Selective LDAP Multi-Master Replication *

Thomas Bauereiss[1], Stefan Gohmann[2], Dieter Hutter[1], and Alexander Kläser[2]

[1] German Research Center for Artificial Intelligence, Bibliothekstr. 1, D-28359 Bremen, Germany, `{thomas.bauereiss|hutter}@dfki.de`
[2] Univention GmbH, Mary-Somerville-Straße 1, D-28359 Bremen, Germany, `{klaeser|gohmann}@univention.de`

Abstract: LDAP directory services are widely used to store and manage information about the assets of organisations and to ease the administration of IT infrastructure. With the popularity of cloud computing many companies start to distribute their computational needs in mixed-cloud infrastructures. However, distributing an LDAP directory including sensitive information to partially trusted cloud servers would constitute a major security risk. In this paper, we describe an LDAP replication mechanism that allows for a fine-grained selection of parts of an LDAP directory tree that are replicated to another server using content-based filters, while maintaining the availability and performance advantages of a full multi-master replication. We discuss sufficient conditions on replication topology and admissible operations such that the replication mechanism provides eventual consistency of selectively replicated data.

1 Introduction

Directory services with LDAP interface are widely used to store and manage information about infrastructure and assets of organisations. Multi-master replication (MMR) mechanisms are readily available for directory services, providing high availability and eventual consistency of directory data on different servers via optimistic replication. However, existing MMR mechanisms provide only limited options for configuring a master to replicate only selected parts of the LDAP directory tree. Besides full replication, typically only the division of the LDAP directory into disjoint subtrees is supported. However, with the popularity of cloud computing many companies start to distribute their computational needs in mixed-cloud infrastructures. Groupware, typically supporting the collaboration between employees, are deployed in the cloud to realise a highest level of availability while data and programs constituting the assets of a company should be kept on in-house installations. To ease administration there is a need for uniform mechanisms to administer the in-house installations as well as the various installations in the cloud. A naive approach would distribute a common LDAP directory to all the individual installations causing a major security hole as this common LDAP directory contains also the information needed

*This work was supported by the German Federal Ministry of Economics and Technology under grant/project SATCLOUD.

to access in-house installations and thus to access the assets of the company. Consequently, the security of these assets would depend on the security of the LDAP directory stored in the cloud.

Therefore, we aim to develop more flexible mechanisms for selective multi-master replication, giving organisations the ability to select which parts of the directory to replicate to which (cloud) server, while maintaining the advantages of full MMR. In this paper, we describe such a mechanism where each LDAP master has an associated view on the LDAP directory defined in terms of a set of LDAP filter expressions. We discuss sufficient conditions on replication topology and admissible operations such that the replication system provides eventual consistency of replicated data.

The rest of this paper is structured as follows. In Section 2, we describe an example application scenario for our replication mechanism. In Section 3 we give a formal model of LDAP directories, operations and filters. We describe the replication mechanism in Section 4 and define its consistency guarantees in Section 5. Section 6 describes related work in the area of optimistic replication. Section 7 concludes the paper with a summary and directions for future work.

2 Example scenario

As a typical application scenario for infrastructure and identity management based on LDAP directories, consider a large organisation with several local branches in different cities. There is a central LDAP master server at the organisation's headquarters that hosts the full LDAP directory tree, so that the top level management has an overview over the organisation. In addition, every local branch office has an LDAP master of its own that only receives and maintains information about its own employees and its used infrastructure.

Such a scenario is typically implemented by modelling the local branches as organisational units (OUs) and configuring the branch masters to replicate their respective OU information. This is possible with existing implementations of LDAP directory servers, e.g. Active Directory, provided that the directory is partitioned into disjoint subtrees.

Now consider the situation where two branches start a joint project and require a groupware server where the employees working on the project can organise appointments and share documents. In order to avoid having to operate additional infrastructure, they decide to set up the groupware server at a cloud provider. There are several possible approaches:

- Set up a groupware server in the cloud without connection to the enterprise's identity management. The disadvantage is that manual creation of accounts is required and there is no password synchronisation.
- Set up an identity provider for single sign-on at a central enterprise server and configure the groupware server to make use of it. However, only server software that supports the chosen framework for single sign-on can be used in this case.
- Set up an LDAP directory server on the groupware machine and configure it to replicate identity data for employees working on the project. Replicating the whole LDAP directory of the enterprise or even of both branches is not acceptable for

performance and security reasons. Replicating only the data of employees in the project using a content-based filter is only supported in read-only (slave) mode in existing LDAP implementations. However, this implies a limited availability if the connection between the groupware server in the cloud and the LDAP server in the enterprise network is temporarily broken. If a user then wants to change his address information, for example, or wants to define a text for an absence notification on the groupware server, then this would fail. If write access is required during a login process, for example due to a mandatory password change, then even the login fails.

In this paper, we describe a mechanism for selective multi-master replication that allows one to specify which parts of an LDAP directory to replicate based on its content. For example, the organisation described above could specify the employees that work on the joint project and should get access to the groupware server by assigning a corresponding value to the "project" attribute of their LDAP entries. The replication component on the groupware server can then be configured to replicate those and only those LDAP entries. Any application with LDAP support can then be configured to read and possibly write this data. Modifications, for example the change of a password by a user, are replicated back to those LDAP masters that can see that part of the LDAP directory tree, e.g. the central LDAP master and the local master of the user's branch. We envision that such a replication mechanism can give organisations more flexibility and control over their replication setup according to their organisational structure and security requirements.

3 Formalisation of LDAP structures

3.1 LDAP, Schemata and Filters

In the spirit of [WL02] we rephrase the notions of LDAP schemata, directories (instances), and filters in a formal way, in order to be able to reason about the consistency of replication later on. We start with an LDAP schema specifying the ontology of an LDAP tree.

Definition 1. An *LDAP schema* $\mathcal{L}$ is a tuple $\langle \mathcal{C}, \mathcal{A}, \mathcal{T}, \mathsf{req}, \mathsf{opt}, \mathsf{type} \rangle$ where $\mathcal{C}$ is a set of classes; $\mathcal{A}$ is a set of attributes for the classes with $\{oc, dn\} \subseteq \mathcal{A}$, where oc and dn denote the "object class" and "distinguished name" of entries, respectively; $\mathcal{T}$ is a set of types for the attributes; $\mathsf{type} : \mathcal{A} \to \mathcal{T}$ maps each attribute to its type; $\mathsf{req} : \mathcal{C} \to 2^{\mathcal{A}}$ maps each class to its required attributes such that $\forall C \in \mathcal{C}.\ \{oc, dn\} \subseteq \mathsf{req}(C)$; and $\mathsf{opt} : \mathcal{C} \to 2^{\mathcal{A}}$ maps each class to its optional attributes such that $\forall C \in \mathcal{C}.\ \mathsf{req}(C) \cap \mathsf{opt}(C) = \emptyset$.

Definition 2. An *LDAP* $\mathsf{L} = \langle N_{\mathsf{L}}, E_{\mathsf{L}} \rangle$ is a forest where each node $N \in N_{\mathsf{L}}$ is labelled by its class C_N and a set I_N of pairs (a, v) where $a \in \mathcal{A}$ and v is a value of type $\mathsf{type}(a)$. Each edge in E_{L} is labelled by a DN pair (a, v) such that each node $N \in N_{\mathsf{L}}$ is uniquely determined by the sequence $(a_0, v_0) \dots (a_n, v_n)$ of labels on the path from its root to itself.

Definition 3. An LDAP L *complies* to an LDAP schema $\mathcal{L} = \langle \mathcal{C}, \mathcal{A}, \mathcal{T}, \mathsf{req}, \mathsf{opt}, \mathsf{type} \rangle$, or is an $\mathcal{L}$-LDAP for short, iff for all $N \in N_{\mathsf{L}}$ it holds that

(i) $\forall a \in \mathsf{req}(C_N).\ \exists v \in \mathsf{type}(a).\ (a, v) \in I_N$, and

(ii) $\forall (a, v) \in I_N.\ a \in \mathsf{req} \cup \mathsf{opt} \wedge v \in \mathsf{type}(a)$.

We use filters to define views on a particular LDAP. In particular, each filter consists of a Boolean expression controlling which parts of the LDAP are visible in the corresponding view. These Boolean expressions operate on the existence or value of selected attributes entries and combine them with the help of Boolean junctions to complex expressions. The following definition specifies the language for building such Boolean expressions.

Definition 4. Let $\mathcal{L} = \langle \mathcal{C}, \mathcal{A}, \mathcal{T}, \mathsf{req}, \mathsf{opt}, \mathsf{type} \rangle$ be an LDAP schema. The set $\mathsf{Expr}_{\mathcal{L}}$ of $\mathcal{L}$-LDAP expressions is the smallest set satisfying

$$\begin{aligned} (a\ =\ *) &\in \mathsf{Expr}_{\mathcal{L}} && \text{if } a \in \mathcal{A}, \\ (a\ op\ t) &\in \mathsf{Expr}_{\mathcal{L}} && \text{if } a \in \mathcal{A}, t \in \mathsf{type}(a) \wedge op \in \{=, <, >, \leq, \geq\}, \\ F_1\ R\ F_2 &\in \mathsf{Expr}_{\mathcal{L}} && \text{if } F_1, F_2 \in \mathsf{Expr}_{\mathcal{L}} \wedge R \in \{\wedge, \vee, \rightarrow\}, \text{ and} \\ \neg F &\in \mathsf{Expr}_{\mathcal{L}} && \text{if } F \in \mathsf{Expr}_{\mathcal{L}}. \end{aligned}$$

Given the values of some attributes A as a set I of attribute-value pairs, an evaluation function $eval_I \colon \mathsf{Expr}_{\mathcal{L}} \rightarrow \mathsf{bool}$ is defined as usually.

In the following we present the formal definition of an LDAP filter. LDAP filters are the main building blocks to define views on LDAPs and thus to determine those parts of an LDAP to be replicated and maintained in a restricted master.

Definition 5. Let $\mathcal{L} = \langle \mathcal{C}, \mathcal{A}, \mathcal{T}, \mathsf{req}, \mathsf{opt}, \mathsf{type} \rangle$ be an LDAP schema. An *$\mathcal{L}$-filter* is a tuple $\langle p, s, \mathsf{A}, expr \rangle$ such that p is a sequence of DN-pairs, $s \in \{\mathsf{base}, \mathsf{one}, \mathsf{sub}\}$, $expr \in \mathsf{Expr}_{\mathcal{L}}$, all attributes occurring in $expr$ are contained in A, and for all $C \in \mathcal{C}.\ \mathsf{req}(C) \subseteq \mathsf{A}$. Given an $\mathcal{L}$-LDAP L and a $\mathcal{L}$-filter F then a node $N \in N_{\mathsf{L}}$ is *in the focus* of F iff

1. $p = \mathsf{Path}(N)$ and $s = \mathsf{base}$,
2. $\exists (a, v).\ p \circ (a, v) = \mathsf{Path}(N)$ and $s = \mathsf{one}$, or
3. $\exists p'.\ p \circ p' = \mathsf{Path}(N)$ and $s = \mathsf{sub}$.

A node $N \in N_{\mathsf{L}}$ is *accessible* wrt. F iff it is in the focus of F and $\mathsf{eval}_{I_N}(expr) = true$.

The application $F(N)$ of a filter to a node N is A if N is accessible and the empty set else.

Definition 6. An *$\mathcal{L}$-view* is a set $\mathcal{V}$ of $\mathcal{L}$-filters. A node $N \in N_{\mathsf{L}}$ is *in the focus* of $\mathcal{V}$ iff it is in the focus of some $F \in \mathcal{V}$. It is *accessible* in $\mathcal{V}$ iff it is accessible wrt. some $F \in \mathcal{V}$. The view $\mathcal{V}(N)$ of a node N is the union of all applications of filters in $\mathcal{V}$ to N.

Using LDAP filters to define the visibility of an LDAP in external masters means that changing the attributes of an object may also change its visibility and thus its accessibility in the cloud. This results in the problem to evaluate a filter in the cloud but having only restricted access to attributes of an object. A simple approach is to require that attributes used in filter expressions have to be a subset of the filter attributes. A more sophisticated approach would be to simplify the filters with respect to the attributes that are not replicated to the cloud. In general however, this results in filter rules that are individual to each object of the replicated LDAP, which is not feasible in practice.

Definition 7. Let L be an $\mathcal{L}$-LDAP and $\mathcal{V}$ be an $\mathcal{L}$-view. Then, $\mathcal{V}$ *induces* an $\mathcal{L}$-LDAP $\mathcal{V}(\mathsf{L})$ on L by

1. an isomorphism $\zeta\colon N'_{\mathsf{L}} \to N_{\mathcal{V}(\mathsf{L})}$, where $N'_{\mathsf{L}} = \{N \in N_{\mathsf{L}} | N$ is accessible wrt. $\mathcal{F}\}$,
2. there is an edge (a, v) between $\zeta(N), \zeta(N') \in N_{\mathcal{V}(\mathsf{L})}$ iff there is an edge (a, v) between $N, N' \in N_{\mathsf{L}}$, and
3. $C_{\zeta(N)} = C_N$ and $I_{\zeta(N)} = \{(a.v)|(a.v) \in I_N|$ a is accessible in N wrt. $\mathcal{V}\}$ hold.

3.2 Operations

In this section we are concerned with manipulating an LDAP or one of its views. The main question is to find appropriate conditions that allow us to relate modifications of the view on an LDAP to corresponding modifications on the LDAP itself. The main requirements to this setting are 1. that the modification on the global LDAP is uniquely determined by the modification on the view and 2. that each modification of the view that results in a consistent state corresponds to a modification on the global view that also results in a consistent state. In order to make this precise, we introduce the notion of admissibility of operations.

Definition 8. A *basic operation* on an LDAP L is one of the following operations:

1. modify (also add or remove)[1] a possibly multi-valued attribute a in a node $N \in N_{\mathsf{L}}$
2. insert or delete a node in L, or
3. rename a node N in L.

Definition 9. Let L be an $\mathcal{L}$-LDAP and $\mathcal{V}$ be a $\mathcal{L}$-view. A basic operation *op* is *admissible* on $\mathcal{V}(\mathsf{L})$ iff $\mathcal{V}(op(\mathsf{L})) = op(\mathcal{V}(\mathsf{L}))$ holds. A basic operation *op* with $op(L) \neq L$ is *visible* in $\mathcal{V}(\mathsf{L})$ iff $\mathcal{V}(op(\mathsf{L})) \neq \mathcal{V}(\mathsf{L})$ holds, and *invisible* on $\mathcal{V}(\mathsf{L})$ otherwise.

For simplicity, we assume that complex operations are broken up into a set of basic operations, such that each basic operation is either completely visible (i.e. admissible) or completely hidden in a view defined by a view $\mathcal{V}$. In particular, modifications of multiple attributes of an entry are broken up into operations modifying a single attribute each. The filtering of an operation then reduces to a binary decision whether the operation is visible in a view or not, and we avoid having to alter the content of operations when filtering. This will be useful when we define operation-based selective replication in Section 4.

The modification of an attribute a of a node N is admissible on $\mathcal{V}(\mathsf{L})$ if $a \in \mathcal{V}(N)$. The deletion of N is admissible on $\mathcal{V}(\mathsf{L})$ if N is accessible wrt. some $F \in \mathcal{V}$. The insertion of a node in the $\mathcal{V}(\mathsf{L})$ corresponds to the insertion of N in L with the exception that we allow the further insertion of default attributes, so called I_0 for N not accessible to $\mathcal{V}(\mathsf{L})$. In all other attributes N and $\zeta(N)$ coincide. The insertion is admissible (wrt. a preset I_0) if N is in the focus of some $F \in \mathcal{V}$ and $\forall a \in \mathsf{A}_N \setminus DOM(I_0).\ a \in \mathcal{V}(N)$ and there is no

[1] We assume that an attribute does not exist in a node if it has no associated value. Hence, insertion and removal can be regarded as special cases of adding or removing values of an attribute. Also, the replacement of attributes as defined by the LDAP standard can be modelled by removing all attribute values known at the time of submission of the operation, and adding the new values.

other node N' in $\mathcal{V}(\mathsf{L})$ with the same path as N. The renaming of a node N to a path p' is admissible if the insertion of N with its attributes is admissible at path p'.

3.3 LDAP conflicts

Many LDAP operations are commutative, e.g. the modification of different attributes of a node or the insertion of nodes at different paths. In some cases, however, the concurrent submission of operations in a multi-master LDAP system can lead to conflicts. If we assume that operations refer to nodes using a unique identifier, then two concurrent operations are in conflict in the following cases:

- both are modifications of an attribute v of the same node N, and there is a value v that is added by one operation and deleted by the other;
- both are insertions or renamings of nodes at the same path; or
- one is a deletion of a node N and the other refers to N, but is not a deletion.

For these conflicts, we aim to perform immediate automatic resolution in some deterministic way so that the repositories are always in a state that is consistent with schema and application constraints, while at the same time recording conflict so that the conflicts can be properly resolved manually.

In order to detect conflicts, we first have to determine concurrency of updates. For this purpose, LDAP masters propagate basic operations enriched with additional metadata. We denote such an enriched update as $('update', op, m, t, H)$ where *op* is an operation submitted at master m at (local) time t, and H is the set of all updates known to m at the time *op* was submitted.[2] An update $a = ('update', op, m, t, H)$ *happened before* an update $b = ('update', op', m', t', H')$, denoted $a \to b$, iff $a \in H'$. Two updates are concurrent, denoted $a \leftrightarrow b$, iff $a \not\to b \land b \not\to a$.

Conflicts are detected by checking if concurrent updates make conflicting changes to an entry. The typical conflict resolution strategy for the modification of attributes is the "Last Writer Wins" strategy, where operations are ordered using timestamps and the newest operation simply overwrites older, conflicting operations. For naming conflicts, we rename the nodes that are moved or created by operations that are dominated by a conflicting operation according to a deterministic naming scheme. For conflicts where a deleted node is concurrently modified, we can either copy the node to a lost-and-found area of the directory tree, or simply ignore the modification.

Overall, a consequence of deterministic conflict resolution is that all concurrent updates commute. In [SPBZ11] it has been shown that strong convergence for full replication easily follows from the commutativity of operations. With selective replication, however, a master might know only a subset of updates, so if only one of two conflicting updates is visible to a master, it cannot perform conflict resolution on its own. In the case of LDAP replication, this affects naming conflicts, as illustrated by the following example.

Example 1. Consider, for example, the insertion of a node N with path $(ou, sales)$,

[2]In the actual implementation, we use a compact representation for this set, such as version vectors [SS05].

$(cn, john)$ and attribute $(project, A)$ into an LDAP $\mathcal{V}(\mathsf{L})$ with

$$\mathcal{V} = \{\langle (ou, sales), \mathsf{sub}, \{ou, cn, project\}, (project = A)\rangle\}$$

If L already contains a node with the same path and attribute $(project, B)$, then the insertion of N causes a conflict that cannot be seen by a master with restricted view $\mathcal{V}(\mathsf{L})$.

This means that a master with restricted view cannot, in general, exclude the possibility of inter-view conflicts for node insertions or renamings. One approach to solve this problem is to provide restricted masters with additional information about hidden nodes that are in the focus of one of its filters, e.g. by replicating a dummy node for each hidden node to the restricted master.

A more general solution is to inform the restricted master about the result of conflict resolution by replicating an explicit conflict resolution operation. When a master receives an update from another master with restricted view $\mathcal{V}$, and it detects that the update is in conflict with another update that is not admissible for $\mathcal{V}$, then it generates a conflict resolution update that contains an operation op' with the effect of the resolution that can be propagated back to the restricted master.

Consider again the conflict in Example 1. Assuming we resolve naming conflicts by renaming all but one affected nodes to unique and deterministic names, the conflict resolution operation is in this case the renaming of the relative DN part of N to $(cn, john+m+t)$, where m is the identifier of the restricted master and t is the timestamp of the insertion operation of N, and $+$ is a special concatenation symbol that can only be introduced by LDAP masters, not by users submitting operations. Hence, the new name of N is unique and deterministic. The renaming update is then propagated to all masters where N is visible. Masters with insufficient view to resolve the conflict themselves will apply the update and reach a state consistent with full masters, while for masters that have already resolved the conflict themselves, the update will have no further effect, because renaming a node to a name it already has is redundant.

4 Replication mechanism

We now describe in more detail an operation-based replication mechanism that incorporates our considerations from above. First, we introduce some notation for the state of a selective multi-master LDAP system. We denote the *state* of an $\mathcal{L}$-LDAP master m as a tuple $\langle \mathsf{L}, \mathcal{H}, \mathcal{Q}\rangle$ comprising an $\mathcal{L}$-LDAP L, a sequence $\mathcal{H}$ of updates that have been applied already, called the history of the master, and a sequence $\mathcal{Q}$ of updates that have been received by other masters, but not yet applied, called the queue. We assume that the updates in the queue are additionally annotated with the master from which they were received. The state of a master evolves as it processes updates submitted by users or received by other masters. We denote the state of m at step k (i.e. after the k-th update) as $m(k) = \langle \mathsf{L}(k), \mathcal{H}(k), \mathcal{Q}(k)\rangle$, where $m(0) = \langle \emptyset, \emptyset, \emptyset\rangle$, i.e. masters are initially empty.

A selective multi-master system then consists of a set of masters that communicate with each other at possibly irregular intervals. In existing full-replication LDAP systems, eventual delivery of operations is ensured by creating a replication topology in the form of a connected graph, i.e. there is a communication path between any two master servers. In the case of selective replication, we have to additionally take into account the views of the master servers. We have to avoid the loss of information that would occur when all paths between two masters m and m' go through masters m'' with a view that is smaller than both the views of m and m'. In order to guarantee that there is always at least one path without information loss, we require that the topology contains a spanning tree such that views always monotonically increase along a path towards the root:

Definition 10. A *selective multi-master system* $\mathcal{M} = \langle M, (\mathcal{V}_m)_{m \in M}, G \rangle$ consists of

- a set M of LDAP masters, with at least one full master $m_{root} \in M$,
- a family $(\mathcal{V}_m)_{m \in M}$ of $\mathcal{L}$-views $\mathcal{V}_m$ for every master m, with $\bigcup_{m \in M} \mathcal{V}_m \subseteq \mathcal{V}_{root}$,
- a replication topology represented as a connected, directed graph $G = (M, E)$ such that $\forall (m, m') \in E.\ \mathcal{V}_m \subseteq \mathcal{V}_{m'}$ holds and for all master $m \in M$ there is a path from m to m_{root} .

We assume that every master m will always eventually propagate relevant updates to every adjacent master m', i.e. we assume liveness of communication. The propagated updates will then eventually appear in the queue of m' in the correct order, i.e. we assume causal delivery. An update is relevant for $\mathcal{V}_{m'}$ either if it is admissible for $\mathcal{V}_{m'}$ at the time of submission, or if it becomes admissible for $\mathcal{V}_{m'}$ afterwards, because the affected node and attributes have been moved into the view by another operation in the meantime. For example, if a node N is in the focus of a filter $F \in \mathcal{V}_{m'}$ with attribute set A, and an operation changes attributes of N such that the filter expression of F becomes true, then all updates affecting attributes of N in A retroactively become relevant for $\mathcal{V}_{m'}$. Formally, we define the subsequence of updates in a history $\mathcal{H}_m(k)$ that are relevant for a view $\mathcal{V}$ as

$$\begin{aligned}\mathcal{V}(\mathcal{H}_m(k)) = [u \in \mathcal{H}_m(k) \mid & admissible(u, \mathcal{V}(\mathsf{L}_{m_u}(k_u))) \vee \\ & \exists k' \leq k. u \in \mathcal{H}_m(k') \wedge admissible(u, \mathcal{V}(\mathsf{L}_m(k')))]\end{aligned}$$

where m_u is the master where u was submitted and k_u the step of m_u at which it was submitted. Such a history filtering is monotonic in the sense that $u \in \mathcal{V}(\mathcal{H}_m(k'))$ implies $u \in \mathcal{V}(\mathcal{H}_m(k))$ for all $k \geq k'$, i.e. the history filtering only grows with increasing k. We consider two history filterings equivalent, denoted $\mathcal{V}(\mathcal{H}) \equiv \mathcal{V}(\mathcal{H}')$, if both contain the same set of updates, but concurrent updates possibly occur in a different order.

There are two types of local state transitions for a master. Either the state transition is caused by an operation that has been submitted by a user, or it takes an update coming from another master out of its queue and applies it to its LDAP. In the second case, it might also be necessary to generate conflict resolution updates for masters with insufficient view. The effects of the two kinds of state transitions are as follows:

1. If a user submits an operation *op* at m at step k and *op* is admissible for $\mathcal{V}_m(\mathsf{L}_m(k))$, then $\mathsf{L}_m(k+1) = op(\mathsf{L}_m(k))$ and $('update', op, m, t, \mathcal{H}_m(k))$ is appended to the history, where t is a current local timestamp.

2. Otherwise, the master dequeues the first update $u = ('update', op, m', t', \mathcal{H})$ from its queue. If u is already known to m, i.e. $u \in \mathcal{H}_m(k)$, or if u is a conflict resolution

update for a conflict that has already been resolved locally, or if u is not admissible for the view of the master from which it has been received, then the update is ignored and the state remains unchanged for $k+1$. Otherwise, the update is appended to the history and applied to the LDAP, i.e. $\mathsf{L}_m(k+1) = op(\mathsf{L}_m(k))$. If op causes a conflict, the master then determines whether it is necessary to generate a conflict resolution update: If there is a master m' with $\mathcal{V}_{m'} \subseteq \mathcal{V}_m$ and the conflict resolution is visible but op is not admissible on $\mathcal{V}_{m'}(\mathsf{L}_m(k))$, then m generates a conflict resolution update r for u and appends it to its history.

We have now defined both the communication behaviour as well as the local state transitions of masters in a selective multi-master system. In the next section, we discuss the consistency guarantees that such a system provides.

5 Eventual consistency with respect to views

In [SPBZ11], strong eventual consistency (SEC) for full replication is defined in terms of eventual delivery, strong convergence, and termination of operations. In this section, we adapt the definitions of these notions for the case of selective replication. Eventual delivery then means that an update that is submitted at a master m eventually reaches a master m' if and only if it is relevant for $\mathcal{V}_{m'}$. The constraints on the replication topology of a selective multi-master system, combined with liveness, are sufficient to ensure eventual delivery.

Theorem 1. *A selective multi-master system $\mathcal{M}$ provides eventual delivery with respect to views, i.e. if an operation op is submitted at a master m at step k, then the corresponding update $u = ('update', op, m, t, \mathcal{H}_m(k))$ eventually reaches a master m' if and only if it is relevant for $\mathcal{V}_{m'}$:*

$$\exists K_{m'}, K_m > k.\ \forall k_m > K_m, k_{m'} > K_{m'}.\ (u \in \mathcal{V}_{m'}(\mathcal{H}_m(k_m)) \Leftrightarrow u \in \mathcal{H}_{m'}(k_{m'}))$$

Proof. This easily follows from the replication topology, liveness of communication, correct history filtering during communication, and monotonicity of history filtering. $\square$

For strong convergence with respect to views, we require that equivalent knowledge of two masters with respect to a common subview implies equivalent states when filtered for that view:

Definition 11. A selective multi master system provides *strong convergence with respect to views* if for all masters m and m' and for every view $\mathcal{V}$ that is a subview of $\mathcal{V}_m$ and $\mathcal{V}_{m'}$:

$$\forall k, k' : (\mathcal{V}(\mathcal{H}_m(k)) \equiv \mathcal{V}(\mathcal{H}_{m'}(k'))) \Longrightarrow \mathcal{V}(\mathsf{L}_m(k)) \equiv \mathcal{V}(\mathsf{L}_{m'}(k))$$

In order to show that our replication mechanism provides strong convergence, we first show a lemma establishing that for each individual master, applying the updates it knows that are relevant for a view to an empty LDAP results in a state equivalent to the master's actual state filtered for that view.

Lemma 1. *Let $m(k)$ be the state of a master in a selective multi-master system, $\mathcal{V} \subseteq \mathcal{V}_m$ a view, and $\mathsf{L}_{\mathcal{V}(\mathcal{H}_m(k))}$ the LDAP that results from successively applying the operations in $\mathcal{V}(\mathcal{H}_m(k))$ to an empty LDAP. Then $\mathcal{V}(\mathsf{L}_m(k)) = \mathcal{V}(\mathsf{L}_{\mathcal{V}(\mathcal{H}_m(k))})$ holds.*

Proof. By induction on k. The base case $k = 0$ trivially holds, as $m(0)$ is initialised with empty LDAP and history. In the induction step, we perform a case distinction on the type of local state transition from k to $k + 1$.

1. Assume a user submits an operation *op* at step k. If *op* is invisible on $\mathcal{V}(\mathsf{L}_m(k))$, then the filtered history and state remain unchanged, and the conclusion follows from the induction hypothesis. If *op* is admissible on $\mathcal{V}(\mathsf{L}_m(k))$, then $\mathcal{V}(\mathcal{H}_m(k + 1))$ results from appending an update containing *op* at the end of the history, and the conclusion follows from admissibility of *op* and the induction hypothesis. If *op* changes attribute values such that a set A of formerly invisible attributes of the affected node N are now visible according to the filters in $\mathcal{V}$, then *op* is admissible on $\mathcal{V}(\mathsf{L}_m(k+1))$, and $\mathcal{V}(\mathcal{H}_m(k+1))$ results from $\mathcal{V}(\mathcal{H}_m(k))$ by appending an update containing *op* and possibly interleaving a sequence U of updates that have been made admissible by *op*. The updates in U are exactly those that affect the attributes A of N and that are not yet contained in $\mathcal{V}(\mathcal{H}_m(k))$. They are independent from and effectively commute with all operations in the filtered history from the previous step that affect other nodes and attributes, and they happen after or concurrently with updates affecting attributes A of N in $\mathcal{V}(\mathcal{H}_m(k))$ due to causal delivery. Hence, applying U and *op* to $\mathsf{L}_{\mathcal{V}(\mathcal{H}_m(t))}$ leads to a state where the values of the attributes A of N are consistent with $\mathsf{L}_m(k + 1)$, while the consistency of other nodes and attributes visible in $\mathcal{V}$ follows from the induction hypothesis.
2. Assume an update u received from another master is dequeued from $\mathcal{Q}_m(k)$ and applied at step $k+1$. If the update is admissible or invisible on $\mathcal{V}(\mathsf{L}_m(k))$ or changes visibility, then the conclusion follows by the same arguments as in the case of local submission. In addition, however, it is now possible that u is in conflict with an update u' in $\mathcal{H}_m(k)$. If the conflict resolution is visible, but u is not admissible on $\mathcal{V}(\mathsf{L}_m(k))$, then m also generates a conflict resolution update that is admissible on $\mathcal{V}(\mathsf{L}_m(k))$ such that $\mathcal{V}(\mathsf{L}_m(k + 1))$ includes the visible effects of the conflict resolution, and the conclusion again follows as above. □

Theorem 2. *If concurrent operations commute, then a selective multi-master LDAP system $\mathcal{M}$ provides strong convergence.*

Proof. This is a direct consequence of Lemma 1: For any m, m', k, and k', if $\mathcal{V}(\mathcal{H}_m(k)) \equiv \mathcal{V}(\mathcal{H}_{m'}(k'))$ for $\mathcal{V}$ with $\mathcal{V} \subseteq \mathcal{V}_m$ and $\mathcal{V} \subseteq \mathcal{V}_{m'}$, then $\mathsf{L}_{\mathcal{V}(\mathcal{H}_m(k))} = \mathsf{L}_{\mathcal{V}(\mathcal{H}_{m'}(k'))}$ by commutativity of concurrent operations. Hence $\mathcal{V}(\mathsf{L}_m(k)) = \mathcal{V}(\mathsf{L}_{m'}(k'))$ by Lemma 1. □

Overall, our replication mechanism in combination with the restrictions on replication topology and admissibility of operations provides both eventual delivery and strong convergence with respect to views. Since we assume termination of operations, we can say that it indeed provides strong convergence in the sense of [SPBZ11], adapted for selective replication.

6 Related work

There is a large body of related work on replication, both in theory and practice, in various settings and with different performance and consistency guarantees. An overview can be found in [CBPS10]. Multi-master replication is an instance of optimistic replication [SS05], where any replica can accept modification operations without waiting for consensus with other replicas. Modifications are propagated from time to time, detecting and resolving any conflicts due to concurrent conflicting modifications. Existing implementations of LDAP directory servers typically support selective replication, but only in slave mode or with limited options for defining which parts of the directory to replicate. To the best of our knowledge, there is no existing support for selective LDAP multi-master replication that allows to define the visible parts of the directory using content-based filters.

In [SBKH05], an abstract formalism for consistency in replicated systems is presented. Partial replication is discussed based on the assumption that the replicated data is partitioned into a set of disjoint databases, with every master replicating a subset of these databases and every database having a primary master. In this paper, we discuss the concrete case of selective LDAP replication and the possible dependencies and conflicts between views. Our definition of eventual consistency includes a notion of eventual delivery with respect to views, and therefore goes beyond the Mergeability property of [SBKH05].

In [RRT+09] a replication platform is presented where devices can select the items they replicate (out of a set of independent items) using content-based filters, similar to our LDAP filter expressions. Also, the Eventual Filter Consistency property of Cimbiosys is similar to our Eventual delivery with respect to views. However, the paper does not discuss dependencies between items or conflicts between updates.

An interesting recent development are "Conflict-free Replicated Datatypes" (CRDTs) [SPBZ11]. These are data types that satisfy certain sufficient conditions for a given definition of eventual consistency. For example, in the case of operation-based replication, the main condition is that all concurrent operations commute, i.e. there are no conflicts. The authors of [SPBZ11] give several examples of non-trivial CRDTs for data structures such as sets, where conflicts are avoided by designing operations for commutativity with the help of additional metadata. In a sense, our work is both a generalisation of the notion of eventual consistency of [SPBZ11] to the case of partial replication, as well as a specialisation to LDAP directories as the data type.

7 Conclusions

In this paper, we presented a mechanism for selective replication of LDAP directory trees together with sufficient conditions to guarantee eventual consistency of replicated data. We are currently working on a prototype implementation of a replication component using the mechanism described in this paper. It is layered on top of a local LDAP server at each master in the replication topology, and is responsible for the communication between masters, enforcing the constraints described above. This includes checking the admissibility of operations submitted by users, propagating correctly filtered operation histories to connected masters, and ensuring that the replication topology satisfies the conditions of Definition 10. The implementation effort also includes work on practical aspects that we were not able to discuss here due to space constraints, e.g. a garbage collection mechanism that allows masters to purge old updates from their histories.

We will evaluate our prototype by integrating it with the Univention Corporate Server (UCS), which is a Debian-based GNU/Linux distribution that allows administrators to manage infrastructure, services and user accounts using tools based on an underlying LDAP directory. We plan to release our prototype as open-source software so that it can be evaluated and applied by others.

Opportunities for further research include the formal verification of correctness and security properties of our replication mechanism with the help of a theorem prover such as Isabelle/HOL [NPW02] in the spirit of works such as [IROM06]. Research in this direction might also lead to a more general theory for selective optimistic replication with eventual consistency for datatypes other than LDAP directory trees.

References

[CBPS10] Bernadette Charron-Bost, Fernando Pedone, and André Schiper, editors. *Replication - Theory and Practice*, volume 5959 of *LNCS*. Springer, 2010.

[IROM06] Abdessamad Imine, Michaël Rusinowitch, Gérald Oster, and Pascal Molli. Formal design and verification of operational transformation algorithms for copies convergence. *Theoretical Computer Science*, 351(2):167–183, February 2006.

[NPW02] Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic*, volume 2283 of *LNCS*. Springer, 2002.

[RRT$^+$09] Venugopalan Ramasubramanian, Thomas L. Rodeheffer, Douglas B. Terry, Meg Walraed-Sullivan, Ted Wobber, Catherine C. Marshall, and Amin Vahdat. Cimbiosys: a platform for content-based partial replication. In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, NSDI'09, page 261–276, Berkeley, CA, USA, 2009. USENIX Association.

[SBKH05] Marc Shapiro, Karthikeyan Bhargavan, Nishith Krishna, and Teruo Higashino. A Constraint-Based Formalism for Consistency in Replicated Systems. In *Principles of Distributed Systems*, volume 3544 of *LNCS*, page 900. Springer, 2005.

[SPBZ11] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-Free Replicated Data Types. In *Stabilization, Safety, and Security of Distributed Systems*, volume 6976 of *LNCS*, pages 386–400. Springer, 2011.

[SS05] Yasushi Saito and Marc Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81, March 2005.

[WL02] Fang Wei and Georg Lausen. A Formal Analysis of the Lightweight Directory Access Protocol. In *Conceptual Modeling for New Information Systems Technologies*, volume 2465 of *LNCS*, pages 306–319. Springer, 2002.