OpenLSD Multiple Legacies Support

Author: Frédéric Brégier under LGPL

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

Multiple Legacies: Logic and Concept

Multiple legacies is the ability for OpenLSD to maintain a mirror of the documents inserted in a Legacy across several (1 to n) servers that implement this Legacy.
One can see a Legacy as a virtual storage. Each virtual storage can have 1 to n implementations on separate servers such that it increases the security of the documents archived in this Legacy.

Considering huge amount of files does not allow making save on tape, so this kind of mirror should be considering as a real security for archive, and probably the only one.

Considering also an organization that has huge network area, this kind of mirror can be used according to localization of each user. For instance, imagine one OpenLSD Server is in the US, another one in Europe, another one in Asia and a fourth one in Africa. Therefore users from Asia should access first to the Asia instance in order to allow fast answer without using intercontinental network links. All of these suppose that all mirrors get the same document archived.

Three ways can be used to have a mirror of the storages:

1) Using a physical mirror from the storage system; this option is easy to implement but could be expensive.
   Its advantage is that it simplifies the handling of the mirror since it is done at the physical level.
   Among its disadvantages, the price could be a problem since most of the time the solution will imply either a Fiber link between the two sites to maintain a good latency and bandwidth for the SAN network, or at least a good TCP/IP link with a real good latency and bandwidth. Another problem is that any bad action from administrator can have an immediate action on the copy (you know the Murphy's Law, one wants to clean something because there is a problem and he cleans in fact a subdirectory containing files, and with the mirror, this delete is done immediately on the second site with no return).
   However it should be considered before to switch to another solution since it is quite simple and efficient.
2) Using an application mirror from the application level; this option needs more work on the application side but could be a good solution to know what is replicated and what is not.
   Its advantage is that it can handle specificity from the application point of view and it relies only on a simple network link.
   However, it implies to integrate the replication system inside the application logic, which could be difficult if not ready.
   You have to double check what's happen when the network link is down or the second server is down and to know what the application needs to do in such a situation. Most of the time, the application will store some zip files (or equivalent) that includes the new files to be ready to be send

to the second server when ready. Also you have to double check about the delete process on both sides.

3) Using the OpenLSD mirror from the OpenLSD level; this option is ready for production in OpenLSD but needs some attention to be sure of your needs.
   Its advantage is it does the job for you and it relies on a simple network link. It already takes care about deletion, network or server down status.
   You need to use the new ML interface instead of the standard one from OpenLSD. Also there are some production tips to know.

The main idea is that each time an import or a delete is done, it is done on a "main" server (most of the time, it should be the one that is closest to the database) and then it stores in the database the actions to do on the others servers that implement the same Legacy.

The replication is therefore asynchronous and starts after a successful action is done. For instance, the replication of one import is done after the first import is done and ok, the same for the deletion.
The asynchronous scheme relies on the database persistence. It stores the actions that are still to do. For a specific document, when all relative actions are done and ok, the relative entries in the database are deleted.

The ML support can be used even after the production was started without this option, and the reverse is also possible, so you can go from or to ML for one Legacy as you want. Although it should not be a good idea to go for instance from no ML to ML and then go back to no ML and again to ML support, since each time you go in the ML support you will have to synchronized the legacy servers.

There is several kind of check in OpenLSD: files from database point of view, database from files point of view and all of them can be done on each component of one ML. There is also a specific function that enables to resynchronized if necessary one or more component of one ML. For instance this function can be used to start a ML instance after a production starts without ML support or to resynchronized to site where one had a problem (like storage failure).

One can use also this ML support not only for security reason but also for efficiency since one can implement web services (even import) using the closest OpenLSD Server as a component for one ML.

The database is unique since this is the kernel point of the OpenLSD implementation to ensure efficiency and security. However, one should take care about the replication of this database since it is not done using the ML support. The reason is that this database could imply business tables that are not related to OpenLSD, so the impossibility for this framework to take care of this replication. Considering very large network, the replication should be done using a master slaves plan, even if the master is changing from time to time

(for instance considering the open hours across the world). Depending one the database software used, several options can be done. Also, an application replication schema can be used where the application takes care of the database replication by assembling SQL orders in one file and pushing it on other sites (that is the option we take for our production, but other options can be taken).

Once the database is replicated, the security is completed. All accesses can be done from everywhere (at least in reading mode) to OpenLSD Servers and also to database schema.

Multiple Legacies: Implementation

The implementation uses the LSDOP table in the LSD database. This table registers operations like import, delete for one LSD index with a status and an IpPort target (OpenLSD Server from the component of this ML). The status says if it is done or in error or in wait.

For an import, the document is first normally imported in the local OpenLSD Server with usual way. Then, when it is imported correctly, the Op table is filled with the LSD index for all IpPort target existing for this Legacy. The local one (where the import took place) is marked as done.
For a delete, the document is first deleted normally from the local OpenLSD Server with the usual way. When correctly deleted, the OpTable is filled with the LSD index for all IpPort target as for import, the local one marked as done.

The Op Handler is running outside of OpenLSD Server and can be running anywhere you want; however it should be closest to the database for efficiency. Every second (can be changed), the Op Handler looks at the Op Table to find some work to do:

- For a delete, it sends the order of deletion of the document points by LSD index and its MD5. When done, the status in the database is changed.
- For an import, it reads from one OpenLSD Server in done status and writes to one OpenLSD Server in waiting status, almost like a proxy. When done, the status in the database is changed for the written one.

Once all operations are done for one LSD index, if they are all OK, the Op Handler removes those operations from the Op Table.
The Op Handler looks at the database once and then wait that all operations are done (in ok or error status) and then it looks again at the database. So its handling is by step and no more than 190 items at a time (PL/SQL limitation on VARCHAR).
The Op Handler is able to produce log files with trace of operations. Those traces allow replicating SQL orders as insert or deleting in external copy of database. For instance, LSDOpHandlerExport allows transforming those files in complete files with business information, enabling the use of LSDOpHandlerImport to import those data in copy of database.

One point is to be cleared however: you must use the same kind of Legacy for all components of one Legacy in ML. That is to say if this Legacy is not using crypto, no components of this Legacy in ML will use the crypto. If this Legacy is using a crypto, every components of this Legacy in ML must use the same crypto key. The reason is for the delete support: the delete operation is permitted if the MD5 is correct. But if you use different behaviors for one LSD index, how can you have one unique MD5 for several crypto key for instance? You can’t, so the reason to use the same crypto key for one Legacy in Multiple Legacies support. It does not mean all of your legacies should use the same key or crypto mode, it is just the components of one specific Legacy at a time.

If Legacy 1 uses crypto key 1, all OpenLSD Servers implementing Legacy 1 should use the same crypto key for the Legacy 1. If Legacy 2 uses no crypto key and is implementing on the same OpenLSD Servers, then all Legacy 2 will have no crypto key.

One ML is created by making several Ip Port entries in the IpPort Table that implement one service attached to one Legacy. So adding the ML support for one existing Legacy is done by adding IpPort entries for the same Services which implement one Legacy.

Multiple Legacies: Production

To enter in production, you have to follow the following steps.

1) Ensure you have a correct standard OpenLSD in function. See the Howto from OpenLSD standard.
2) You then create the new openlsd xml configuration file and its relative legacy xml configuration files for the new OpenLSD Server that will implement some Legacies as part of a ML mode. Most of them should be exactly the same than the previous ones except eventually the port and the base path.
3) Then you execute **`openlsd.multiple.admin.LSDInitDBMLFromConfigFiles`** giving a database configuration file, the previous openlsd configuration file and the hostname (DNS or IP) of the target server. It would create the entries in IpPort for the Services relatives to the Legacies defined in the openlsd configuration file.
4) Make sure the new OpenLSD Server is fully configured (same openlsd configuration files, checking access and shutdown, …).
5) You can now starts the Op Handler from **`openlsd.multiple.ophandler.LSDOpHandler`** giving the database configuration file, the number of threads during its actions, the stop file (when this file exists, the Op Handler stops once in stable status) and the time (in ms) between two scans of the database when the previous step is over. It should start immediately to synchronize the OpenLSD Servers. If nothing is in the Op table, try to import one file and see if everything seems ok.
6) If the first OpenLSD Server was up and includes some files before the second one (or more) comes into the ML support, you have to synchronize all copies either in concurrent mode (slower since each update is done with synchronization) or in no concurrency mode (no import or delete are done during the procedure is running). You can use either **`openlsd.multiple.admin.LSDInitOpFromDB`** or **`openlsd.multiple.admin.LSDInitOpFromDBForStorage`**. Two other version exist and use PL/SQL procedures to enable 2.5 faster updates using **`openlsd.multiple.admin.LSDInitOpFromDBPL`** or **`openlsd.multiple.admin.LSDInitOpFromDBPLForStorage`**. It will populate the Op table from the existing documents from the given source OpenLSD Server, so the Op Handler will take those to synchronize all OpenLSD Servers.
7) You can check the consistency using **`openlsd.multiple.admin.LSDCheckInDBThreadedML`**. It only checks LSD consistency with database, but can only correct LSD storage since another ML can have the missing files. So no action is taken into the database. However, it will say if anything goes wrong too from the database. A specific option ('**`-opfix`**') allows to use the OpHandler to correct the ML copies (import from a valid source when a document is missing, delete the file (if using MD5 checking)).

8) You can also check the consistency using **openlsd.multiple.admin.LSDCheckInDBThreadedDualLimitML.** It has the same effect than the previous one except that it will use two dates to limit the search (files tested will be those created between these two dates). This version enables to stop during a short time only all operations of import or delete (around 1 minute before and after the start of this process).

In production, it can be sometimes useful to stop the Op Handler. In this case, just create the "stop file" and it should stop (or either use the HTTP Administration interface as for the OpenLSD Server). It can be restart later on. Be careful that never more than one Op Handler is started or the behavior will be unknown.

When a break for whatever reason arrives on the production (disk crash, network link break, …), you can still reuse one of the functions from step 6 to resynchronized in whatever order and way the Legacy from this ML.
Most of the tools from OpenLSD standard mode could be used in the ML mode but sometimes they need to specify on which server you want to act, so the existence of some specific implementation that just reuse the same codes but specifying which server (address and port) to use.

Multiple Legacies: Perspectives

Here are some ideas that we could implement later on:

- Now, when you import or delete or access a document, it is done according to the parameter that gets one Ip Port from the table of services. However one could want to access to the closest Legacy component implementing one Legacy in ML. This can be done using an algorithm that check the local IP address and getting the closest IP address that fits some rules and where the service is opened.
- Or another way could be to try from one OpenLSD Server (let say the first one) and if not ready try the next one.
- One could also want to use something like a load balancer between ML repositories. Although I don't think it could be a good idea since they should be distant so as to enforce the security aspect of the physical storages.
- We now can make a file containing database orders (insert, update, delete) such that the replication from an application point of view should be easier. This file could be generated for each step of the Op Handler.
- We can improve the number of functions from standard OpenLSD to ML support.
- We could improve production approach by including a test to see when we launch the Op Handler if this is the only one running.