Abstract

This paper defines a multi-master, incremental replication protocol using the LDAP protocol [LDAPv3]. This protocol uses and builds upon previous LDAP support protocols, namely the changelog [change] and LDIF [LDIF] protocols. It defines the use of two types of transport protocols for replication data, and specifies the schema that must be supported by a server that wishes to participate in replication activities using this protocol.

Introduction

LDAP is increasing in popularity as a generalized query, access, and retrieval protocol for directory information. Data replication is key to effectively distributing and sharing such information. Therefore, it becomes important to create a replication protocol for use specifically with LDAP to ensure that heterogeneous directory servers can reliably exchange information. This document defines a multi-master, incremental replication protocol for use with LDAP. It does not specifically address the needs of single-master (i.e., master-slave) systems, though this document could be used as the basis for such a scheme. In addition, it defines how to use that replication protocol over two transport mechanisms: standard email and LDAP. The new replication protocol requires new data to be entered into the directory for use with this protocol. Therefore, we must define new schema to hold that information. Also, the data must be transmitted in a specific format; we will use the proposed LDIF format [LDIF] for doing this.

Protocol Behavior

2.1 A glossary of replication terminology
There are 6 axes along which replication functionality can be provided. These are:

- single-master vs. multi-master
- full vs partial
- whole vs fractional
- transactional vs loosely consistent
- complete vs. incremental
- synchronous vs. asynchronous

Each of these terms is described below.

A single-master (also known as master-slave) replication model assumes that only one server (the master) allows write access to the replicated entries. Changes flow from the master server to all of the replicas. A multi-master replication model assumes that entries can be written on multiple servers. Changes must then propagate from all masters to every replica, which requires additional work for conflict resolution. Here, an update conflict is defined as updating the same data within a given same replication interval (i.e., not necessarily at exactly the same time).

Full replication is where every object in a database or DSA is copied to the replica. Partial replication is where some subset of the objects is copied.

Whole and fractional replication refer to the attributes transmitted during replication. If all attributes of the replicated objects are copied, this is referred to as whole replication. If only a subset of the attributes are copied, this is referred to as fractional replication.

Transactional replication requires that the replica gets and commits all changes between its copy of the data and the master’s copy of the data before the client is notified that the change was successful. Note that ‘commit’ is used in the general sense to define the action of writing changes to a data store and verifying that those changes were written successfully. Specifically, it does NOT imply two-phased commit as used in databases. Loosely consistent means that there are times when the written server has data that the replicas do not, from the client’s point of view. Note also that a general replication topology may well have a mix of links that are transactional and loosely consistent.

Complete replication requires the replicating server to send a complete copy of itself to the replica every time it replicates. Incremental replication allows the replicating server to only send that data which has changed.

Synchronous replication updates the replica as soon as the source data is changed. Asynchronous replication updates the replica some time after the source data has been modified.

2.2 Single-master versus multi-master replication

This section provides some additional general information that will help lay the groundwork for understanding replication.
Replication technology enables the placement of copied and/or shared data at different locations distributed throughout an organization. This is usually done for two reasons: (1) providing 'fast' local access by eliminating long-distance connections between users and database servers, and/or (2) providing corporate-wide high-availability access for critical applications, making them more resistant to single system failures.

Replication topologies determine what is to be replicated as well as what can be updated when and where. Replication policies define how replication is done as well as how update conflicts are resolved. Both of these are orthogonal to this specification, and so will only be mentioned for completeness of understanding in this document.

Single-master replication designates one, and only one, copy of the data as being the 'master' or authoritative source of data. All updates are done to the master, and the master is responsible for ensuring that all replicas contain the same data as it does. Single-master schemes require the definition of both replication topology as well as policy (or policies), although since there is by definition no update conflict, these topologies and policies are in general simpler than those of their multi-master counterparts.

Multi-master replication allows updates to different servers, where each of the updated servers allow the same naming contexts to be writable. An update conflict occurs when the same data is updated on multiple servers within the same replication interval (i.e., not necessarily at the same time). This means that copies can be temporarily out of sync with each other. This is acceptable, as long as over time the data converges to the same values at all sites. This requires a rich definition of topology and policy or policies.

2.3 The basics of multi-master, incremental replication

This specification is aimed primarily at supporting multi-master, incremental, loosely consistent, asynchronous replication. This specification could also be used to support single-master replication, if one treats only one of the servers in a replication agreement as writable. Some of the information required in this specification (i.e., conflict resolution) is not needed for the single-master case. However, the ability to define what has been changed and to ensure that those changes are propagated throughout the system is common to both single- and multi-master replication. Therefore, this specification will address only multi-master
replication.

To implement multi-master, incremental, loosely consistent, asynchronous replication, each server which wishes to master data MUST have the facilities necessary to track changes to the replicate data. In addition, each master server MUST have the ability to transmit those changes to other replicas, and MUST have techniques to implement conflict detection and resolution. The replication protocol enables servers to transmit changes over several transport protocols. This document also provides algorithms for detecting and resolving conflicts.

2.4 The Naming Context (NC)

The Directory Information Base (DIB) is the collection of information about objects stored in the directory and their relationships. The DIB may be organized as a hierarchy (or tree), where objects higher in the hierarchy provide naming resolution for their subordinate objects. This tree, called the Directory Information Tree (DIT), provides the basis for using names to query, access, and retrieve information. The DIT can in turn be comprised of a set of subtrees.

The basic unit of replication is the NC. A Naming Context consists of a non-leaf node (called the root of the naming context) and some subset of its descendants subject to the following restriction: a descendant cannot be part of a naming context unless all of its ancestors which are descendants of the naming context root are in the naming context (e.g. an NC is a complete subtree and cannot have any holes).

Each DSA will have one or more naming contexts. These naming contexts will be defined and available in the Configuration container pointed to by the root DSE of the server. The requisite schema are defined in section 3.

To replicate a given naming context, the only requirement is that the two servers agree on the contents of every schema entry needed to define all the objects in the naming context. The reconciliation of these entries is beyond the scope of this protocol.

2.4.1 Tracking changes to an NC

Borrowing from the ChangeLog draft [change], each change to a replicated NC is logged in its own entry in the changeLog container. This entry has object class ‘changeLogEntry’ and holds the trace of the change, in LDIF format. For more details on the format, see [change]. However, the current ChangeLog draft is designed to provide single master replication. To provide multi-master, incremental replication, much more information needs to be kept.

In addition to the information required by the ChangeLog draft, servers MUST also keep track of the following information and MUST write it to the changeLog entry:
- a version number for each property of every entry
- a timestamp for the time each property is changed,
- the attributes that were changed in this particular entry
- the object classes of this particular entry
- the naming context in which a given entry resides
- a unique identifier for each entry, which is NOT the DN or RDN of the entry
In addition, servers MUST also keep track of the following information and MAY write it to the changeLog entry:

- a unique identifier for each entry’s parent, which is NOT the DN or RDN of the parent, when the operation performed on this entry is a modifyDN.

2.4.2 Discussion of the required new changeLog information

The version number and timestamp are required for conflict resolution in multi-master replication.

The attribute and object class tracking are useful for directory synchronization with special-purpose directories. The actual changes themselves are stored in a single binary blob in the changeLog entry. This allows special-purpose directories (such as mail server directories) to extract only the changes they need.

The NC is required for conflict resolution in multi-master replication. The NC in which a given entry resides allows efficient replication of a given naming context. While this may in principle be derivable from the DN of the changed entry, adding this information allows much easier retrieval of the appropriate entries.

The unique identifier is required to handle modifyDN conflicts correctly.

In addition, the server MUST write the entry’s parentUniqueID to the changeLog entry during tracking of a modifyDN operation. This is required by the reconciliation algorithms defined below.

The new attributes are defined in section 3.

2.5 Defining the replication topology

Each server replicating a given set of naming contexts needs to have information about that naming context, including information on how to replicate it. However, this information is orthogonal to the replication protocol and as such is beyond the scope of this document.

2.6 Replication conflict resolution policies

This section will describe a simple, yet powerful, policy for reconciling conflicts in a multi-master replication environment. This policy is one implementation of resolving conflicts. However, some applications might require more granular control, where different policies are used for different parts of the DIT or even at different times under different circumstances. However, a detailed analysis of different replication policies is beyond the scope of this document.

2.6.1 Using ChangeLog to implement a replication conflict resolution policy

In a multi-master environment, conflict resolution between incompatible updates is crucial. Since each change listed in the ChangeLog includes the version number of the attribute, every attribute received in a replication update is reconciled with the local version of the attribute in the following way:

A. If the version numbers are different, the higher version is favored
If the version numbers are the same, the version with the more recent time stamp is favored.

If both the version and time-stamp match, the values themselves are compared and the one with the lowest value is favored. This guarantees that the system will quiesce consistently.

If all three of these match, the values are identical.

If an object is deleted, a server implementing this replication protocol MUST keep a 'tombstone' of the deleted object. This is essentially a copy of the deleted object that can be used to restore it. This document does not specify the length of time that such tombstones must be kept (this is part of the replication policy that is implemented in a set of replicated servers). When an object is deleted and there are replication changes that affect that object, there are some special rules that must be applied.

Deletions are allowed only on objects which have no children. If a deletion is received for an object that has a child, the reconciliation is to simply ignore the deletion. The server MAY flag this as an error and issue an error to the administrator, who is then responsible for correcting the problem.

If an incoming replication change is to create a new object under an already deleted object, then the tombstones of all the ancestors of the already deleted object are reanimated and the new object is inserted in the correct place. This reanimation must minimally restore the RDN and object class attributes of the ancestor.

A modifyDN operation is not considered, for purposes of replication, to be a combination of a delete and an add operation unless such an operation would move the object to a new naming context.

In the case where the operation does not cross NC boundaries, it is a single operation that essentially modifies an entry’s parentUniqueID. Since this attribute is treated as an attribute of the entry itself, the standard reconciliation logic applies.

In the case where the operation does cross the NC boundaries, it must be treated as a delete and add combination. A server conforming to this specification will in addition treat the delete and add combination as an atomic operation.

In addition, a modifyDN or modifyRDN operation may cause two objects to have the same DN. In that case, the replication system MUST algorithmically change the RDN of one or both of the objects. The algorithmically generated RDN is propagated so that the system will still reach a consistent state. The easiest way to guarantee a non-conflicting RDN is to use the object’s UID as the new RDN.

2.6.2 Loading data

In a replicated environment, the problem of loading multiple remote systems in a coordinated fashion is much more complex. There are three different methods to instantiate data at participating locations in a replicated environment: (1) pre-instantiation, (2) on-line instantiation, and (3) off-line instantiation.

Pre-instantiation copies all data to all locations before beginning the actual configuration of the replication environment.
This guarantees identical data copies before allowing the replication process to begin. This requires that no replication activity take place against the data until all locations are up and configured.

On-line instantiation is used when there is an initial location that has already been populated with data that will be used to replicate data to a set of remote locations. Note that in a general replication topology, there may be several authoritative servers that master data to different sets of replicas. The replication policy that is used in this case should ensure that there are no duplicate replica sets for the initial loading of data. This has the advantage of guaranteeing that all locations have the same data and gives the administrator a single point of control. However, there can be significant delay while data are copied over the network.

In off-line instantiation, an initial location is configured as a master. Replication agreements are then set up between it and other locations to replicate data at a later time. The initial location stores changes destined for future locations and pushes them to the other locations as they come on line. This enables other locations to be loaded with data and synchronized with the initial location when they need new data.

On-line instantiation is enabled by the use of the FullReplicaControl control, discussed in section 4.1.

3: Schema

This section defines new attributes used in this protocol. Object classes and attributes which are not defined in this document can be found in [LSPA] or in [change].

3.1 Changes to the ChangeLog document

As noted above, multi-master replication requires a substantial number of changes to the changeLog document. Here are the new object class and attributes.

Note that commonName, namingContexts, and description are all defined in other documents.

3.1.1 Changes to changeLogEntry

( 2.16.840.1.113730.3.2.1
  NAME 'changeLogEntry'
  SUP 'top'
  STRUCTURAL
  MUST (
    ChangeNumber $ targetDN $ changeType $ changes $ changedAttribute $ entryObjectClass $ namingContext $ uniqueIdentifier )
  MAY  (  
    ParentUniqueIdentifier $ NewRDN $ deleteOldRDN $ newSuperior
  )
)

3.1.2 Changed attributes

( 2.16.840.1.113730.3.1.5
NAME 'changeNumber'
DESC 'a 64 bit number which uniquely identifies a change made to a Directory entry'
SYNTAX 'Integer'

3.1.3 New attributes

(1.2.840.113556.1.4.AAO
 NAME changedAttribute
 DESC 'OID of changed attribute'
 SYNTAX 'DirectoryString'
)

( 1.2.840.113556.1.4.AAR
 NAME 'entryObjectClass'
 DESC 'object class this entry participates in'
 SYNTAX 'DirectoryString'
)

(1.2.840.113556.1.4.AAS
 NAME 'parentUniqueIdentifier'
 DESC 'Unique identifier of the entry’s parent'
 SYNTAX 'DirectoryString'
)

3.4 Changes to the LDIF document

To allow incremental efficient multi-master replication, we required two pieces of information for each attribute to be transmitted that must appear on a per-attribute basis; version number and timestamp. This should be transmitted in the LDIF format as qualifiers on the appropriate attribute: i.e. ‘commonName;2,19970308133106Z: Fred Foobar’. The version number is always the second to last qualifier, the timestamp is always the last qualifier. Note that this information is formatted this way for transmission purposes only.

4: LDAP transport

One of the two methods used to transport replication data is by using the LDAP protocol itself. The target server sets up an ordinary LDAP session with the source server, binding to the source DSA as the target server (remember that the root naming context has been replicated everywhere and so every server participating in a given replication topology knows about all the other servers) and issues a search with the new ‘replicate’ extended control. The target server will specify the changeLog container as the base of the search, and will use a filter that states that all records with changeNumber greater than the current high update number, that reside in one of the replicated naming contexts, will be given back. The source server MUST then order the results in such a way so that when they are applied to the replica in that order, the replica will be synced with the source server at the time that the replication snapshot was taken. This ordering of the changes is imperative. One possible way to provide such an ordering would be to sort the results on changeNumber. There will be a number of LDAP implementations which may not wish to provide a general sort facility for search results; however, a conformant implementation of the replicate control MUST order the results into a correct order.

Once the target starts receiving entries, it then applies each of the changeLogEntries to its own database, in the same order in which the
entries were sorted, incrementing its highUpdateNumber attribute for that server appropriately. If the source server has indicated that it has more entries, the target server can then reissue the search with the new highUpdateNumber. In an environment with a rapidly changing directory, the source directory may at its discretion return a maximum highUpdateNumber indicating the highest number used by the server at the start of the session. The target server should then use that number as an additional term on the filter on subsequent search requests to allow a 'snapshot' of the data to be replicated. Otherwise, the target server might never close the connection to the source server, which would impact source server performance and available bandwidth.

The replicate control is included in the searchRequest and searchResultDone messages as part of the controls field of the LDAPMessage, as defined in Section 4.1.12 of [LDAPv3]. The structure of this control is as follows:

\[
\text{replicateControl ::= SEQUENCE} \\
\text{\hspace{1cm}controlType 1.2.840.113556.1.4.617} \\
\text{\hspace{1cm}criticality BOOLEAN DEFAULT TRUE} \\
\text{\hspace{1cm}controlValue INTEGER (1..2^{64}-1)} \\
\]

The replicateControl controlValue is used by the source server to return a maximum highUpdateNumber if it wishes to allow the target server to take a snapshot of the replication data.

4.1: Full updates

If the target server wishes to retrieve a full listing of the current contents of the DSA, it must issue a fullReplicaControl. This control is used on a search operation, just like replicateControl. The structure of the control is as follows:

\[
\text{fullReplicaControl ::= SEQUENCE} \\
\text{\hspace{1cm}ControlType 1.2.840.113556.1.4.618} \\
\text{\hspace{1cm}Criticality BOOLEAN DEFAULT TRUE} \\
\text{\hspace{1cm}ControlValue INTEGER (1..2^{64}-1)} \\
\]

The fullReplicaControl controlValue is used by the source server on the first response message to indicate how many entries it will be sending. If the stream of entries is interrupted, the target server MUST flush the entries received so far and issue the fullReplicaControl control again. When the source server sends the last entry, it must set the ControlValue on the message to the correct highUpdateNumber so that subsequent replication operations can retrieve only the data that has been changed.

5: Mail transport

The other method of transporting replication data is by using an email protocol. In this case, the target server mails the search command with the replicate extended control or the fullReplicaControl control to the source server, and then the source server mails the results of the replication command back to the target server, in LDIF format as modified above [LDIF]. When the target server receives the changes, it processes
them as appropriate. The actual mail transport protocol used is not
covered in this document; it needs to be established as a bilateral
agreement between the two servers. The security on this transaction is
enabled by the security of the underlying mail protocol chosen.

6: Security Considerations

Replication requires secure connections and the ability to secure the
change information stored in the directory. Securing the change
information is covered in [change]. Standard LDAP security should be
applied to the LDAP transmission of data. Standard mail security should
be applied to the mail transmission of data. The information necessary
to secure these connections will be stored as part of the URLs defining
the connection points.

7: References

[change] Good, Gordon, Definition of an Object Class to Hold LDAP Change
Records, Internet Draft, July 1997. Available as draft-ietf-ldap-
changelog-01.txt

[LDIF] Good, Gordon, The LDAP Data Interchange Format (LDIF) -
draft-ietf-ldap-ldif-02.txt.

[LSPA] Wahl, M. A Summary of the Pilot X.500 Schema for use in LDAPv3,
Internet Draft, March 1997. Available as draft-ietf-ldap-schema-
pilot-00.txt

8: Author’s addresses

Chris Weider
Cweider@microsoft.com
1 Microsoft Way
Redmond, WA 98052
+1-206-703-2947

John Strassner
Johns@cisco.com
170 West Tasman Drive
San Jose, CA 95134
+1-408-527-1069