Synchronous Multi-master in the M-Vault Directory Server

Damy Mahl
VP Engineering
Isode Limited
damy.mahl@isode.com

1. Introduction

LDAP servers commonly implement multi-master using an asynchronous eventual convergence model where changes are propagated across the network of servers and any conflicting changes resolved automatically, if possible.

The specific requirements of Isode's customers were that changes be reflected in the entire replication network immediately and without delay, and so a multi-master model where changes are replicated synchronously was chosen. The benefits to the approach are twofold; that changes to the data at any node are immediately available to all clients connected to the service and that conflicting changes cannot be applied to the data during normal operation.

In this paper we provide an overview of the synchronous multi-master approach, covering the pros and cons of the design and also on future directions for improvement of the solution.

2. Requirements on Replication

Replication is performed to provide service resilience (no single point of failure) and accessibility (bringing the data closer to the users of it). There are broader properties that can make a particular replication technology appropriate to a given directory service, and these include:

- Throughput – how many changes the service can handle.
- Data consistency – how consistent the view of the data is across the directory service (the C in the classical ACID properties of database theory).

These two properties tend to conflict and resulting trade offs have to be made. Next we provide an overview of the common approaches to directory data replication, together with discussion of their advantages and disadvantages.

3. Single-master

The original replication model as defined in X.500 was the Directory Information Shadowing Protocol (DISP). To date this is the only model for directory replication defined in a standard, perhaps providing some indication of the complex relationship between requirements and available technology in this space. DISP specifies a single-master approach where a single server accepts writes to the directory data and these are distributed to a number of read-only shadow servers.

Changes can be pushed as incremental updates or as a total update (for initialization or in the case of loss of synchronization). From a logical perspective the directory data consists of a single well ordered sequence of changes, and this means that there is always a single consistent view of the data at any one point in time. Given a snapshot of the data, moving forward to a snapshot at some later time can be generated by applying the sequence of changes since the time of the original snapshot. Thus synchronization of servers is simple and well defined and will always lead to the same view of the data in all servers in the replication network.

So the DISP transfer of changes from master to shadow ensures that the data in the shadows is fully consistent with that held in the master. Thus, it may be implied that the *whole* system has that same level of consistency of data view. However, DISP provides no guarantee that changes are replicated to shadow servers before the update operation completes. Consider then the following scenario:

1. Directory client modifies a user password (in the master directory) and is notified of success.
2. Directory client then reads back the password (to verify the change) but connects to a shadow directory server.
3. If this read happens before shadowing the update is completed, the client gets back the wrong answer. The data is not consistent across the directory service.

It can thus be seen that ACID characteristics do not always apply to the directory service as a whole. This can be a significant operational concern, although in many deployments and situations the replication delays are quite acceptable. When sensitive data is changed, it may be important that it is changed in all locations without delay. Later, we will discuss approaches that can be used to address this.

Aside from ACID concerns the main problem with single-master is the implied single point of failure. This can be addressed to an extent by including hot failovers of the master, the downside being that manual intervention is generally needed to promote failover system to the active state in the event of failure (the problem being the difficulty in distinguishing network and server failures).

4. Multi-master

The alternative to the single-master approach, where updates are applied to a specific directory server, is multi-master, which allows changes to be applied to any of a group of directory servers and for any changes to be propagated automatically to the other members of the group.

There are a number of scenarios where multi-master is preferable, including:

- Where it is important to always allow updates and where failover in the event of directory server failure should be automatic. (Single master fail-over may need operator intervention).
- To support a clustered service using a peer to peer service, where each node may make directory updates. A good example of this in Isode's customer is a clustered Isode M-Link XMPP service. In this architecture, it makes sense for updates to be applied to the local M-Vault directory server.
- Where it is important to be able to make updates to the same part of the directory from different locations, when there may be network partitioning between the organizations.

In comparison with single-master, multi-master is a complex problem that has various solutions. There are two broad approaches (eventual convergence and synchronous) and these are discussed below.

4.1. Multi-master by Eventual Convergence

Key research on distributed directories was done at Xerox in the 1980s with Clearinghouse and Grapevine. These systems had a multi-master model where changes are made to a local directory server, and then changes are reconciled subsequently between the servers. Most multi-master directories take this "Eventual Convergence" approach.

Eventual convergence is a practical approach that works well in many situations, as changes to directories tend to be isolated and to not conflict. Eventual convergence is, however, not ACID. The key downside is that conflicting changes can be made at different servers (e.g., a single valued attribute can be modified in different ways by two different clients connecting to two different directory servers). This means that the directory is inconsistent for a while. Then change reconciliation will need to resolve the conflict, which it will do by picking one of the changes. This will resolve the inconsistency, but in a way that means that the change made by one of the clients is reversed so that a client thinks it made a change, which ends up not being made, strongly violating the durability part of ACID. For some uses of directory, this behaviour is highly undesirable.

Another issue with an asynchronous approach such as this is the possibility of synchronization latency. Due to the asynchronous approach changes made on one server are only reflected on other servers at some later point in time (consider again the password change scenario highlighted earlier). The flipside to this is that the asynchronous approach allows for high throughput of operation, as servers processing write operations do not have to wait for replication to complete before notifying the client that the write operation has 'completed'.

4.2. Synchronous Multi-master

In adding multi-master capability to M-Vault, we looked at how we could provide multi-master with ACID characteristics. Network and computer performance and resilience have massively increased since many of the original multi-master directories were developed. This allows us to adopt an approach that would not have been viable in the 1980s.

In a multi-master directory setup, every server participating as a master is directly connected to each of the other master servers. The update process consists of a two-phase commit, with operations following the sequence below:

1. A directory client requests an update.
2. The directory server obtains appropriate locks. For modifying an entry, an update lock on the entry will be obtained. Other updates such as renames will need different locks. First a local lock will be obtained and then a lock on every other master.
3. The change is made on each of the remote masters and the local master. Locks are released at each server on successful commit.
4. The client is told that the update has been made.

Important aspects of this with respect to ACID properties are:

1. The servers are locked in a way that ensures that the same change is made on each server, so the problems arising with eventual convergence are avoided. There is no need for change resolution.
2. All servers are updated before the client is notified of successful completion. So once the client has been told that a change is made, any subsequent directory reads will reflect the change.

Note that this approach does not mean that each directory will process changes in the same order. The important point is that change processing will only happen in a way that does not impact the eventual state, as entry locking prevents conflicting changes from taking place concurrently.

One major advantage of the synchronous approach is that it lends itself to the implementation of LDAP transaction processing, something that would not be possible in an eventual convergence model.

5. The M-Vault Approach in Detail

Central to the M-Vault approach is the locking model used to prevent conflicting changes from entering the system. Two basic lock types are used:

1. Entry lock (exclusive). This protects against concurrent changes to the contents of the entry and against concurrent attempts to remove the entry.
2. Subordinate lock (shared or exclusive). This is used to protect against removal of the parent of the target of the operation, i.e. to prevent an entry from being removed while an operation is adding an entry beneath it. Specifically, this lock protects against transitions from non-leaf to leaf state, and vice versa, where that transition might result in a conflict.

For example, if adding an entry "cn=Will Havelock,o=Widget Inc.,c=GB", the following locks would be taken:

1. Exclusive lock on "cn=Will Havelock,o=Widget Inc.,c=GB".
2. Shared lock on the subordinates of "o=Widget Inc.,c=GB".

A concurrent attempt to remove "o=Widget Inc.,c=GB" would take the following locks:

1. Exclusive lock on "o=Widget Inc.,c=GB".
2. Exclusive lock on the subordinates of "o=Widget Inc.,c=GB".

The second of the locks in each of the sets above conflict, so preventing the changes from being processed concurrently. Where a lock is rejected due to conflict the issuing server retries a set number of times before rejecting the operation and sending a 'busy' error to the client.

In general the preference is to keep the logic and protocol as simple as possible, avoiding the complexity of full blown consensus algorithms such as Paxos or Raft, and the specific aim of this locking scheme is to minimize the number of locks required to achieve the required level of consistency. As well as making the logic more understandable this reduces the load imposed on the internal locking subsystem in addition to reducing the likelihood of lock conflict.

The locking information associated with a particular change consists of the set of locks required, and also a change identifier, which is used to associate the lock set with the invoking change operation. As well as simply identifying a lock set, the change identifier is also used to decide which operation wins in the event of a lock conflict. The change identifier consists of three elements:

1. A timestamp (at one second resolution).
2. A sequence number (reset after every second within the generating server).
3. An integer server ID, where each server's ID is unique.

A simple algorithm ranks change identifiers (and by extension lock sets) by comparing each of these in order. The change identifier that orders earlier (earlier timestamp, lower sequence number or server ID) is deemed to have a higher priority. This prioritization is used to determine which lock wins in a conflict case.

If a lock is rejected due to lock conflict then the server processing the client operation retries. The timestamp and sequence number components of the change identifier mean that it naturally increases in priority over time. This means that at subsequent retries a lock set is more likely to win in a conflict situation, thereby helping to reduce the possibility of lock starvation.

6. Improvements and Enhancements

The current release of M-Vault implements the first version of the multi-master protocol, and this has a number of areas that are planned for improvement in the next (R17.0) release of the server.

6.1. Leadership Election

In the current version locks are sent to every other server in the multi-master network, and this adds to the overhead of update operations and thus has a negative impact on the overall throughput. The next version of the M-Vault multi-master protocol will include leader election, where the leader will handle (amongst other things) all lock operations. This will have the following beneficial effects:

- Higher throughput, due to the reduced wait for locks to complete at every server.
- Improved bandwidth usage.
- Reduced overall CPU load, as lock computations don't take place on all servers.

6.2. Handling Network Partition

At present M-Vault's multi-master system operates on a quorum basis. A quorum is a strict majority (e.g., two out of three; three out of five; three out of four, etc.). Consider a multi-master directory with three servers, and network partition splits into one and two (or one server fails). The group of two is a quorum and updates may be made. The server on its own is less than a quorum and so updates are not made while it is separate.

This model allows updates in the event of server failure, provided that there are at least three servers. Because changes will only be made by a “majority” group of servers, inconsistent changes will not arise.

The downside to the quorum approach is that should the multi-master servers be partitioned, perhaps due to a network split, then the two sides of the partition cannot operate independently. We plan to address this by implementing a protocol for two-way synchronization, including conflict detection and automatic resolution.

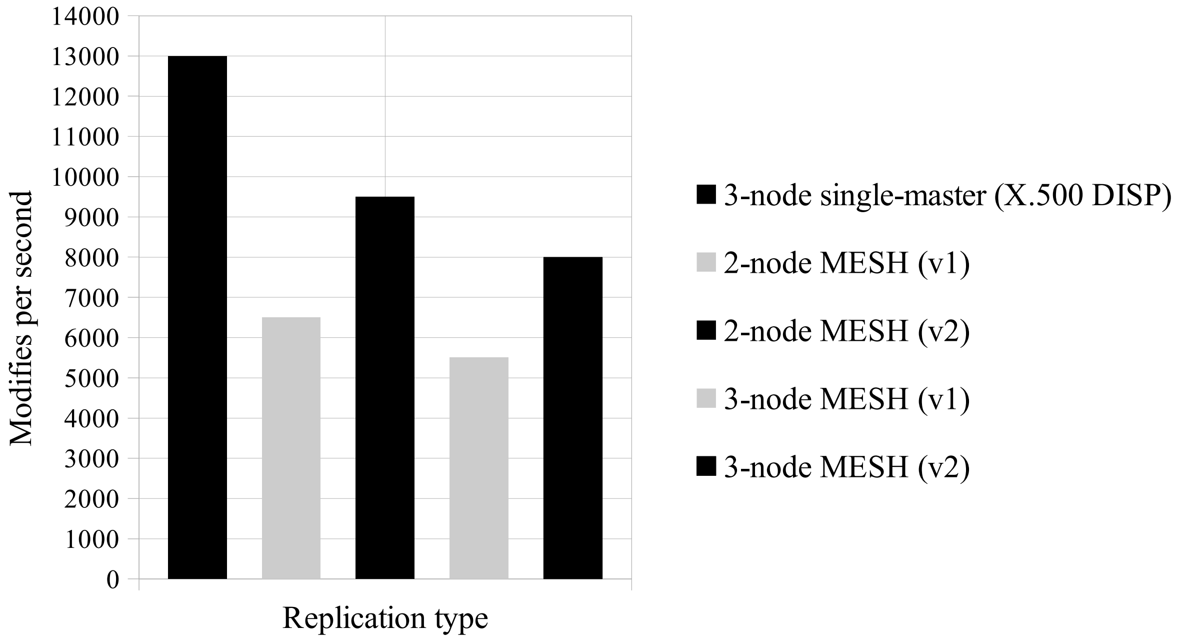
7. Performance

Some comparative tests have been made using groups of two and three node replication networks, testing M-Vault with:

- DISP (X.500 shadowing).
- MESH (v1 - current protocol version) – all servers process lock requests.
- MESH (v2 - next protocol version) – locking handled by elected leader.

Low end systems were used in this test:

- Quad-Core Opteron 8356.
- 7200-rpm disks - EXT-3 filesystem.



The throughput of the asynchronous X.500 DISP replication case is clearly best. Note though that, under heavy load, synchronization latencies of up to three seconds were observed.

The disparity between DISP and MESH, the multi-master protocol, is mitigated to a significant degree in the upcoming version of the protocol, which shows a close to 50% improvement in throughput over the current version.

8. Summary

Synchronous multi-master was preferred in M-Vault over the more common eventual convergence approach in order to maintain ACID properties in the directory data and always provide a consistent view of the information. The trade off with the use of a synchronous approach is the loss of throughput in comparison with asynchronous methods (though we expect to close the gap in the upcoming R17.0) release. In cases where the synchronous approach is not appropriate, for example where the directory service is widely and geographically distributed, X.500 DISP is offered as an appropriate solution.

9. Further Reading

1. "The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment" Derek C. Oppen and Yogen K. Dalal - http://bitsavers.informatik.uni-stuttgart.de/pdf/xerox/parc/techReports/OPD-T8103_The_Clearinghouse.pdf
2. "Experience with Grapevine: The Growth of a Distributed System" Michael D. Schroeder, Andrew D. Birrel, and Roger M. Needham, Xerox Palo Alto Research Center - https://birrell.org/andrew/papers/ExperienceWithGrapevine.pdf
3. "Rethinking Eventual Consistency" Philip A. Bernstein and Sudipto Das, Microsoft Research - http://research.microsoft.com/pubs/192621/sigtt611-bernstein.pdf
4. "In Search of an Understandable Consensus Algorithm" Diego Ongaro and John Ousterhout, Stanford University - http://ramcloud.stanford.edu/raft.pdf