
ALTER DATABASE ADD LOGFILE GROUP 12 SIZE 1024M
/

DECLARE
  lv_loggroup v$log.group#%type := 0;
BEGIN
  WHILE lv_loggroup != 10 LOOP
    EXECUTE IMMEDIATE 'ALTER SYSTEM SWITCH LOGFILE';
    SELECT group#
      INTO lv_loggroup
      FROM v$log
     WHERE status = 'CURRENT';
  END LOOP;
END;
/

ALTER SYSTEM CHECKPOINT GLOBAL
/

BEGIN
  FOR r_loggroups IN ( SELECT group# FROM v$log WHERE group# not in (10,11,12) )
LOOP
    EXECUTE IMMEDIATE 'ALTER DATABASE DROP LOGFILE GROUP ' ||
                      r_loggroups.group# ;
  END LOOP;
END;
/
```

Space management

From MOS: SMCO (Space Management Coordinator) For Autoextend On Datafiles And How To Disable/Enable [ID 743773.1]

> "SMCO coordinates the following space management tasks. It performs proactive space allocation and space reclamation. It dynamically spawns slave processes (Wnnn) to implement the task.
>
> - Tablespace-level space (Extent) pre-allocation.
>
>   Pre-allocation here refers to datafile extention, Datafile extension happens when a space request (extent allocation) operation generally triggered via Insert / loading to a segment does not find contiguous space in the tablespace, the session will extend the file by the next increment set for the datafile and will continue with the space request or Extent allocation.
>
>   For SMCO to autoextend a datafile, the AUTOEXTEND should be set to ON for the datafile. SMCO decides to expand the tablespace based on history, extension is split evenly across all datafiles in the tablespace which have not reached their maxsize and are still limited to 10% of the full tablespace size in one hourly SMCO wakeup.
>   (Full tablespace size = Sum of datafile sizes at any given instance of time.)
>
> Apart from the above mentioned task, the SMCO process is also responsible for performing the following tasks.
>
> - updating block and extents counts in SEG$ for locally managed tablespaces after adding an extent (from unpublished Bug 12940620)
> - Securefile lob segment pre-extension.
> - Securefile lob segment in-memory dispenser space pre-allocation.
> - Securefile lob segment space reclamation (moving free chunks from uncommitted free space area to committed free space area).
> - Temporary segment space reclamation.
>
> The main advantage is that sessions need not wait for reactive space allocation/deallocation operations as this is proactively done, and hence a gain in performance."

My friend Bernhard de Cock-Buning wrote a nice blogpost about this feature[24], which was originated by the issues we suffered on the system after go-life and on which we worked on in a team of DBA's. I see no direct need to rephrase his words:

> "SMCO is responsible for space management tasks. This means when the available spaces in a tablespace/datafile is less than 5% of the datafile size is will automatically starts to preallocate 5% until the maximum defined size. Beside space management for preallocation it also take care of secure file segment related operations.
>
> The pre-allocation will be done in the background and will be based on the autoextend next size. Assume SMCO want to allocate 100GB, it will do this in a loop of 1000 operations each time allocating 100MB.
>
> But if this is 5% of the datafile, how big is the datafile? If the datafile is just 1GB, a pre-allocation will be 50MB, which will be done in one operation. So get the HW lock, extend, release lock, transactions can continue. If your datafile is 10TB, preallocation means 500GB to add to the datafile. Which means 500 preallocation steps are required and this is where the contention starts. You can extend your file with another action like your insert statement, which in that case will slow down and TX wait

24 http://blog.grid-it.nl/index.php/2012/10/22/hwtx-contention-caused-by-smco-background-process/

event is seen, waiting for the HW to be ready. So if this means you need to wait around 500 times before your insert can complete you can be a disappointed user.

Now this is clear are you happy?

- Well if you look this from a point of view of the self- healing database, it is nice. If you look at this in a way of in control you probably not.
- As long as SMCO is busy and your transactions requires space in the datafile HW and TX wait events are seen, which can result in a performance degradation.
- When you have large files, 5% pre-allocation can be a lot and a waste of space.

But at least it can explain cases where you notice the required disk space of your database is increasing very fast although the application isn't doing much of inserts. If notices during particular times HW/TX contention you can start to wonder if SMCO is active.

Can you stop SMCO to do his job?
As soon as SMCO start to pre-allocate space you cannot stop this process. There is the parameter _enable_space_preallaction, which can be set to a value, but this can have other side effects which are unwanted. And settings underscore parameters is also not a supported way.

So what can you do if you don't want SMCO to kick-inn?
Make sure proper monitoring is in place and get alerted when around 90% of the space if used. Then start to pre-allocate the space yourself, and/or resize the datafile to the required capacity beforehand. "

The claims made in Bernhards article are supported by a couple of MOS notes:

- AUTOEXTEND Grows To Full Size Without Reason [ID 1459097.1]
- Wnnn processes consuming high CPU [ID 1492880.1]
- Bug 11710238 - Instance crash due to ORA-600 [1433] for SMCO messages [ID 11710238.8]
- SMCO (Space Management Coordinator) For Autoextend On Datafiles And How To Disable/Enable [ID 743773.1]

In production we saw loads of library cache contention. All caused by this SMCO process.

To demonstrate the issue I dropped the hotsos_demo_lobs tablespace and recreated it with a very small size and a minimal extent size. The minimal extent size for an uniform extent is at least 5 blocks, so in an 8Kb tablespace this would be 40Kb. Unless you are using SecureFile lobs. Then the minimum number of blocks is 14. If you set a value of 5, which is valid upon tablespace creation time, you would end up with:

```
ORA-60019: Creating initial extent of size 5 in tablespace of extent size 14
```

14 extents equals 112K, which is something else than Bug 9477178 (ORA-60019: CREATING INITIAL EXTENT OF SIZE X IN TABLESPACE FOR SecureFile lobs) is suggesting. Now if you create the tablespace with a size of 14 times the blocksize, and you will run into this error when the first extent is added:

```
ERROR at line 1:
ORA-00600: internal error code, arguments: [ktsladdfcb-bsz], [3], [], [], [],
[], [], [], [], [], [], []
ORA-06512: at line 24
```

This ORA-00600 is not documented at MOS, but some people did write about it. Even though my italian isn't great, this guy did give me the best clue: http://cristiancudizio.wordpress.com/2009/09/22/test-con-11gr2-requisiti-per-i-securefile/ . Just add one byte to the value and you should be okay. So (14*8k)+1 = 114689

```
SQL> CREATE BIGFILE TABLESPACE hotsos_demo_lobs
  2    DATAFILE SIZE 100M AUTOEXTEND ON MAXSIZE unlimited
  3    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 114689
  4    SEGMENT SPACE MANAGEMENT AUTO
  5  /

Tablespace created.
```

The filestorage table is created with filesystem like logging for maximum performance while inserting:

```
CREATE TABLE filestorage
( did              NUMBER NOT NULL,
  document_name    VARCHAR2(1024) NOT NULL,
  bfiledata        BLOB,
  CONSTRAINT pk_filestorage PRIMARY KEY (did)
)
TABLESPACE hotsos_demos
LOB (bfiledata) STORE AS SECUREFILE
  (
    TABLESPACE hotsos_demo_lobs DISABLE STORAGE IN ROW NOCACHE
FILESYSTEM_LIKE_LOGGING
  )
/
```

Next up the full documentation of both Oracle Database 11.2 as well as the Fusion Middleware 11.1.1.6 documentation was loaded, some 113621 files in 7.1 GB . Data was loaded using the following procedure:

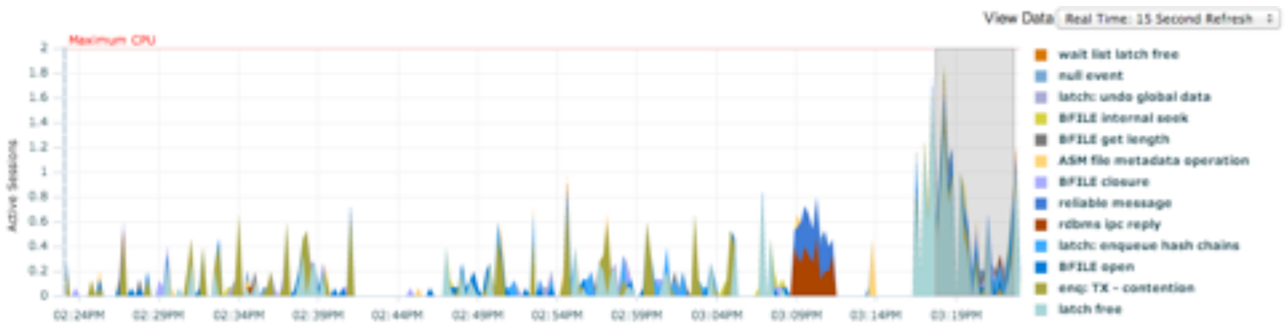
```
set serveroutput on

DECLARE
  lv_file       BFILE;
  lv_blob       BLOB;
  lv_start      NUMBER;
  lv_start_cpu  NUMBER;
  lv_directory  VARCHAR2(1024) := '/u01/files/docs';
  lv_ns         VARCHAR2(1024);
  e_21560       EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_21560,-21560);
BEGIN
  -- Search the files
  SYS.DBMS_BACKUP_RESTORE.SEARCHFILES(lv_directory, lv_ns);

  lv_start     := DBMS_UTILITY.GET_TIME;
  lv_start_cpu := DBMS_UTILITY.GET_CPU_TIME;

  FOR each_file IN (SELECT fname_krbmsft AS name
                    FROM sys.xkrbmsft) LOOP
    -- Open the file
    lv_file := BFILENAME('DOCS', substr(each_file.name, length(lv_directory) +
2) );
    DBMS_LOB.OPEN(lv_file, DBMS_LOB.LOB_READONLY);

    BEGIN
      INSERT INTO filestorage (did, document_name, bfiledata )
        VALUES ( seq_did.nextval, each_file.name, empty_blob() )
        RETURNING bfiledata INTO lv_blob;

      DBMS_LOB.OPEN(lv_blob, DBMS_LOB.LOB_READWRITE);
      DBMS_LOB.LOADFROMFILE(DEST_LOB => lv_blob,SRC_LOB => lv_file,AMOUNT   =>
DBMS_LOB.GETLENGTH(lv_file));
      DBMS_LOB.CLOSE(lv_blob);
    EXCEPTION
      WHEN e_21560 THEN
        DBMS_LOB.CLOSE(lv_blob);
        DBMS_OUTPUT.PUT_LINE('error inserting: ' || each_file.name);
    END;

    DBMS_LOB.CLOSE(lv_file);

  END LOOP;

  DBMS_OUTPUT.put_line('loaded in filestorage: ' || (DBMS_UTILITY.GET_TIME -
lv_start) || ' hsecs, cpu: ' || (DBMS_UTILITY.GET_CPU_TIME - lv_start_cpu) || '
hsecs');

END;
/
```

When this procedure was run in three (3) different sessions to create some load at the system. When I look at top -c when running the PL/SQL block I see this output:

```
top - 15:22:28 up 1 day, 20:08,  4 users,  load average: 5.90, 4.60, 3.53
Tasks: 221 total,   2 running, 219 sleeping,   0 stopped,   0 zombie
Cpu(s): 18.9%us, 47.7%sy,  0.0%ni, 11.4%id, 14.5%wa,  0.0%hi,  7.5%si,  0.0%st
Mem:   4055244k total,  3972860k used,    82384k free,    11612k buffers
Swap:  6094844k total,   624336k used,  5470508k free,  2472036k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 6892 oracle    20   0 2338m 269m 233m D 18.1  6.8   1:15.84 oracleDEMODB
(DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq)))
 6912 oracle    20   0 2338m 276m 240m D 17.8  7.0   1:14.68 oracleDEMODB
(DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq)))
 6893 oracle    20   0 2338m 265m 230m D 16.8  6.7   1:15.05 oracleDEMODB
(DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq)))
 1974 oracle    20   0 2302m 525m 516m S 14.5 13.3   4:02.84 ora_w000_DEMODB
 6037 oracle    20   0 2302m 219m 211m R 13.5  5.6   0:25.09 ora_w008_DEMODB
 6153 oracle    20   0 2302m 203m 195m S 13.2  5.1   0:19.27 ora_w009_DEMODB
15477 oracle    -2   0 2289m  15m  13m S 12.5  0.4  24:02.73 ora_vktm_DEMODB
 3251 oracle    -2   0 1234m  12m  10m S  8.6  0.3 164:59.08 asm_vktm_+ASM
 5939 oracle    20   0 2302m 246m 238m S  7.9  6.2   0:32.41 ora_w005_DEMODB
 2531 oracle    20   0 2302m 513m 505m S  6.9 13.0   3:40.88 ora_w006_DEMODB
15493 oracle    20   0 2312m  69m  65m D  5.6  1.7   2:36.89 ora_lgwr_DEMODB
 2343 oracle    20   0 2302m 521m 513m S  2.3 13.2   3:51.31 ora_w001_DEMODB
 3511 oracle    20   0 2302m 455m 447m S  2.0 11.5   2:56.63 ora_w007_DEMODB
15491 oracle    20   0 2301m 595m 583m S  2.0 15.0   1:57.67 ora_dbw0_DEMODB
 5936 oracle    20   0 2302m 229m 221m S  1.6  5.8   0:30.82 ora_w003_DEMODB
30181 oracle    20   0 2302m 527m 519m S  1.6 13.3   3:55.08 ora_w004_DEMODB
 7011 root      20   0     0    0    0 S  1.3  0.0   0:01.67 [kworker/0:3]
 2693 oracle    20   0 2302m 504m 495m S  1.0 12.7   3:16.53 ora_w002_DEMODB
```

As you can see the w00X processes are consuming a considerable amount of CPU. These W00X processes are all space management slaves of the SMCO process trying to extent the table and the datafile.

Also I am seeing considerable amounts of latch free waits:



These latch free waits should be originated in table and datafile extents. This suggests that pre-allocating space manually is “cheaper” and therefore best practice.
Apart from pre-allocation of space, the space management is also responsible for moving free chunks of space from uncommitted free space to committed free space. This sounds very useful for a WebCenter Content setup for Information Lifecycle Management (ILM) purposes. The WebCenter Content way of implementing ILM is Webcenter Records[25]. This paper is not for going into too much detail about WebCenter Records, but the high level working of retention is records is:

Every X time, usually night, a query is run over de metadata repository to select documents that fit certain criteria. These criteria can be time (documents older than), category, size or a combination of those. If a document, or series of documents, fits the criteria than this document is moved to an other storage provider in WebCenter Content. This can be a NAS or some other kind of cheap storage device. Technically this means that the SecureFile lob is deleted from the database.

If more criteria than just time are used to identify documents to be moved, this can have some serious implications for your database. Let me demonstrate that:

```
SQL> CREATE TABLE filestorage
  2  ( did                NUMBER NOT NULL,
  3    document_name      VARCHAR2(1024) NOT NULL,
  4    document_type      VARCHAR2(4),
  5    bfiledata          BLOB,
  6    CONSTRAINT pk_filestorage PRIMARY KEY (did)
  7  )
  8  TABLESPACE hotsos_demos
  9  LOB (bfiledata) STORE AS SECUREFILE
 10    (
 11      TABLESPACE hotsos_demo_lobs DISABLE STORAGE IN ROW NOCACHE
 12    )
 13  /

Table created.
```

25 http://www.oracle.com/technetwork/middleware/webcenter/content/index-100357.html

```
SQL> DECLARE
  2    lv_file       BFILE;
  3    lv_blob       BLOB;
  4    lv_start      NUMBER;
  5    lv_start_cpu  NUMBER;
  6    lv_directory  VARCHAR2(1024) := '/u01/files/docs';
  7    lv_ns         VARCHAR2(1024);
  8    e_21560       EXCEPTION;
  9    lv_type       filestorage.document_type%type;
 10    PRAGMA EXCEPTION_INIT(e_21560,-21560);
 11  BEGIN
 12    -- Search the files
 13    SYS.DBMS_BACKUP_RESTORE.SEARCHFILES(lv_directory, lv_ns);
 14
 15    lv_start     := DBMS_UTILITY.GET_TIME;
 16    lv_start_cpu := DBMS_UTILITY.GET_CPU_TIME;
 17
 18    FOR each_file IN (SELECT fname_krbmsft AS name
 19                        FROM sys.xkrbmsft) LOOP
 20      -- Open the file
 21      lv_file := BFILENAME('DOCS', substr(each_file.name,length(lv_directory)
+ 2) );
 22      DBMS_LOB.OPEN(lv_file, DBMS_LOB.LOB_READONLY);
 23
 24      BEGIN
 25        IF LENGTH(SUBSTR(each_file.name, INSTR(each_file.name,'.',-1) +1 )) >
4 THEN
 26          lv_type := NULL;
 27        ELSE
 28          lv_type := SUBSTR(each_file.name, INSTR(each_file.name,'.',-1)
+1 );
 29        END IF;
 30        INSERT INTO filestorage (did, document_name, document_type,
bfiledata )
 31          VALUES ( seq_did.nextval, each_file.name, lv_type, empty_blob() )
 32          RETURNING bfiledata INTO lv_blob;
 33
 34        DBMS_LOB.OPEN(lv_blob, DBMS_LOB.LOB_READWRITE);
 35        DBMS_LOB.LOADFROMFILE(DEST_LOB => lv_blob,SRC_LOB => lv_file,AMOUNT
=> DBMS_LOB.GETLENGTH(lv_file));
 36        DBMS_LOB.CLOSE(lv_blob);
 37      EXCEPTION
 38        WHEN e_21560 THEN
 39          DBMS_LOB.CLOSE(lv_blob);
 40          DBMS_OUTPUT.PUT_LINE('error inserting: ' || each_file.name);
 41      END;
 42
 43      DBMS_LOB.CLOSE(lv_file);
 44
 45    END LOOP;
 46
 47    DBMS_OUTPUT.put_line('loaded in: ' || (DBMS_UTILITY.GET_TIME - lv_start)
|| ' hsecs, cpu time:' || (DBMS_UTILITY.GET_TIME - lv_start_cpu) || ' hsecs');
 48
 49  END;
 50  /
error inserting: /u01/files/docs/E11882_01/dcommon/css/darbbook.css
loaded in: 16774 hsecs, cpu time:430778563 hsecs

PL/SQL procedure successfully completed.


SQL> exec dbms_stats.gather_table_stats(ownname => 'DEMO',tabname =>
'FILESTORAGE', cascade => TRUE, estimate_percent => NULL);
```

```
PL/SQL procedure successfully completed.

SQL> SELECT num_rows, blocks
  2    FROM user_tables
  3   WHERE table_name = 'FILESTORAGE'
  4  /

  NUM_ROWS     BLOCKS
---------- ----------
     24102        496

1 row selected.

SQL> column column_name format a30
SQL> SELECT ulb.table_name, ulb.column_name, ulb.deduplication, sum(ust.bytes)/
1024/1024 as size_in_mb
  2    FROM user_lobs     ulb,
  3         user_segments ust
  4   WHERE ust.segment_name = ulb.segment_name
  5   GROUP BY ulb.table_name, ulb.column_name, ulb.deduplication
  6  /

TABLE_NAME             COLUMN_NAME                    DEDUPLICATION   SIZE_IN_MB
---------------------- ------------------------------ --------------- ----------
FILESTORAGE            BFILEDATA                      LOB               2371.125

1 row selected.

SQL> SELECT document_type, count(*)
  2    FROM filestorage
  3   GROUP BY document_type
  4* /

DOCU   COUNT(*)
---- ----------
db          219
             14
jpg          37
dtd           1
xml           1
htm       14875
js          223
epub        219
html       2027
png         466
pdf         216
mobi        219
gif        5559
css          25
zip           1

15 rows selected.

SQL> DELETE FROM filestorage
  2   WHERE document_type = 'gif';

5559 rows deleted.

SQL> COMMIT;

Commit complete.
```

```
exec dbms_stats.gather_table_stats(ownname => 'DEMO',tabname => 'FILESTORAGE',
cascade => TRUE, estimate_percent => NULL);

SELECT num_rows, blocks
  FROM user_tables
 WHERE table_name = 'FILESTORAGE'
/

  NUM_ROWS     BLOCKS
---------- ----------
     18543        496

1 row selected.

column column_name format a30
SQL>
SELECT ulb.table_name, ulb.column_name, ulb.deduplication, sum(ust.bytes)/
1024/1024 as size_in_mb
  FROM user_lobs     ulb,
       user_segments ust
 WHERE ust.segment_name = ulb.segment_name
 GROUP BY ulb.table_name, ulb.column_name, ulb.deduplication
  6  /

TABLE_NAME             COLUMN_NAME                    DEDUPLICATION   SIZE_IN_MB
---------------------- ------------------------------ --------------- ----------
FILESTORAGE            BFILEDATA                      LOB               2371.125

1 row selected.
```

So, deleting rows and immediately calculating size doesn't change anything. Data is still allocated, no free space is given back. Now if I would load the same documents again, you would expect to see a size less than double of the current size (less than ~4680MB), because the removed .gif files would fit perfect in the free space.

```
DECLARE
  lv_file       BFILE;
  lv_blob       BLOB;
  lv_start      NUMBER;
  lv_start_cpu  NUMBER;
  lv_directory  VARCHAR2(1024) := '/u01/files/docs';
  lv_ns         VARCHAR2(1024);
  e_21560       EXCEPTION;
  lv_type       filestorage.document_type%type;
  PRAGMA EXCEPTION_INIT(e_21560,-21560);
BEGIN
  -- Search the files
  SYS.DBMS_BACKUP_RESTORE.SEARCHFILES(lv_directory, lv_ns);

  lv_start     := DBMS_UTILITY.GET_TIME;
  lv_start_cpu := DBMS_UTILITY.GET_CPU_TIME;

  FOR each_file IN (SELECT fname_krbmsft AS name
                      FROM sys.xkrbmsft) LOOP
    -- Open the file
    lv_file := BFILENAME('DOCS', substr(each_file.name,length(lv_directory) +
2) );
    DBMS_LOB.OPEN(lv_file, DBMS_LOB.LOB_READONLY);

    BEGIN
      IF LENGTH(SUBSTR(each_file.name, INSTR(each_file.name,'.',-1) +1 )) > 4
THEN
        lv_type := NULL;
      ELSE
        lv_type := SUBSTR(each_file.name, INSTR(each_file.name,'.',-1) +1 );
      END IF;
      INSERT INTO filestorage (did, document_name, document_type, bfiledata )
        VALUES ( seq_did.nextval, each_file.name, lv_type, empty_blob() )
        RETURNING bfiledata INTO lv_blob;

      DBMS_LOB.OPEN(lv_blob, DBMS_LOB.LOB_READWRITE);
      DBMS_LOB.LOADFROMFILE(DEST_LOB => lv_blob,SRC_LOB => lv_file,AMOUNT   =>
DBMS_LOB.GETLENGTH(lv_file));
      DBMS_LOB.CLOSE(lv_blob);
    EXCEPTION
      WHEN e_21560 THEN
        DBMS_LOB.CLOSE(lv_blob);
        DBMS_OUTPUT.PUT_LINE('error inserting: ' || each_file.name);
    END;

    DBMS_LOB.CLOSE(lv_file);

  END LOOP;

  DBMS_OUTPUT.put_line('loaded in: ' || (DBMS_UTILITY.GET_TIME - lv_start) || '
hsecs, cpu time:' || (DBMS_UTILITY.GET_TIME - lv_start_cpu) || ' hsecs');

END;
/

exec dbms_stats.gather_table_stats(ownname => 'DEMO',tabname => 'FILESTORAGE',
cascade => TRUE, estimate_percent => NULL);

SELECT ulb.table_name, ulb.column_name, ulb.deduplication,
       SUM(ust.bytes)/1024/1024 as size_in_mb
  FROM user_lobs     ulb,
       user_segments ust
 WHERE ust.segment_name = ulb.segment_name
 GROUP BY ulb.table_name, ulb.column_name, ulb.deduplication
/

TABLE_NAME                 COLUMN_NAME                DEDUPLICATION   SIZE_IN_MB
-------------------------- -------------------------- --------------- ----------
FILESTORAGE                BFILEDATA                  NO                4607.125

1 row selected.
```

This suggests that the freespace is not reused for new SecureFile lobs, the uncommited free space stays as it is. THIS PART NEEDS MORE TESTING

Filesystem like logging

For testing of the filesystem like logging feature two filestorage tables were created:

```
CREATE TABLE filestorage_l
( did              NUMBER NOT NULL,
  bfiledata        BLOB,
  CONSTRAINT pk_filestorage_l PRIMARY KEY (did)
)
TABLESPACE hotsos_demos
LOB (bfiledata) STORE AS SECUREFILE
  (
    TABLESPACE hotsos_demo_lobs DISABLE STORAGE IN ROW NOCACHE LOGGING
  )
/

CREATE TABLE filestorage_fl
( did              NUMBER NOT NULL,
  bfiledata        BLOB,
  CONSTRAINT pk_filestorage_fl PRIMARY KEY (did)
)
TABLESPACE hotsos_demos
LOB (bfiledata) STORE AS SECUREFILE
  (
    TABLESPACE hotsos_demo_lobs DISABLE STORAGE IN ROW NOCACHE
FILESYSTEM_LIKE_LOGGING
  )
/
```

As you can see there are two differences between the two tables:

1. The table name
2. The logging type: `logging` versus `filesystem_like_logging`

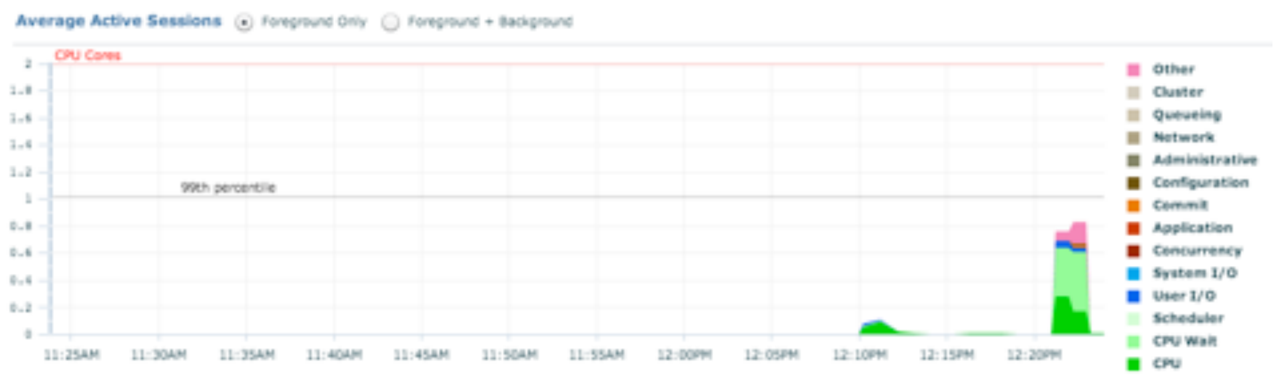
To test any performance differences between the two logging modes a block of PL/SQL code is used. This block inserts 1000 records into both tables and measures the time spend while inserting.

```
set serveroutput on

DECLARE
  lv_file       BFILE;
  lv_blob       BLOB;
  lv_start      NUMBER;
  lv_iterations PLS_INTEGER := 1000;
BEGIN
  -- Open the file
  lv_file := BFILENAME('FILES', 'largest.pdf');
  DBMS_LOB.OPEN(lv_file, DBMS_LOB.LOB_READONLY);

  lv_start := DBMS_UTILITY.GET_TIME;
  FOR i IN 1..lv_iterations LOOP

    -- logging insert
    INSERT INTO filestorage_l (did, bfiledata )
      VALUES ( seq_did.nextval, empty_blob() )
      RETURNING bfiledata INTO lv_blob;

    DBMS_LOB.OPEN(lv_blob, DBMS_LOB.LOB_READWRITE);
    DBMS_LOB.LOADFROMFILE(DEST_LOB => lv_blob,
                          SRC_LOB  => lv_file,
                          AMOUNT   => DBMS_LOB.GETLENGTH(lv_file));
    DBMS_LOB.CLOSE(lv_blob);

  END LOOP;
  DBMS_OUTPUT.put_line('logging: ' || (DBMS_UTILITY.GET_TIME - lv_start) ||
                       ' hsecs');

  lv_start := DBMS_UTILITY.GET_TIME;
  FOR i IN 1..lv_iterations LOOP

    -- filesystem_like_logging insert
    INSERT INTO filestorage_fl (did, bfiledata )
      VALUES ( seq_did.nextval, empty_blob() )
      RETURNING bfiledata INTO lv_blob;

    DBMS_LOB.OPEN(lv_blob, DBMS_LOB.LOB_READWRITE);
    DBMS_LOB.LOADFROMFILE(DEST_LOB => lv_blob,
                          SRC_LOB  => lv_file,
                          AMOUNT   => DBMS_LOB.GETLENGTH(lv_file));
    DBMS_LOB.CLOSE(lv_blob);

  END LOOP;
  DBMS_OUTPUT.put_line('filesystem_like_logging: ' ||
                        (DBMS_UTILITY.GET_TIME - lv_start) || ' hsecs');
  DBMS_LOB.CLOSE(lv_file);

END;
/
```

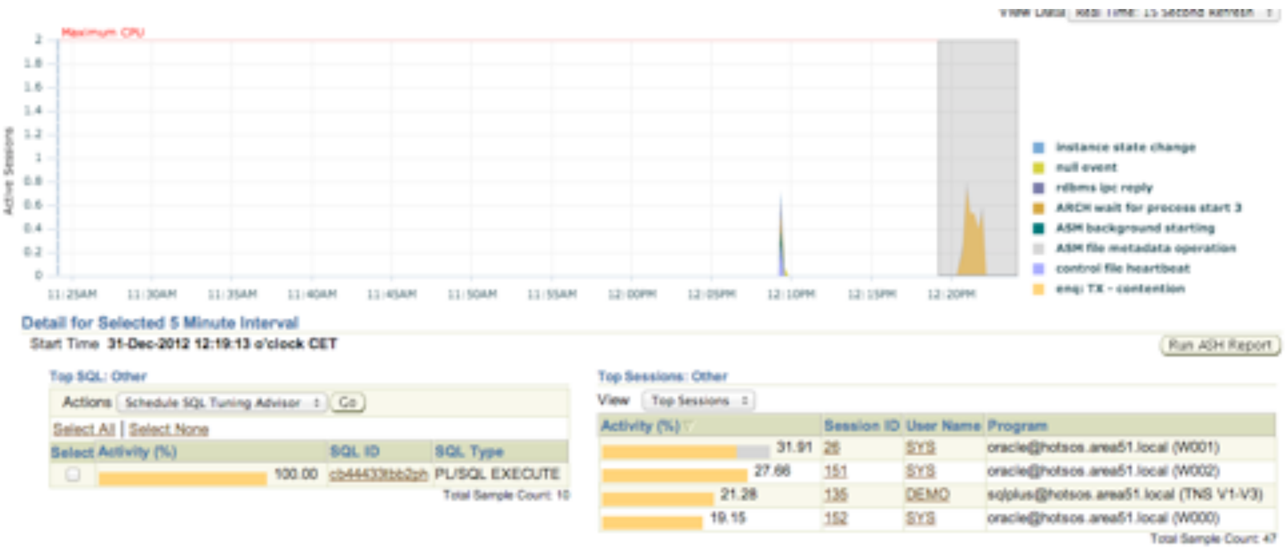
When I ran the procedure the results were these:

```
logging: 5679 hsecs
filesystem_like_logging: 2849 hsecs
```

This suggests that filesystem_like_logging takes about half the time to finish loading 1000 documents into a table. However, when I look at the performance page in the dbconsole I notice the following:



Another waitclass appears taking considerable amount of time, in waitclass other:



Detail for Selected 5 Minute Interval

Start Time 31-Dec-2012 12:19:13 o'clock CET

Top SQL: Other

Select	Activity (%)	SQL ID	SQL Type
	100.00	cb44433tbb2ph	PL/SQL EXECUTE

Total Sample Count: 10

Top Sessions: Other

Activity (%)	Session ID	User Name	Program
31.91	26	SYS	oracle@hotsos.area51.local (W001)
27.66	151	SYS	oracle@hotsos.area51.local (W002)
21.28	135	DEMO	sqlplus@hotsos.area51.local (TNS V1-V3)
19.15	152	SYS	oracle@hotsos.area51.local (W000)

Total Sample Count: 47

This `enq: TX - contention` event seems to be a catch all for TX locks that are not part of some other TX event. One of the typical causes for this wait event is resizing of a datafile (SMCO). Initially the hotsos_demo_lobs datafile was created with a 100MB size. When querying dba_data_files it shows that the new size is 2100M:

```
SQL> SELECT bytes/1024/1024
  2    FROM dba_data_files
  3   WHERE tablespace_name = 'HOTSOS_DEMO_LOBS';

BYTES/1024/1024
---------------
           2100

1 row selected.
```

To remove time spend on extending the datafile from the process time the hotsos_demo_lobs tablespace is enlarged to 2200M[26] before the test is repeated.

Just preallocating the datafile isn't enough to remove all the waits on enq: TX - contention though. Preallocating extents for the table and lobsegment is needed too to eliminate all allocation time spend. Therefore the setup was recreated again, the tablespace was enlarged to 2400MB and the bfiledata lobs were preallocated.

```
ALTER TABLE filestorage_l MODIFY LOB (bfiledata) (allocate extent (size 1200M) )
/
```

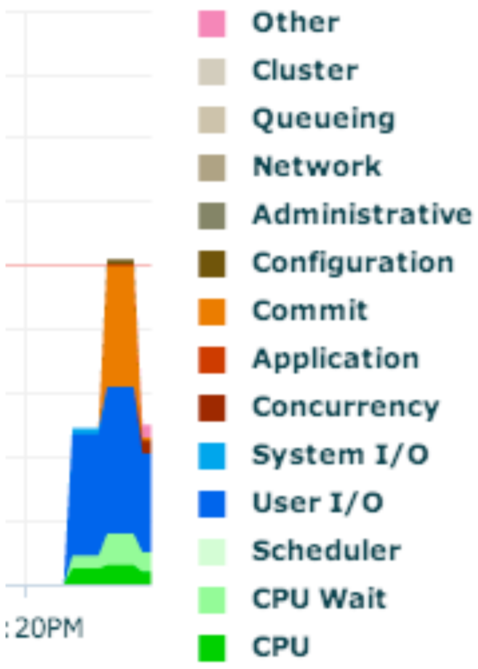
When you run this statement on a table without any rows, you will run into this error:
`ORA-14220: Invalid operation for object without a segment`

Allocating extents to the table first will allow you to create extents for the lobsegments, then you will not run into this ORA-14220 error. By extending the table and datafile before loading you also remove the proactive space pre-allocation by the SMCO background process. When the test was rerun again the results were these:

```
logging: 3763 hsecs
filesystem_like_logging: 2433 hsecs
```

26 alter tablespace hotsos_demo_lobs resize 2200M;

The dbconsole showed a completely different image again:



The wait information suggests that log file sync is taking too much time. Tuning that all over again is beyond scope for this test. Comparing the results from these tests the following can be said about filesystem like logging:

- filesystem_like_logging is faster than logging
- The better the database is managed, the smaller the performance differences are.

This performance gain by using filesystem_like_logging has to come at some cost. Since the biggest performance hit is caused by log file sync it would suggest that less data is stored in the logs when using filesystem_like_logging. This is in line with documentation[27]:

> "FILESYSTEM_LIKE_LOGGING means that the system only logs the metadata. This option is invalid for BasicFiles LOBs. This setting is similar to metadata journaling of file systems, which reduces mean time to recovery from failures. The LOGGING setting for SecureFile lobs LOBs is similar to the data journaling of file systems. Both the LOGGING and FILESYSTEM_LIKE_LOGGING settings provide a complete transactional file system with SecureFile lobs LOBs.
>
> For SecureFile lobs LOBs, the NOLOGGING setting is converted internally to FILESYSTEM_LIKE_LOGGING.
>
> FILESYSTEM_LIKE_LOGGING ensures that data is completely recoverable after a server failure"

Now especially this last two sentence are interesting. First of all nologging is basically a non existing feature for SecureFile lobs. This can be surprising when running a batch run, especially when you don't expect any redo at all. Secondly documentation says that data is completely recoverable after a sever failure.

27 http://docs.oracle.com/cd/E11882_01/appdev.112/e18294/adlob_smart.htm#ADLOB45951

If you would google for “server failure oracle” you end up at Fusion Middleware documentation, and thats highly unlikely what is meant in the SecureFile lob documentation. I *think* documentation points towards instance crash, but I don’t know for sure. To survive instance crash all relevant information would have to be in the redo log.

To find out what is written in de redo log exactly, you would have to dump the actual logfile.

To create a clean testcase I modified the loading procedure slightly, so it only loads one document into each table (filestore_l and filestore_fl). Also some logswitches were performed to create “clean” redo logs for dumping.

```
DECLARE
  lv_file       BFILE;
  lv_blob       BLOB;
BEGIN
  -- Open the file
  lv_file := BFILENAME('FILES', 'largest.pdf');
  DBMS_LOB.OPEN(lv_file, DBMS_LOB.LOB_READONLY);

  EXECUTE IMMEDIATE 'ALTER SYSTEM SWITCH LOGFILE';
  -- logging
  INSERT INTO filestorage_l (did, bfiledata )
    VALUES ( seq_did.nextval, empty_blob() )
    RETURNING bfiledata INTO lv_blob;

  DBMS_LOB.OPEN(lv_blob, DBMS_LOB.LOB_READWRITE);
  DBMS_LOB.LOADFROMFILE(DEST_LOB => lv_blob,
                        SRC_LOB  => lv_file,
                        AMOUNT   => DBMS_LOB.GETLENGTH(lv_file));
  DBMS_LOB.CLOSE(lv_blob);

  EXECUTE IMMEDIATE 'ALTER SYSTEM SWITCH LOGFILE';

  -- filesystem_like_logging
  INSERT INTO filestorage_fl (did, bfiledata )
    VALUES ( seq_did.nextval, empty_blob() )
    RETURNING bfiledata INTO lv_blob;

  DBMS_LOB.OPEN(lv_blob, DBMS_LOB.LOB_READWRITE);
  DBMS_LOB.LOADFROMFILE(DEST_LOB => lv_blob,
                        SRC_LOB  => lv_file,
                        AMOUNT   => DBMS_LOB.GETLENGTH(lv_file));
  DBMS_LOB.CLOSE(lv_blob);

  EXECUTE IMMEDIATE 'ALTER SYSTEM ARCHIVE LOG CURRENT';

END;
/
```

Next the logfiles themselves can be dumped. The logfiles to be dump can be found by querying v $archived_log (output slightly modified for layout purposes):

```
SQL> SELECT to_char(first_time, 'DD-MON-RRRR HH24:MI:SS') as first_time ,
  2         blocks, name
  4    FROM v$archived_log
  5*  WHERE trunc(first_time) = trunc(sysdate)
SQL> /

FIRST_TIME                 BLOCKS NAME
---------------------- ---------- ----------------------------------------
05-JAN-2013 21:52:20            3 +DATA/demodb/archivelog/2013_01_05/
                                  thread_1_seq_714.311.803944341
05-JAN-2013 21:52:20           17 +DATA/demodb/archivelog/2013_01_05/
                                  thread_1_seq_715.312.803944343
05-JAN-2013 21:52:22         2018 +DATA/demodb/archivelog/2013_01_05/
                                  thread_1_seq_716.313.803944347
05-JAN-2013 21:52:26            4 +DATA/demodb/archivelog/2013_01_05/
                                  thread_1_seq_717.314.803944349
05-JAN-2013 21:52:28            5 +DATA/demodb/archivelog/2013_01_05/
                                  thread_1_seq_718.363.803944353
05-JAN-2013 21:52:31           93 +DATA/demodb/archivelog/2013_01_05/
                                  thread_1_seq_719.359.803944355
05-JAN-2013 21:52:35            4 +DATA/demodb/archivelog/2013_01_05/
                                  thread_1_seq_720.394.803944357
```

It seems that `+DATA/demodb/archivelog/2013_01_05/thread_1_seq_716.313.803944347` is the logfile with the logging filestorage and `+DATA/demodb/archivelog/2013_01_05/thread_1_seq_719.359.803944355` is the redo log with the filesystem_like_logging filestorage data. These logfiles can be dump by issuing the following command:

```
SQL> ALTER SYSTEM DUMP LOGFILE '+DATA/demodb/archivelog/2013_01_05/
thread_1_seq_719.359.803944355';

System altered.
```

In the redo.log file that contains the logging SecureFile the data of the lob is shown at the KDLI data keyword. What is interesting though, is that the SecureFile with filesystem_like_logging contains less redo records. Also the layout of the records is different, some data is included in the filesystem_like_logging record that is not included in the logging record.

In the redo records you can find KDLI info that shows the actual lobid. This is the reference to the metadata that is mentioned in documentation “FILESYSTEM_LIKE_LOGGING means that the system only logs the metadata”.

```
REDO RECORD - Thread:1 RBA: 0x0002cf.00000057.0168 LEN: 0x0058 VLD: 0x01
SCN: 0x0000.00392d4e SUBSCN:  1 01/05/2013 21:52:32
CHANGE #1 INVLDX AFN:6 DBA:0x00000891 BLKS:0x006c OBJ:70156 SCN:0x0000.00392d48
SEQ:1 OP:26.4 ENC:0
KDLI common [12]
  op    0x04 [INVL]
  type  0x20 [data]
  flg0  0x02
  flg1  0x00
  psiz  8060
  poff  128
  dba   0x00000891
KDLI info [1.17]
  lobid 0000000100000024022c
  block 0x00000000
  slot  0x0000
```

Tanel Põder stated in his presentation about lobs[28] that the lobid can be found by dumping the segments header block. To be able to dump the header block, you first have to identify the actual block:

```
SQL> SELECT dst.header_file, dst.header_block
  2    FROM dba_lobs     dlb,
  3         dba_segments dst
  4   WHERE dlb.table_name = 'FILESTORAGE_FL'
  5     AND dlb.column_name = 'BFILEDATA'
  6*    AND dst.segment_name = dlb.segment_name
SQL> /

HEADER_FILE HEADER_BLOCK
----------- ------------
          6         1577

1 row selected.
```

Next up we can dump the block

```
SQL> ALTER SYSTEM DUMP DATAFILE 6 BLOCK 1577;

System altered.
```

Looking at the header I was expecting to find the lobid, but instead I find that the type is not LOB BLOCK and there is no lobid in the header:

```
Block dump from disk:
buffer tsn: 6 rdba: 0x00000629 (1024/1577)
scn: 0x0000.00392d50 seq: 0x03 flg: 0x04 tail: 0x2d503f03
frmt: 0x02 chkval: 0xda12 type: 0x3f=NGLOB: Segment Header
Hex dump of block: st=0, typ_found=1
Dump of memory from 0x00007F441169DA00 to 0x00007F441169FA00
7F441169DA00 0000A23F 00000629 00392D50 04030000  [?...)...P-9.....]
7F441169DA10 0000DA12 00000000 00000000 00000000  [................]
7F441169DA20 00000000 00000003 00000110 00001FE8  [................]
7F441169DA30 00000002 00000080 00000080 00000980  [................]
7F441169DA40 00000000 00000002 00000000 00000110  [................]
...
```

28 http://www.slideshare.net/tanelp/oracle-lob-internals-and-performance-tuning

This shows that Oracle changed internals at block level with SecureFile lobs. Oracle introduced six new block types for SecureFile lobs. Al these types start with NGLOB:

1. NGLOB: Lob Extent Header
   - First block of every SecureFile lob extent
2. NGLOB: Segment Header
   - Second block of the first extent
   - Highwater Mark, Extent Map
   - Administration of Hash Bucket Blocks
3. NGLOB: Extent Map
4. NGLOB: Committed Free Space
5. NGLOB: Persistent Undo
6. NGLOB: Hash Buckets – variable chunk-size
   - 7 Buckets for chunks of different sizes: 2k to 32K, 32k to 64k, 64k to 128k, 128k to 256k, 256k to 512k, 512k to 1m, 1m to 64m

These blocks can be viewed by dumping the segment header

```
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_DUMP('HOTSOS_DEMO_LOBS', 6, 1577);
```

Example of a Lob Segment Header:

```
NGLOB Segment Header
  Extent Control Header
  -----------------------------------------------------------------
  Extent Header:: spare1: 0      spare2: 0      #extents: 3      #blocks: 272
                  last map  0x00000000  #maps: 0      offset: 8168
      Highwater::  0x00000980  ext#: 2      blk#: 128    ext size: 128
  #blocks in seg. hdr's freelists: 0
  #blocks below: 272
  mapblk  0x00000000  offset: 2
                   Unlocked
     Map Header:: next  0x00000000  #extents: 3    obj#: 70156  flag: 0x40000000
  Extent Map
  -----------------------------------------------------------------
   0x00000628  length: 16
   0x00000880  length: 128
   0x00000900  length: 128

  ---------------
CFS hbbcnt:7 hbbmx:7 Mxrng:7 UFS hbbcnt:2 PUA cnt:1 Flag:2
Segment Stats
-------------
Retention: -1
Free Space: 261
PUA Batchsize: 12
UFS Array
---------
DBA: 0x0000062a Inst:1
DBA: 0x0000062b Inst:1
Ufs off:152 sz:512
CFS Array
---------
Range: 0 DBA: 0x0000062c
Range: 1 DBA: 0x0000062d
Range: 2 DBA: 0x0000062e
Range: 3 DBA: 0x0000062f
Range: 4 DBA: 0x00000630
Range: 5 DBA: 0x00000631
Range: 6 DBA: 0x00000632
```

```
Cfs off:664 sz:576
PUA Array
---------
DBA: 0x00000885 Inst:1
pua off:1240 sz:8
```

Example of a Hash Bucket:

```
 Range: 0 Hash bucket block DBA: 0x0000062c
---------------------------------
  Dump of Hash Bucket Block
  --------------------------------------------------------
Hash Bucket Header Dump
Range: 2k to 32k
Inst:1 Flag:5 Total:0 FSG_COUNT:0 OBJD:70156 Inc:0
Self Descriptive
 fsg_count:0
Head:0x 0x00000000  Tail:0x 0x00000000
 Opcdode:0 Locker Xid: 0x0000.000.00000000
Fsbdba: 0x0    Fbrdba: 0x0
Head Cache Entries
-------------------
-------------------
Tail Cache Entries
-------------------
-------------------
Free Space Chunk Summary
Inst:1 Minc:0 Maxc:0 Count:0 Tot:0 MAXC:0
CFS Chunk List
--------------
```

Now I'm no big expert at reading blocks, nor have I ever been in the position where I really needed to look at the blocks (as far as I know). Other people have much more expertise with that. I found two sources[29] that seemed reliable and that helped me with this little information about dumping the blocks. Besides the lack of knowledge I also doubt the usefulness of knowledge like this for my day to day job, this is too much internal and beyond my usual scope for me to dig into further into this block business.

Not finding the lobid did eat away at me though (I actually lost sleep over this one). By some coincidence I ran into dbms_lobutil[30] in Julian Dyke's presentation. This package is not documented with the 11R2 database documentation.

The file that contains the package is ${ORACLE_HOME}/rdbms/admin/dbmslobu.sql and it contains a very limited description:

> "The new package DBMS_LOBUTIL is a container for diagnostic and utility functions and procedures specific to 11globs.
>
> Since diagnostic operations are not part of the standard programmatic APIs in DBMS_LOB, they are provided in a separate namespace to avoid clutter. The diagnostic API is also not quite as critical to document for end-users; its main use is for internal developer, QA, and DDR use (especially since it peeks into the internal structure of 11glob inodes and lobmaps)."

29 http://www.database-consult.de/docs/LOBversusSF1.pdf & http://www.juliandyke.com/Presentations/LOBInternals.ppt

30 http://psoug.org/reference/dbms_lobutil.html

With this package it is possible to retrieve the lobid from the actual lob:

```
DECLARE
  lv_blob   BLOB;
  lv_inode  DBMS_LOBUTIL_INODE_T;
BEGIN

  -- we know there is only 1 record in this table, so no filter needed
  SELECT bfiledata
    INTO lv_blob
    FROM demo.filestorage_fl;

  lv_inode := DBMS_LOBUTIL.GETINODE (lv_blob);
  DBMS_OUTPUT.PUT_LINE ('LOBID: '||RAWTOHEX (lv_inode.lobid));

END;
/

LOBID: 0000000100000024022C
```

This lobid is consistent with what we have seen in the redo logfile. That still doesn't show me where the lobid is actually stored in a block, but it does give me some way to compare output of redo logs to what I find in the database.

Most of the database I have worked on with a physical stand-by database run in force logging mode. Just to make sure if this imposes a problem I want to check what happens in force logging is enabled.

```
SQL> ALTER DATABASE FORCE LOGGING
  2  /

Database altered
```

You can see that the logfiles are more or less the same size:

```
SQL> SELECT to_char(first_time, 'DD-MON-RRRR HH24:MI:SS') as first_time ,
  2         blocks, name
  3    FROM v$archived_log
  4*  WHERE trunc(first_time) = trunc(sysdate)
SQL> /

FIRST_TIME                BLOCKS NAME
--------------------- ---------- ------------------------------------------------
02-JAN-2013 10:14:41        3605 +DATA/demodb/archivelog/2013_01_02/
                                 thread_1_seq_674.372.803644173
02-JAN-2013 10:29:33           3 +DATA/demodb/archivelog/2013_01_02/
                                 thread_1_seq_675.368.803644173
02-JAN-2013 10:29:33          22 +DATA/demodb/archivelog/2013_01_02/
                                 thread_1_seq_676.337.803644175
02-JAN-2013 10:29:34        2088 +DATA/demodb/archivelog/2013_01_02/
                                 thread_1_seq_677.336.803644175
02-JAN-2013 10:29:35           3 +DATA/demodb/archivelog/2013_01_02/
                                 thread_1_seq_678.335.803644177
02-JAN-2013 10:29:36           6 +DATA/demodb/archivelog/2013_01_02/
                                 thread_1_seq_679.334.803644179
02-JAN-2013 10:29:38        2005 +DATA/demodb/archivelog/2013_01_02/
                                 thread_1_seq_680.333.803644179
02-JAN-2013 10:29:38           5 +DATA/demodb/archivelog/2013_01_02/
                                 thread_1_seq_681.332.803644181

8 rows selected.
```

So if the database is in force logging mode, the filesystem_like_logging option is ignored (as expected).

Final verdict on the filesystem type logging really depends on the features needed. For the project that started the investigation into SecureFile lobs a Recovery Point Objective (RPO) of zero is defined. Not having the lob data in the redo logfiles would be unacceptable. The system also needs a physical stand-by to be able to meet Recovery Time Objectives (RTO) defined in the SLA, I so no redo logs would be out of the question.

If you can restart loading or don't care about redo , for instance while running a batch operation, the filesystem_like_logging option is great. You can always reconfigure the lob to resume logging with an alter table statement after your batch operation is finished.

Compression

When investigating compression for SecureFile lobs the following questions arose:

- How much space will we actually save when using compression?
- Compression costs CPU for compressing and decompressing blocks when reading them. How much CPU will we waste on compression?
- How does table compression relate to SecureFile lob compression?

To investigate this compression feature the production dataset was analyzed and we agreed upon the following dataset:

Type	Name	Size
RTF	small.rtf	31472
RTF	medium.rtf	83762
RTF	large.rtf	232138
PDF/X	small.pdf	20526
PDF/X	medium.pdf	278333
PDF/X	large.pdf	470061
PDF/X	largest.pdf	928778
PDF/A	small_pdfa.pdf	26219
PDF/A	large_pdfa.pdf	442573
DOC	small.doc	22016
DOC	medium.doc	235520
DOC	large.doc	399872

To represent the dataset in the database, each file will be loaded a different number of times. With this mixture a fairly accurate guestimate of the actual compression rate can be made.

	PDF/A	PDF/X	DOC	RTF	Cumulative	Relative
small	500	100	150	100	850	39,53%
medium	0	100	150	100	350	16,28%
large	600	100	150	100	950	44,19%
Cumulative	1100	300	450	300	2150	
Relative	51,16%	13,95%	20,93%	13,95%		

This translates to SQL:

```
CREATE TABLE testdata_configuration
(
  document_type VARCHAR2(6) NOT NULL,
  document_name VARCHAR2(20) NOT NULL,
  num_loads     VARCHAR2(20) NOT NULL,
  CONSTRAINT testdata_configuration_pk PRIMARY KEY (document_type,
document_name)
)
/

INSERT INTO testdata_configuration (document_type, document_name, num_loads)
VALUES ('PDF/A','small_pdfa.pdf','500');
INSERT INTO testdata_configuration (document_type, document_name, num_loads)
VALUES ('PDF/A','large_pdfa.pdf','600');
INSERT INTO testdata_configuration (document_type, document_name, num_loads)
VALUES ('PDF/X','large.pdf','100');
INSERT INTO testdata_configuration (document_type, document_name, num_loads)
VALUES ('PDF/X','small.pdf','100');
INSERT INTO testdata_configuration (document_type, document_name, num_loads)
VALUES ('PDF/X','medium.pdf','100');
INSERT INTO testdata_configuration (document_type, document_name, num_loads)
VALUES ('DOC','small.doc','150');
INSERT INTO testdata_configuration (document_type, document_name, num_loads)
VALUES ('DOC','medium.doc','150');
INSERT INTO testdata_configuration (document_type, document_name, num_loads)
VALUES ('DOC','large.doc','150');
INSERT INTO testdata_configuration (document_type, document_name, num_loads)
VALUES ('RTF','small.rtf','100');
INSERT INTO testdata_configuration (document_type, document_name, num_loads)
VALUES ('RTF','medium.rtf','100');
INSERT INTO testdata_configuration (document_type, document_name, num_loads)
VALUES ('RTF','large.rtf','100');


COMMIT;
```

The filestorage table storing the documents in a SecureFile was modified slightly for easy reporting on compression rate:

```
CREATE TABLE filestorage
( did              NUMBER NOT NULL,
  compression_type VARCHAR2(6) NOT NULL,
  document_type    VARCHAR2(5) NOT NULL,
  document_name    VARCHAR2(14) NOT NULL,
  bfiledata        BLOB,
  CONSTRAINT pk_filestorage PRIMARY KEY (did),
  CONSTRAINT ck_fse_compression CHECK (
    compression_type IN ('NONE','MEDIUM','HIGH')
  ),
  CONSTRAINT ck_fse_document CHECK (
    document_type IN ('PDF/A','PDF/X','DOC','RTF')
  )
)
TABLESPACE hotsos_demos
LOB (bfiledata) STORE AS SECUREFILE
  (
    TABLESPACE hotsos_demo_lobs DISABLE STORAGE IN ROW NOCACHE
  )
PARTITION BY LIST (compression_type)
  SUBPARTITION BY LIST (document_type) SUBPARTITION TEMPLATE
    ( SUBPARTITION p_pdfa  VALUES ('PDF/A'),
      SUBPARTITION p_pdfx  VALUES ('PDF/X'),
      SUBPARTITION p_doc   VALUES ('DOC'),
      SUBPARTITION p_rtf   VALUES ('RTF')
    )
( PARTITION p_none   VALUES ('NONE')
    LOB (bfiledata) STORE AS SECUREFILE ( NOCOMPRESS ) ,
  PARTITION p_medium VALUES ('MEDIUM')
    LOB (bfiledata) STORE AS SECUREFILE ( COMPRESS MEDIUM) ,
  PARTITION p_high   VALUES ('HIGH')
    LOB (bfiledata) STORE AS SECUREFILE ( COMPRESS HIGH)
)
/
```

The compression_type column is used as a distinct key to drive the lob into a compression setting of none, medium or high. The document_type column is used to distinct different type of documents. As you can see this table is easily extensible to investigate other types of documents.

Because of possible interference by SMCO space allocation space should be pre-allocated, but then obviously I cannot find the actual compression rate anymore. So no pre-allocation in this test.

Loading documents into the table is performed with a PL/SQL block. This PL/SQL block loads using external table FILES that was created initially. The PL/SQL block is:

```
DECLARE
  l_file        BFILE;
  l_blob        BLOB;
BEGIN
  -- Loop through all testdata
  FOR r_testdata IN ( SELECT document_type, document_name, num_loads
                      FROM testdata_configuration ) LOOP

    -- Open the file
    l_file := BFILENAME('FILES', r_testdata.document_name);
    DBMS_LOB.OPEN(l_file, DBMS_LOB.LOB_READONLY);

    -- run for as many loops as configured
    FOR i IN 1..r_testdata.num_loads LOOP

      -- Insert using no compression
      INSERT INTO filestorage (did, compression_type, document_type, bfiledata )
        VALUES ( seq_did.nextval, 'NONE', r_testdata.document_type,
empty_blob() )
        RETURNING bfiledata INTO l_blob;

      DBMS_LOB.OPEN(l_blob, DBMS_LOB.LOB_READWRITE);
      DBMS_LOB.LOADFROMFILE(DEST_LOB => l_blob,
                            SRC_LOB  => l_file,
                            AMOUNT   => DBMS_LOB.GETLENGTH(l_file));
      DBMS_LOB.CLOSE(l_blob);

      -- Insert using medium compression
      INSERT INTO filestorage (did, compression_type, document_type, bfiledata )
        VALUES ( seq_did.nextval, 'MEDIUM', r_testdata.document_type,
empty_blob() )
        RETURNING bfiledata INTO l_blob;

      DBMS_LOB.OPEN(l_blob, DBMS_LOB.LOB_READWRITE);
      DBMS_LOB.LOADFROMFILE(DEST_LOB => l_blob,
                            SRC_LOB  => l_file,
                            AMOUNT   => DBMS_LOB.GETLENGTH(l_file));
      DBMS_LOB.CLOSE(l_blob);

      -- Insert using high compression
      INSERT INTO filestorage (did, compression_type, document_type, bfiledata )
        VALUES ( seq_did.nextval, 'HIGH', r_testdata.document_type,
empty_blob() )
        RETURNING bfiledata INTO l_blob;

      DBMS_LOB.OPEN(l_blob, DBMS_LOB.LOB_READWRITE);
      DBMS_LOB.LOADFROMFILE(DEST_LOB => l_blob,
                            SRC_LOB => l_file,
                            AMOUNT   => DBMS_LOB.GETLENGTH(l_file));
      DBMS_LOB.CLOSE(l_blob);

    END LOOP;

    -- Close your bfile
    DBMS_LOB.CLOSE(l_file);

  END LOOP;

END;
/
```

After running the block stats have to be created to be able to validate the actual number of rows in the partitions. Stats should be generated using the DBMS_STATS procedure[31]. To make sure that the stats are 100% correct the estimate_percent is set to NULL. The granularity is set to SUBPARTITION to allow the usage of global stats.

```
exec dbms_stats.gather_table_stats(ownname => 'DEMO',tabname => 'FILESTORAGE',
cascade => TRUE, estimate_percent => NULL, granularity => 'SUBPARTITION');
```

After the stats procedure has finished, we can query the user_tab_subpartition view to validate the dataset.

```
SQL> SELECT partition_name, subpartition_name, num_rows
  2    FROM user_tab_subpartitions
  3   WHERE table_name = 'FILESTORAGE'
  4*  ORDER BY partition_name
SQL> /

PARTITION_NAME                 SUBPARTITION_NAME                NUM_ROWS
------------------------------ ------------------------------ ----------
P_HIGH                         P_HIGH_P_PDFX                         300
P_HIGH                         P_HIGH_P_DOC                          450
P_HIGH                         P_HIGH_P_RTF                          300
P_HIGH                         P_HIGH_P_PDFA                        1100
P_MEDIUM                       P_MEDIUM_P_RTF                        300
P_MEDIUM                       P_MEDIUM_P_DOC                        450
P_MEDIUM                       P_MEDIUM_P_PDFA                      1100
P_MEDIUM                       P_MEDIUM_P_PDFX                       300
P_NONE                         P_NONE_P_PDFX                         300
P_NONE                         P_NONE_P_RTF                          300
P_NONE                         P_NONE_P_DOC                          450
P_NONE                         P_NONE_P_PDFA                        1100

12 rows selected.
```

To find the actual size of the compressed SecureFile you have to look for the size of the lob subpartition. The name of the subpartition is system generated, but since we are using a list subpartition based on document type we can find out the actual document type in the lob subpartition by querying the high_value of the table subpartition:

```
SQL> SELECT uts.high_value, uts.num_rows, uls.compression,
  2         ( SELECT SUM(bytes)
  3             FROM user_segments
  4            WHERE partition_name = uls.lob_subpartition_name ) as lob_size
  5    FROM user_lob_subpartitions uls,
  6         user_tab_subpartitions uts
  7   WHERE uls.table_name = 'FILESTORAGE'
  8     AND uls.column_name = 'BFILEDATA'
  9     AND uts.table_name = uls.table_name
 10     AND uts.subpartition_name = uls.subpartition_name
 11*    AND uts.subpartition_position = uls.subpartition_position
SQL> /
```

31 http://docs.oracle.com/cd/E11882_01/appdev.112/e25788/d_stats.htm#i1036461

```
HIGH_VALUE   NUM_ROWS COMPRE   LOB_SIZE
---------- ---------- ------ ----------
'PDF/A'          1100 NO      318767104
'PDF/X'           300 NO       92274688
'DOC'             450 NO      117440512
'RTF'             300 NO       41943040
'PDF/A'          1100 MEDIUM   58720256
'PDF/X'           300 MEDIUM   67108864
'DOC'             450 MEDIUM   16777216
'RTF'             300 MEDIUM    8388608
'PDF/A'          1100 HIGH     50331648
'PDF/X'           300 HIGH     67108864
'DOC'             450 HIGH      8388608
'RTF'             300 HIGH      8388608

12 rows selected.
```

Now when analyzing these number a bit further, I calculated the compression rate. The formula used is (obviously): `( ( compressed_size - nocompress_size ) / nocompress_size) * 100%`

	PDF/A		PDF/X		DOC		RTF	
	Absolute	Rate	Absolute	Rate	Absolute	Rate	Absolute	Rate
NONE	318767104	0%	92274688	0%	117440512	0%	41943040	0%
MEDIUM	58720256	-82%	67108864	-27%	16777216	-86%	8388608	-80%
HIGH	50331648	-84%	67108864	-27%	8388608	-93%	8388608	-80%

Analyzing these numbers will lead to the following conclusions for this specific dataset:
- Not all documents get the same compression rate (no surprise huh)
- Compression rate high doesn't compress documents to a much smaller size than compression rate medium does.

So far for compression rate. Now on to the cost of compression.

To investigate the cost of compression the same table was used, including the same configuration. What did change was the pre-allocation (it was pre-allocated now) and the PL/SQL block loading the data:

```
DECLARE
  lv_start      NUMBER;
  lv_start_cpu  NUMBER;
  l_file        BFILE;
  l_blob        BLOB;
BEGIN
  -- Loop through all testdata
  lv_start     := DBMS_UTILITY.GET_TIME;
  lv_start_cpu := DBMS_UTILITY.GET_CPU_TIME;

  FOR r_testdata IN ( SELECT document_type, document_name, num_loads
                      FROM testdata_configuration ) LOOP

    -- Open the file
    l_file := BFILENAME('FILES', r_testdata.document_name);
    DBMS_LOB.OPEN(l_file, DBMS_LOB.LOB_READONLY);

    -- run for as many loops as configured
    FOR i IN 1..r_testdata.num_loads LOOP

      -- Insert using no compression
      INSERT INTO filestorage (did, compression_type, document_type,
document_name, bfiledata )
```

```
        VALUES ( seq_did.nextval, 'NONE', r_testdata.document_type,
r_testdata.document_name, empty_blob() )
        RETURNING bfiledata INTO l_blob;

      DBMS_LOB.OPEN(l_blob, DBMS_LOB.LOB_READWRITE);
      DBMS_LOB.LOADFROMFILE(DEST_LOB => l_blob,SRC_LOB => l_file,AMOUNT   =>
DBMS_LOB.GETLENGTH(l_file));
      DBMS_LOB.CLOSE(l_blob);

    END LOOP;

    -- Close your bfile
    DBMS_LOB.CLOSE(l_file);

  END LOOP;

  DBMS_OUTPUT.put_line('loaded uncompressed filestorage: ' ||
(DBMS_UTILITY.GET_TIME - lv_start) || ' hsecs, cpu: ' ||
(DBMS_UTILITY.GET_CPU_TIME - lv_start_cpu) || ' hsecs');

  lv_start     := DBMS_UTILITY.GET_TIME;
  lv_start_cpu := DBMS_UTILITY.GET_CPU_TIME;

  FOR r_testdata IN ( SELECT document_type, document_name, num_loads
                      FROM testdata_configuration ) LOOP

    -- Open the file
    l_file := BFILENAME('FILES', r_testdata.document_name);
    DBMS_LOB.OPEN(l_file, DBMS_LOB.LOB_READONLY);

    -- run for as many loops as configured
    FOR i IN 1..r_testdata.num_loads LOOP

      -- Insert using no compression
      INSERT INTO filestorage (did, compression_type, document_type,
document_name, bfiledata )
        VALUES ( seq_did.nextval, 'MEDIUM', r_testdata.document_type,
r_testdata.document_name, empty_blob() )
        RETURNING bfiledata INTO l_blob;

      DBMS_LOB.OPEN(l_blob, DBMS_LOB.LOB_READWRITE);
      DBMS_LOB.LOADFROMFILE(DEST_LOB => l_blob,SRC_LOB => l_file,AMOUNT   =>
DBMS_LOB.GETLENGTH(l_file));
      DBMS_LOB.CLOSE(l_blob);

    END LOOP;

    -- Close your bfile
    DBMS_LOB.CLOSE(l_file);

  END LOOP;

  DBMS_OUTPUT.put_line('loaded medium compressed filestorage: ' ||
(DBMS_UTILITY.GET_TIME - lv_start) || ' hsecs, cpu: ' ||
(DBMS_UTILITY.GET_CPU_TIME - lv_start_cpu) || ' hsecs');

  lv_start     := DBMS_UTILITY.GET_TIME;
  lv_start_cpu := DBMS_UTILITY.GET_CPU_TIME;

  FOR r_testdata IN ( SELECT document_type, document_name, num_loads
                      FROM testdata_configuration ) LOOP

    -- Open the file
    l_file := BFILENAME('FILES', r_testdata.document_name);
    DBMS_LOB.OPEN(l_file, DBMS_LOB.LOB_READONLY);

    -- run for as many loops as configured
    FOR i IN 1..r_testdata.num_loads LOOP

      -- Insert using no compression
      INSERT INTO filestorage (did, compression_type, document_type,
document_name, bfiledata )
        VALUES ( seq_did.nextval, 'HIGH', r_testdata.document_type,
r_testdata.document_name, empty_blob() )
        RETURNING bfiledata INTO l_blob;

      DBMS_LOB.OPEN(l_blob, DBMS_LOB.LOB_READWRITE);
      DBMS_LOB.LOADFROMFILE(DEST_LOB => l_blob,SRC_LOB => l_file,AMOUNT   =>
DBMS_LOB.GETLENGTH(l_file));
      DBMS_LOB.CLOSE(l_blob);

    END LOOP;

    -- Close your bfile
    DBMS_LOB.CLOSE(l_file);

  END LOOP;

  DBMS_OUTPUT.put_line('loaded high compressed filestorage: ' ||
(DBMS_UTILITY.GET_TIME - lv_start) || ' hsecs, cpu: ' ||
(DBMS_UTILITY.GET_CPU_TIME - lv_start_cpu) || ' hsecs');

END;
/
```

The results were these:

```
loaded uncompressed filestorage: 1953 hsecs, cpu: 303 hsecs
loaded high compressed filestorage: 1281 hsecs, cpu: 711 hsecs
loaded medium compressed filestorage: 1099 hsecs, cpu: 526 hsecs
```

Now these numbers seem off, which is caused by the amount of testdata. Larger datasets (e.g. the complete Oracle 11R2 documentation set) give different results:

```
loaded uncompressed filestorage: 23950 hsecs, cpu: 4467 hsecs
loaded high compressed filestorage: 24340 hsecs, cpu: 12772 hsecs
loaded medium compressed filestorage: 22530 hsecs, cpu: 10100 hsecs
```

This shows not only that more compression uses more CPU, but that some compression can also perform better.

The specific Siebel Document Store that started this paper does not use a full text index to search through documents, therefore the impact of compression on such an index was not investigated.

Since documents are not only stored, it is relevant to investigate the impact of compression on retrieving of documents too. To test what the impact of compression is on retrieving of documents is, the previously created and populated filestorage table will be reused. As a test-case some documents are retrieved and the SQL is traced to look for differences. The document DID's to retrieve were based on this query:

```
SQL> SELECT compression_type, min(did), max(did)
  2    FROM filestorage
  3*  GROUP BY compression_type
SQL> /

COMPRE   MIN(DID)   MAX(DID)
------ ---------- ----------
NONE            1      17200
MEDIUM       2151      19350
HIGH         4301      15050

3 rows selected.
```

To find out if there is a noticeable difference in selecting compressed SecureFile lobs the following PL/SQL block was used:

```
set serveroutput on
DECLARE

  PROCEDURE flush_all
  IS
  BEGIN
    EXECUTE IMMEDIATE 'ALTER SYSTEM FLUSH shared_pool';
    EXECUTE IMMEDIATE 'ALTER SYSTEM FLUSH buffer_cache';
  END flush_all;

  PROCEDURE select_record
    (pp_did IN filestorage.did%TYPE)
  IS
    lv_blob   filestorage.bfiledata%TYPE;
    lv_start  NUMBER;
  BEGIN
    lv_start := DBMS_UTILITY.GET_TIME;
    SELECT bfiledata INTO lv_blob FROM filestorage WHERE did=pp_did;
    DBMS_OUTPUT.put_line('1: ' || (DBMS_UTILITY.GET_TIME - lv_start) ||
                         ' hsecs');
  END select_record;

BEGIN
  flush_all;
  select_record(1);
  flush_all;
  select_record(17200);
  flush_all;
  select_record(2151);
  flush_all;
  select_record(19350);
  flush_all;
  select_record(4301);
  flush_all;
  select_record(15050);
END;
/
```

This led to the following output:

```
1: 16 hsecs
17200: 19 hsecs
2151: 19 hsecs
19350: 18 hsecs
4301: 19 hsecs
15050: 17 hsecs
```

The results are so close, that in my opinion it is safe to draw the conclusion that there is no noticeable performance difference in retrieving compressed SecureFile lobs to nocompress SecureFile lobs.

As last test on compression I wanted to find out more about how table compression relates to SecureFile lob compression. To do this, I created three tables:

```
CREATE TABLE filestorage
( did               NUMBER NOT NULL,
  bfiledata         BLOB,
  CONSTRAINT pk_filestorage PRIMARY KEY (did)
)
TABLESPACE hotsos_demos
LOB (bfiledata) STORE AS SECUREFILE
  (
    TABLESPACE hotsos_demo_lobs DISABLE STORAGE IN ROW NOCACHE
  )
/

CREATE TABLE filestorage_tc
( did               NUMBER NOT NULL,
  bfiledata         BLOB,
 CONSTRAINT pk_filestorage_tc PRIMARY KEY (did)
)
TABLESPACE hotsos_demos COMPRESS
LOB (bfiledata) STORE AS SECUREFILE
  (
    TABLESPACE hotsos_demo_lobs DISABLE STORAGE IN ROW NOCACHE
  )
/

CREATE TABLE filestorage_tlc
( did               NUMBER NOT NULL,
  bfiledata         BLOB,
  CONSTRAINT pk_filestorage_tlc PRIMARY KEY (did)
)
TABLESPACE hotsos_demos COMPRESS
LOB (bfiledata) STORE AS SECUREFILE
  (
TABLESPACE hotsos_demo_lobs DISABLE STORAGE IN ROW NOCACHE COMPRESS
  )
/
```

As you can see, the first table (filestorage) uses no compression whatsoever, the second one (filestorage_tc) uses compression at table level and not at SecureFile lob level, the last table (filestorage_tlc) uses compression at table and SecureFile lob level. I validated the configuration by running a simple query:

```
SELECT utl.table_name, utl.compression as table_compression, utl.compress_for,
ulb.compression as lob_compression
  FROM user_tables   utl,
       user_lobs     ulb
 WHERE utl.table_name = ulb.table_name
ORDER BY utl.table_name
/

TABLE_NAME                     TABLE_CO COMPRESS_FOR LOB_CO
------------------------------ -------- ------------ ------
FILESTORAGE                    DISABLED              NO
FILESTORAGE_TC                 ENABLED  BASIC        NO
FILESTORAGE_TLC                ENABLED  BASIC        MEDIUM

3 rows selected.
```

After resizing the hotsos_demo_lobs tablespace to a size that can contain all the data that I will load (to prevent SMCO from blocking me) I ran some PL/SQL to load data into the tables. This PL/SQL looks like the PL/SQL we used before, it loads data from an external directory. The data loaded was the full Oracle 11R2 database documentation:

```
DECLARE
  lv_file       BFILE;
  lv_blob       BLOB;
  lv_start      NUMBER;
  lv_start_cpu  NUMBER;
  lv_directory  VARCHAR2(1024) := '/u01/files/docs';
  lv_ns         VARCHAR2(1024);
  e_21560       EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_21560,-21560);
BEGIN
  -- Search the files
  SYS.DBMS_BACKUP_RESTORE.SEARCHFILES(lv_directory, lv_ns);

  lv_start     := DBMS_UTILITY.GET_TIME;
  lv_start_cpu := DBMS_UTILITY.GET_CPU_TIME;

  FOR each_file IN (SELECT fname_krbmsft AS name
                      FROM sys.xkrbmsft) LOOP
    -- Open the file
    lv_file := BFILENAME('DOCS', substr(each_file.name, length(lv_directory) +
2) );
    DBMS_LOB.OPEN(lv_file, DBMS_LOB.LOB_READONLY);

    BEGIN
      INSERT INTO filestorage (did, bfiledata )
        VALUES ( seq_did.nextval, empty_blob() )
        RETURNING bfiledata INTO lv_blob;

      DBMS_LOB.OPEN(lv_blob, DBMS_LOB.LOB_READWRITE);
      DBMS_LOB.LOADFROMFILE(DEST_LOB => lv_blob,SRC_LOB => lv_file,AMOUNT   =>
DBMS_LOB.GETLENGTH(lv_file));
      DBMS_LOB.CLOSE(lv_blob);
    EXCEPTION
      WHEN e_21560 THEN
        DBMS_LOB.CLOSE(lv_blob);
        DBMS_OUTPUT.PUT_LINE('error inserting: ' || each_file.name);
    END;

    DBMS_LOB.CLOSE(lv_file);

  END LOOP;

  DBMS_OUTPUT.PUT_LINE('loaded in filestorage: ' || (DBMS_UTILITY.GET_TIME -
lv_start) || ' hsecs, cpu: ' || (DBMS_UTILITY.GET_CPU_TIME - lv_start_cpu) || '
hsecs');

  lv_start     := DBMS_UTILITY.GET_TIME;
  lv_start_cpu := DBMS_UTILITY.GET_CPU_TIME;

  FOR each_file IN (SELECT fname_krbmsft AS name
                      FROM sys.xkrbmsft) LOOP
    -- Open the file
    lv_file := BFILENAME('DOCS', substr(each_file.name, length(lv_directory) +
2) );
    DBMS_LOB.OPEN(lv_file, DBMS_LOB.LOB_READONLY);

    BEGIN
      INSERT INTO filestorage_tc (did, bfiledata )
```

```
        VALUES ( seq_did.nextval, empty_blob() )
        RETURNING bfiledata INTO lv_blob;

      DBMS_LOB.OPEN(lv_blob, DBMS_LOB.LOB_READWRITE);
      DBMS_LOB.LOADFROMFILE(DEST_LOB => lv_blob,SRC_LOB => lv_file,AMOUNT   =>
DBMS_LOB.GETLENGTH(lv_file));
      DBMS_LOB.CLOSE(lv_blob);
    EXCEPTION
      WHEN e_21560 THEN
        DBMS_LOB.CLOSE(lv_blob);
        DBMS_OUTPUT.PUT_LINE('error inserting: ' || each_file.name);
    END;

    DBMS_LOB.CLOSE(lv_file);

  END LOOP;

  DBMS_OUTPUT.PUT_LINE('loaded in filestorage_tc: ' || (DBMS_UTILITY.GET_TIME -
lv_start) || ' hsecs, cpu: ' || (DBMS_UTILITY.GET_CPU_TIME - lv_start_cpu) || '
hsecs');

  lv_start     := DBMS_UTILITY.GET_TIME;
  lv_start_cpu := DBMS_UTILITY.GET_CPU_TIME;

  FOR each_file IN (SELECT fname_krbmsft AS name
                      FROM sys.xkrbmsft) LOOP
    -- Open the file
    lv_file := BFILENAME('DOCS', substr(each_file.name, length(lv_directory) +
2) );
    DBMS_LOB.OPEN(lv_file, DBMS_LOB.LOB_READONLY);

    BEGIN
      INSERT INTO filestorage_tlc (did, bfiledata )
        VALUES ( seq_did.nextval, empty_blob() )
        RETURNING bfiledata INTO lv_blob;

      DBMS_LOB.OPEN(lv_blob, DBMS_LOB.LOB_READWRITE);
      DBMS_LOB.LOADFROMFILE(DEST_LOB => lv_blob,SRC_LOB => lv_file,AMOUNT   =>
DBMS_LOB.GETLENGTH(lv_file));
      DBMS_LOB.CLOSE(lv_blob);
    EXCEPTION
      WHEN e_21560 THEN
        DBMS_LOB.CLOSE(lv_blob);
        DBMS_OUTPUT.PUT_LINE('error inserting: ' || each_file.name);
    END;

    DBMS_LOB.CLOSE(lv_file);

  END LOOP;

  DBMS_OUTPUT.PUT_LINE('loaded in filestorage_tlc: ' || (DBMS_UTILITY.GET_TIME -
lv_start) || ' hsecs, cpu: ' || (DBMS_UTILITY.GET_CPU_TIME - lv_start_cpu) || '
hsecs');

END;
/
```

The results from the PL/SQL procedure were these:

```
loaded in filestorage: 23157 hsecs, cpu: 4236 hsecs
loaded in filestorage_tc: 23399 hsecs, cpu: 4926 hsecs
loaded in filestorage_tlc: 22642 hsecs, cpu: 10081 hsecs
```

So adding compression at table level adds some CPU usage and runtime to the procedure. When compressing at SecureFile lob level much more CPU is used but the total runtime of the PL/SQL code is shorter.

Next I ran some SQL to check for the size of the lobs in the database:
```
SELECT utl.table_name, utl.compression as table_compression, utl.compress_for,
ulb.compression as lob_compression, sum(ust.bytes)/1024/1024 as size_in_mb
  FROM user_tables   utl,
       user_lobs     ulb,
       user_segments ust
 WHERE utl.table_name = ulb.table_name
   AND ust.segment_name = ulb.segment_name
 GROUP BY utl.table_name, utl.compression, utl.compress_for, ulb.compression
 ORDER BY utl.table_name
/

TABLE_NAME                     TABLE_CO COMPRESS_FOR LOB_CO SIZE_IN_MB
------------------------------ -------- ------------ ------ ----------
FILESTORAGE                    DISABLED              NO      2496.1875
FILESTORAGE_TC                 ENABLED  BASIC        NO      2496.1875
FILESTORAGE_TLC                ENABLED  BASIC        MEDIUM  1479.1875

3 rows selected.
```

The output of this query shows that compression at table level has no impact on the size of the lob segments. So the added runtime and CPU usage was only used for compressing the data that is not lob. Compression at SecureFile lob level does lower the actual size of the lobs.

Finally I checked the actual size of the lobs:

```
DECLARE
  lv_segment_size_blocks NUMBER;
  lv_segment_size_bytes  NUMBER;
  lv_used_blocks         NUMBER;
  lv_used_bytes          NUMBER;
  lv_expired_blocks      NUMBER;
  lv_expired_bytes       NUMBER;
  lv_unexpired_blocks    NUMBER;
  lv_unexpired_bytes     NUMBER;
  lv_segment_name        user_segments.segment_name%type;
BEGIN

  FOR r_lob IN ( SELECT table_name, segment_name
                   FROM user_lobs ) LOOP

    DBMS_SPACE.SPACE_USAGE(
      segment_owner       => 'DEMO',
      segment_name        => r_lob.segment_name,
      segment_type        => 'LOB',
      segment_size_blocks => lv_segment_size_blocks,
      segment_size_bytes  => lv_segment_size_bytes,
      used_blocks         => lv_used_blocks,
      used_bytes          => lv_used_bytes,
      expired_blocks      => lv_expired_blocks,
      expired_bytes       => lv_expired_bytes,
      unexpired_blocks    => lv_unexpired_blocks,
      unexpired_bytes     => lv_unexpired_bytes,
      partition_name      => NULL );

    DBMS_OUTPUT.PUT_LINE('table name: ' || r_lob.table_name);
    DBMS_OUTPUT.PUT_LINE('used bytes : '   || lv_used_bytes);
    DBMS_OUTPUT.PUT_LINE('used blocks : '  || lv_used_blocks);
  END LOOP;

END;
/

table name: FILESTORAGE
used bytes : 2444460032
used blocks : 298396
table name: FILESTORAGE_TC
used bytes : 2461835264
used blocks : 300517
table name: FILESTORAGE_TLC
used bytes : 1372405760
used blocks : 167530
```

As you can see in my testresults, the size of the LOB in the database only changes by using SecureFile lob compression. Compression on table level has no influence on the actual size of the SecureFile lob.

Deduplication

When investigating deduplication for SecureFile lobs the following questions arose:

- How expensive is duplication?
- Does deduplication scale?
- Is duplication more expensive when SecureFile compression is used?

To investigate the working of deduplication the filestorage table is setup to support deduplication. Since deduplication cannot span subpartitions or partitions, a non partitioned table is used. This should give the same results as a partitioned table. The following statement reflects the filestorage table as used in this test:

```
CREATE TABLE filestorage
( did                NUMBER NOT NULL,
  document_name      VARCHAR2(1024) NOT NULL,
  bfiledata          BLOB,
  CONSTRAINT pk_filestorage PRIMARY KEY (did)
)
TABLESPACE hotsos_demos
LOB (bfiledata) STORE AS SECUREFILE
   (
     TABLESPACE hotsos_demo_lobs DISABLE STORAGE IN ROW NOCACHE DEDUPLICATE
   )
/
```

As a base dataset of 24102 documents[32] were loaded into the filestorage table. To load these documents, the following PL/SQL was used:

```
DECLARE
  lv_file       BFILE;
  lv_blob       BLOB;
  lv_start      NUMBER;
  lv_start_cpu  NUMBER;
  lv_directory  VARCHAR2(1024) := '/u01/files/docs';
  lv_ns         VARCHAR2(1024);
  e_21560       EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_21560,-21560);
BEGIN
  -- Search the files
```

32 Since deduplications uses a bitmap index, there cannot be a performance difference in the type of document used. Therefore a bunch of not so random html files were used: a bunch of Oracle documentation from E11882_01.zip. This is the full 11R2 database documentation

```
  SYS.DBMS_BACKUP_RESTORE.SEARCHFILES(lv_directory, lv_ns);

  lv_start     := DBMS_UTILITY.GET_TIME;
  lv_start_cpu := DBMS_UTILITY.GET_CPU_TIME;

  FOR each_file IN (SELECT fname_krbmsft AS name
                      FROM sys.xkrbmsft) LOOP
    -- Open the file
    lv_file := BFILENAME('DOCS', substr(each_file.name, length(lv_directory) +
2) );
    DBMS_LOB.OPEN(lv_file, DBMS_LOB.LOB_READONLY);

    BEGIN
      INSERT INTO filestorage (did, document_name, bfiledata )
        VALUES ( seq_did.nextval, each_file.name, empty_blob() )
        RETURNING bfiledata INTO lv_blob;

      DBMS_LOB.OPEN(lv_blob, DBMS_LOB.LOB_READWRITE);
      DBMS_LOB.LOADFROMFILE(DEST_LOB => lv_blob,
                            SRC_LOB  => lv_file,
                            AMOUNT   => DBMS_LOB.GETLENGTH(lv_file));
      DBMS_LOB.CLOSE(lv_blob);
    EXCEPTION
      WHEN e_21560 THEN
        DBMS_LOB.CLOSE(lv_blob);
        DBMS_OUTPUT.PUT_LINE('error inserting: ' || each_file.name);
    END;

    DBMS_LOB.CLOSE(lv_file);

  END LOOP;

  DBMS_OUTPUT.put_line('loaded in: ' || (DBMS_UTILITY.GET_TIME - lv_start) ||
                       ' hsecs, cpu time:' ||
                       (DBMS_UTILITY.GET_TIME - lv_start_cpu) || ' hsecs');

END;
/

error inserting: /u01/files/docs/E11882_01/dcommon/css/darbbook.css
loaded in: 23272 hsecs, cpu time:434961153 hsec

PL/SQL procedure successfully completed.
```

Next the content of the dataset is checked by running this SQL:

```
SQL> SELECT num_rows, blocks
  2    FROM user_tables
  3   WHERE table_name = 'FILESTORAGE'
  4  /

  NUM_ROWS     BLOCKS
---------- ----------
     24102        496

1 row selected.

SQL> SELECT ulb.table_name, ulb.column_name, ulb.deduplication,
  2         SUM(ust.bytes)/1024/1024 as size_in_mb
  3    FROM user_lobs     ulb,
  4         user_segments ust
  5   WHERE ust.segment_name = ulb.segment_name
  6*  GROUP BY ulb.table_name, ulb.column_name, ulb.deduplication
```

```
SQL> /

TABLE_NAME                 COLUMN_NAME                DEDUPLICATION   SIZE_IN_MB
-------------------------- -------------------------- --------------- ----------
FILESTORAGE                BFILEDATA                  LOB              2375.1875

1 row selected.
```

So loading the same dataset into the filestorage table again should result in double the rows, but no extra data stored in lobsegments:

```
DECLARE
  lv_file       BFILE;
  lv_blob       BLOB;
  lv_start      NUMBER;
  lv_start_cpu  NUMBER;
  lv_directory  VARCHAR2(1024) := '/u01/files/docs';
  lv_ns         VARCHAR2(1024);
  e_21560       EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_21560,-21560);
BEGIN
  -- Search the files
  lv_start     := DBMS_UTILITY.GET_TIME;
  lv_start_cpu := DBMS_UTILITY.GET_CPU_TIME;

  FOR each_file IN (SELECT fname_krbmsft AS name
                      FROM sys.xkrbmsft) LOOP
    -- Open the file
    lv_file := BFILENAME('DOCS', substr(each_file.name, length(lv_directory) +
2) );
    DBMS_LOB.OPEN(lv_file, DBMS_LOB.LOB_READONLY);

    BEGIN
      INSERT INTO filestorage (did, document_name, bfiledata )
        VALUES ( seq_did.nextval, each_file.name, empty_blob() )
        RETURNING bfiledata INTO lv_blob;

      DBMS_LOB.OPEN(lv_blob, DBMS_LOB.LOB_READWRITE);
      DBMS_LOB.LOADFROMFILE(DEST_LOB => lv_blob,
                            SRC_LOB  => lv_file,
                            AMOUNT   => DBMS_LOB.GETLENGTH(lv_file));
      DBMS_LOB.CLOSE(lv_blob);
    EXCEPTION
      WHEN e_21560 THEN
        DBMS_LOB.CLOSE(lv_blob);
        DBMS_OUTPUT.PUT_LINE('error inserting: ' || each_file.name);
    END;

    DBMS_LOB.CLOSE(lv_file);

  END LOOP;

  DBMS_OUTPUT.put_line('loaded in: ' || (DBMS_UTILITY.GET_TIME - lv_start) ||
                       ' hsecs, cpu time:' ||
                       (DBMS_UTILITY.GET_TIME - lv_start_cpu) || ' hsecs');

END;
/

error inserting: /u01/files/docs/E11882_01/dcommon/css/darbbook.css
loaded in: 9707 hsecs, cpu time: 434964415 hsecs

PL/SQL procedure successfully completed.
```

Statistics need to be gathered for reporting and the report queries are run again:

```
SQL> exec dbms_stats.gather_table_stats(ownname => 'DEMO',tabname =>
'FILESTORAGE', cascade => TRUE, estimate_percent => NULL);

SQL> SELECT num_rows, blocks
  2    FROM user_tables
  3   WHERE table_name = 'FILESTORAGE'
  4  /

  NUM_ROWS     BLOCKS
---------- ----------
     48204       1000

1 row selected.

SQL> SELECT ulb.table_name, ulb.column_name, ulb.deduplication,
  2         SUM(ust.bytes)/1024/1024 as size_in_mb
  3    FROM user_lobs     ulb,
  4         user_segments ust
  5   WHERE ust.segment_name = ulb.segment_name
  6*  GROUP BY ulb.table_name, ulb.column_name, ulb.deduplication
SQL> /

TABLE_NAME                COLUMN_NAME               DEDUPLICATION   SIZE_IN_MB
------------------------- ------------------------- --------------- ----------
FILESTORAGE               BFILEDATA                 LOB              2375.1875

1 row selected.
```

Looking at these results, the following can be concluded:

- Deduplication works as expected: no duplicate lobs are stored in the lobsegments
- Not having to store the lob gives a significant performance benefit compared to storing the lob (duh!).

For the Siebel Document Store the notion of duplicate documents doesn't exist, there is always a minimal difference. If for some sort of reason duplicates do end up in the content store the number of duplicates is expected to be very low. Therefore it is interesting to find out what is the performance penalty for adding deduplication as setting to your SecureFile lobs configuration.

To test the actual performance penalty of deduplication two filestorage tables were setup. One with deduplication and one without deduplication:

```
CREATE TABLE filestorage
( did              NUMBER NOT NULL,
  document_name    VARCHAR2(1024) NOT NULL,
  bfiledata        BLOB,
  CONSTRAINT pk_filestorage PRIMARY KEY (did)
)
TABLESPACE hotsos_demos
LOB (bfiledata) STORE AS SECUREFILE
  (
    TABLESPACE hotsos_demo_lobs DISABLE STORAGE IN ROW NOCACHE
  )
/
```

```
CREATE TABLE filestorage_dedup
( did                NUMBER NOT NULL,
  document_name      VARCHAR2(1024) NOT NULL,
  bfiledata          BLOB,
  CONSTRAINT pk_filestorage PRIMARY KEY (did)
)
TABLESPACE hotsos_demos
LOB (bfiledata) STORE AS SECUREFILE
  (
    TABLESPACE hotsos_demo_lobs DISABLE STORAGE IN ROW NOCACHE DEDUPLICATE
  )
/
```

Next the 11R2 database documention was loaded into both tables, while timing the results:

```
DECLARE
  lv_file        BFILE;
  lv_blob        BLOB;
  lv_start       NUMBER;
  lv_start_cpu   NUMBER;
  lv_directory   VARCHAR2(1024) := '/u01/files/docs';
  lv_ns          VARCHAR2(1024);
  e_21560        EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_21560,-21560);
BEGIN
  -- Search the files
  SYS.DBMS_BACKUP_RESTORE.SEARCHFILES(lv_directory, lv_ns);

  lv_start     := DBMS_UTILITY.GET_TIME;
  lv_start_cpu := DBMS_UTILITY.GET_CPU_TIME;

  FOR each_file IN (SELECT fname_krbmsft AS name
                      FROM sys.xkrbmsft) LOOP
    -- Open the file
    lv_file := BFILENAME('DOCS', substr(each_file.name, length(lv_directory) +
2) );
    DBMS_LOB.OPEN(lv_file, DBMS_LOB.LOB_READONLY);

    BEGIN
      INSERT INTO filestorage (did, document_name, bfiledata )
        VALUES ( seq_did.nextval, each_file.name, empty_blob() )
        RETURNING bfiledata INTO lv_blob;

      DBMS_LOB.OPEN(lv_blob, DBMS_LOB.LOB_READWRITE);
      DBMS_LOB.LOADFROMFILE(DEST_LOB => lv_blob,
                            SRC_LOB  => lv_file,
                            AMOUNT   => DBMS_LOB.GETLENGTH(lv_file));
      DBMS_LOB.CLOSE(lv_blob);
    EXCEPTION
      WHEN e_21560 THEN
        DBMS_LOB.CLOSE(lv_blob);
        DBMS_OUTPUT.PUT_LINE('error inserting: ' || each_file.name);
    END;

    DBMS_LOB.CLOSE(lv_file);

  END LOOP;

  DBMS_OUTPUT.put_line('loaded in filestorage: ' ||
                       (DBMS_UTILITY.GET_TIME - lv_start) || ' hsecs, cpu: ' ||
                       (DBMS_UTILITY.GET_CPU_TIME - lv_start_cpu) || ' hsecs');

  lv_start     := DBMS_UTILITY.GET_TIME;
  lv_start_cpu := DBMS_UTILITY.GET_CPU_TIME;

  FOR each_file IN (SELECT fname_krbmsft AS name
                    FROM sys.xkrbmsft) LOOP
    -- Open the file
    lv_file := BFILENAME('DOCS', substr(each_file.name, length(lv_directory) +
2) );
    DBMS_LOB.OPEN(lv_file, DBMS_LOB.LOB_READONLY);

    BEGIN
      INSERT INTO filestorage_dedup (did, document_name, bfiledata )
        VALUES ( seq_did.nextval, each_file.name, empty_blob() )
        RETURNING bfiledata INTO lv_blob;

      DBMS_LOB.OPEN(lv_blob, DBMS_LOB.LOB_READWRITE);
      DBMS_LOB.LOADFROMFILE(DEST_LOB => lv_blob,
                            SRC_LOB  => lv_file,
                            AMOUNT   => DBMS_LOB.GETLENGTH(lv_file));
      DBMS_LOB.CLOSE(lv_blob);
    EXCEPTION
      WHEN e_21560 THEN
        DBMS_LOB.CLOSE(lv_blob);
        DBMS_OUTPUT.PUT_LINE('error inserting: ' || each_file.name);
    END;

    DBMS_LOB.CLOSE(lv_file);

  END LOOP;

  DBMS_OUTPUT.put_line('loaded in filestorage_dedup: ' ||
                       (DBMS_UTILITY.GET_TIME - lv_start) || ' hsecs, cpu: ' ||
                       (DBMS_UTILITY.GET_CPU_TIME - lv_start_cpu) || ' hsecs');

END;
/
```

The results were these:

```
error inserting: /u01/files/docs/E11882_01/dcommon/css/darbbook.css
loaded in filestorage: 32314 hsecs, cpu: 5275 hsecs
error inserting: /u01/files/docs/E11882_01/dcommon/css/darbbook.css
loaded in filestorage_dedup: 34814 hsecs, cpu: 7700 hsecs
```

These results show that inserting data into a SecureFile lobs lob with deduplication takes approximately 7% more time. Other tests with the same set showed a larger difference. When the dataset is enlarged with another 89519 documents[33], the results are these:

```
error inserting: /u01/files/docs/E23943_01/dcommon/css/darbbook.css
error inserting: /u01/files/docs/E11882_01/dcommon/css/darbbook.css
loaded in filestorage: 84658 hsecs, cpu: 20264 hsecs
error inserting: /u01/files/docs/E23943_01/dcommon/css/darbbook.css
error inserting: /u01/files/docs/E11882_01/dcommon/css/darbbook.css
loaded in filestorage_dedup: 114029 hsecs, cpu: 32013 hsecs
```

This proves that deduplication is costing you more and more time whenever you insert more SecureFile lobs into the same table when you are loading mostly unique documents.

33 E23943_01.zip, complete 11.1.1.6 Fusion Middleware documentation

```
SQL> SELECT ulb.table_name, ulb.column_name, ulb.deduplication,
  2         SUM(ust.bytes)/1024/1024 as size_in_mb
  3    FROM user_lobs     ulb,
  4         user_segments ust
  5   WHERE ust.segment_name = ulb.segment_name
  6*  GROUP BY ulb.table_name, ulb.column_name, ulb.deduplication
SQL> /

TABLE_NAME                COLUMN_NAME               DEDUPLICATION   SIZE_IN_MB
------------------------- ------------------------- --------------- ----------
FILESTORAGE_DEDUP         BFILEDATA                 LOB               7948.125
FILESTORAGE               BFILEDATA                 NO                7935.125

2 rows selected.
```

Why does this query suggest that the deduplicated table is using more storage !? I cannot find a reason for now?

This makes the deduplication feature a non viable feature for very large SecureFile setups with little duplicates and a high number of inserts. If you are capable of partitioning (or subpartitioning) the data into small sets or if have many duplicate documents the feature can be usable.

Epilogue

SecureFile LOBs turned out to have a little more administration overhead than was suggested by documentation, caveats the space management background process did have a serious impact for the customer. Custom preallocation of extents for the LOB segments seems mandatory for larger systems. Also lots of patches appear that are performance related, if your company is not really savvy on patching this might turn into a problem.

The licensable features compression and deduplication can be useful depending on the specific content you are storing, you cannot make any generic statements about the benefits they can bring you.

And as always, you really have to test as much as possible besides monitoring your production setup.

Since I wasn't the only person working on the Secure File LOBs, I want to express my gratitude to the team I worked with while designing, implementing and maintaining the WebCenter Content solution. The solution as it is running at the customer today, is a team effort. Everyone in the team had its own role for optimizing certain areas of the solution. I want to thank Wijbrand Pauw, Jeroen Evers, Corné van den Beemd, Ludi Wang, Nagaraj Krishnappa, Sandeep Sharma, Bernhard de Cock Buning, Jan Janssen and Fabian Scherpenzeel for their efforts while working with me on different occasions. Also I want to thank Doug Burns for convincing me to sent in an abstract for Hotsos symposium 2013.