Internal Workflow of OpenLSD

Author: Frédéric Brégier under LGPL

BASIC WORKFLOW OF A DOCUMENT IMPORT INSIDE OPENLSD 2
BASIC WORKFLOW OF A DOCUMENT RETRIEVE FROM OPENLSD 4
BASIC WORKFLOW OF A DOCUMENT DELETE FROM OPENLSD 4
THE OPHANDLER IS USING THE FOLLOWING WORKFLOW: 5
DISASTER RECOVERY WITH ML SUPPORT 6
OTHER FUNCTIONS 6

Website: http://openlsd.free.fr/en/OpenLSD.html

Basic workflow of a document import inside OpenLSD

1) The document comes with its relevant business index information.

2) Optionally, if CheckSimilar option is used (specific implementation only usable on the OpenLSD Server, so not in Net mode), it first tests if the document exists already in the Legacy, based on the size, the MD5 and then a binary compare. If it exists, the document is rejected. If not, it goes to the next step.
This option is only functional in not crypted Legacy. This function is only functional where imports are done from the final OpenLSD Legacy server.

3) Once a document is identified from the business index and the Legacy is known, the document business index is tested for existence. If it exists already, the import of this document is rejected. Then the next operation is to take one reference in the database according to the Legacy, the size of the document and the available storages. It gives an LSD (Legacy Storage Document) unique index. At this step, only the size and the reference of the document is stored, but not its MD5 key and its date of valid import. This operation is quite sensible since it should ensure unique index and be efficient to allow thousands of requests. That's why these operations should take place in a PL procedure in database. Two kinds of procedures exist:
    a. Simple index: It takes the Legacy index, the size of the document and returns a new created LSD index from the database where this document can fit in the given Storage (enough space in the corresponding storage accordingly to the previous size of the storage, the global size allowed in each storage of this Legacy and the size of this new document)..
    b. Block indexes: It takes the Legacy index, the sum of sizes of a block of documents (multiple documents) and returns a list of new indexes in one particular Storage (where all new documents can fit inside this storage). This operation is faster than the previous one since one lock/unlock operation is done for multiple indexes instead of one lock/unlock for each index.

    Note that sizes of documents are rounded to the closest value to the block size of the underlying filesystem (generally 4KB but can be changed in the code) in the Storage counter (not on the document itself). This is intend to take into account the real occupation by the document on the filesystem in order to prevent error when writing a new document. However, you should always keep some extra space from the Legacy/Storage size since this implementation cannot take into account the size taken by extra information like directories themselves.

4) Once the LSD index is known and on which Legacy it must be imported, the import can start using one of the following methods:
    a. Net Import: the documents are not on the same Server than the one hosting the OpenLSD Server. The full document is passed to the OpenLSD Server as a Put or PutNoAck operation. PutNoAck is more efficient since each packet is sent without waiting acknowledge of the OpenLSD Server, except the last packet, at the price eventually of more memory consumption. Put version will send only one packet at a time, waiting acknowledge of this one before sending the next one, at the price of longer delay since each packet has to be validated. Each block is about 16K.
    b. FileImport: the documents are on the same Server than the one hosting the OpenLSD Server. The reference (path) only of the document is send to the OpenLSD Server as a PutFile operation, so giving only one message by file.

5) For each of those methods, three kinds of import procedure can be used:
   a. If documents are grouped in one file containing all references and indexes, the import can be done by Block (optimized version).
   b. If documents are not grouped in one file containing all references (one index file by document), the import can be done one by one.
   c. If the documents come in one particular directory which is pooled by an OpenLSD daemon, the import can be done by one of the previous kinds (block or one by one) from this directory.

6) The document is send (using the Net or File import version) to the OpenLSD Server.

7) In case of bad answer, the protocol try to redo if possible this step:
   a. In case the OpenLSD Server is down, the import is in failed status.
   b. In case the Legacy is not available (should be a temporary situation), the import is retried.
   c. In case a file already exists (and shouldn't so it is a bad file), the bad file is deleted and the import is retried.
   d. In case an error occurs during import (from OpenLSD Server like disk full), the partial file is deleted and the import is retried.
   e. In case the error comes from the Client part, the import is retried.
   f. In case the length of the imported file is wrong, the file is deleted and the import is retried.
   g. In case the file length is OK, but if the MD5 key is not filled (error from OpenLSD Server but not severe), the client ask for the MD5 key. If it can't get this MD5 key, the file is deleted and the import is retried.

   In any case of those errors, only 3 attempts will be done to try to import the document. If all attempts failed, the import falls in a failed status.
   If the import is OK (from the first attempt or the following), the import is in success status.

8) The OpenLSD Server stores the file accordingly to the LSD indexes and the Legacy implementation (filesystem, cryptographic or not). When a document is inserted or deleted, all upward directories of the given Legacy are updated accordingly to the current date so as to improve consistency check by only checking new documents (see consistency check).
9) If the import is in failed status, the reference in the database is deleted (in fact, marked as no more used since it is quicker, and the given Storage has its size decrease of the given document size).
10) If the import is in success status, the MD5 key and the date of import are updated in the database reference.

11) In case of Multiple Legacies support (ML version of OpenLSD), each time an import is in success status, the client creates the corresponding entries in the Operation table. The table stores the operations to do on document across all OpenLSD Servers implementing Legacies in ML behaviour. In this case, a reference is stored for the already imported document from the source OpenLSD Server, and others similar references are stored to point to other OpenLSD Server for the same Legacy (replication across n servers means n entries in the Operation table where 1 is for the source and 'n-1' are for the destination Servers).
    The new optimized version of ML support creates by default the necessary entries during the step 3 but in no-action mode. They are updated according to the action to do in the step 11.

Basic workflow of a document retrieve from OpenLSD

1) The user gives a business index.
2) The program transforms this business index in LSD internal index.
3) The program finds one OpenLSD Server implementing the given Legacy (L part of the index) from the database.

4) The program connects to the OpenLSD Server and sends one of the following operations:
    a. Get: symmetric operation of Put, it sends the original file block by block and the OpenLSD Server is waiting for each block that the previous one is acknowledge by the client. Each block is about 16K.
    b. GetNoAck: symmetric operation of PutNoAck, it sends the original file block by block without waiting for each block to be acknowledged, except the last one. Each block is about 16K.
    c. GetCopy (Info message): this operation is somehow symmetric to the PutFile operation as it writes the original file into a copy in the "out" path of the Legacy (so stored in the same server than OpenLSD Server) and returns the location (path) in the message as an answer.

5) The client is responsible to reassemble the blocks in order to recreate the original file (except in GetCopy where the file is integrally created by the OpenLSD Server itself).

Basic workflow of a document delete from OpenLSD

1) The program gets a business index.
2) The program transforms this business index in LSD internal index and gets also the corresponding MD5 key.
3) The program finds all OpenLSD Servers implementing the given Legacy (L part of the index) from the database.

4) The program connects to one OpenLSD Server and sends a Delete operation including the LSD index and the MD5 key and the final passkey (authentication).
5) If the passkey is correct, the OpenLSD Server checks the MD5 key with the stored file. If the MD5 are the same, then the file is really deleted and the server returns OK. If not, the file is not deleted and the server returns KO.

6) In case of OK answer, the client deletes the reference in the database (clearing the virtual space in the corresponding Storage).

7) In case of ML support, the client also stores references in the Operation table of Delete operation across all others OpenLSD Servers participating to this Legacy.

The OpHandler is using the following workflow:

1) The OpHandler Server is pooling the Operation table for work to be done.
2) Once he gets some works (ops change of status from ToDo to ToSched), it delegates those ops to worker threads. There are two types of ops:
    a. The master op which changes from ToDo status to ToSched status when it is scheduled. It is relative to the source of the original operation (import or delete). In case of import, it will be used as source.
    b. The slave ops which change from StatusError to StatusBeeing. They are the dependant operations to be done (destination of import or delete remote operations).

3) Each worker thread run one op according to the code of the operation:
    a. In case of Delete: the document is deleted from the destination OpenLSD Server.
    b. In case of Import: the document will be imported from the master op's OpenLSD Server into the slave op's OpenLSD Server.
        i. It first tests the non-existence of the file in the target (slave op). If it exists, the op is considered as done (StatusDone).
        ii. It then tests the existence of the file in the source (master op). If it does not exist, the op is marked as StatusError.
        iii. Finally it sends a request of GetNoAck to the source and a PutNoAck to the target, acting as a proxy would do.

4) When an operation is not OK (no import or no delete), the slave op is marked as StatusError.
5) When an operation is OK (import or delete), the slave op is marked as StatusDone.
6) If one op at least is in StatusError, the master op is marked as ToDo again.

7) If all relative operations to the same master op are OK, all are deleted from the Operation table.

In case the OpHandler has the option '-exportop' specified, the OpHandler stores in successive files all the valid references of the operations (import or delete) in order to allow replication of SQL orders in case one wants to have replicated database using an applicative way (redo of insert or delete SQL orders on replicated database).
The format of the file is "idip;op;lid;sid;did" where idip is the id of one ML LSD Server, op is 2 for insert, 3 for delete. A post action needs to be done to get also the MD5, the size and the business index of each file in order to get the full information to import in the replicated database. A framework is given to do this job with the LSDOpHandlerExport class.
It produces a new file containing "idip;op;lid;sid;did;size;md5;business-index" which can be sent to the host with the replicated database (referenced by idip) and then running with a program like LSDOpHandlerImport class example. Optimization can be achieved by replacing those functions by PL/SQL procedures or functions since they only concern database information.

However, other options can be used to replicate such database information, for instance using proprietary functionality of the database software itself.

Disaster recovery with ML support

In case of disaster in one copy of one Legacy, one can force the update by running LSDInitOpFromDB. This function takes all current references in the database and adds them in the Op table in order to allow the resynchronisation. This function exists in two models and both in two versions:

1) LSDInitOpFromDB which takes all documents from all Storages of one given Legacy
2) LSDInitOpFromDBFromStorage which takes all documents from one specific Storage of one given Legacy

   The usage of one or the other depends one the status of the disaster. If only one Storage is crashed, then it allows to only resynchronizing the specific Storage. This version is probably the most current one to be used since each Storage should be in a different Physical Storage where a disaster can occur.

1) Each version can use a full java method without PL/SQL procedures, i.e. LSDInitOpFromDB and LSDInitOpFromDBFromStorage. Those versions take longer time (about 2.5 times longer).
2) Each version can use a method with PL/SQL procedures, i.e. LSDInitOpFromDBPL and LSDInitOpFromDBPLFromStorage. Those versions are faster (about 2.5 times faster).

The OpHandler can be running during those functions are called, enabling concurrent activity with the forced update. However, one could stop the insertion and deletion while those functions are running in order to have a clean status from the source.

Those functions can be called either on primary Legacy or on secondary Legacies, so as to enable disaster recovery both on primary and secondary.

Other functions

One function is still not implemented but could be done easily, it is the reorganisation function based on the move operation. The move operation is the ability to move one particular document from one position (LSD) to another one (L'S'D'). The source Legacy can be different or equal to the target Legacy. In case of different Legacies, it could be preferred to use an import then delete operations instead of the move function. This reorganisation function would be useful for instance to reorganize storage in order to minimize the Storage numbers and so the number of filesystems attached (1 by Storage). It is relevant for instance when one Legacy will have no more new documents but some of them will be (or already have been) deleted. In this situation, the real necessary number of Storage can dramatically be reduced and using this function will enable to release that Storage back for other uses.