rfc4533

LDAP Content Synchronization Operation

Ce document définit l'opération lDAP Content Synchronization, ou Sync Operation, qui permet à un client de maintenir une copie synchronisée d'un fragment d'un DIT. Sync Operation est définis comme un jeu de contrôles et d'autres élément du protocole qui étendent l'opération de recherche.

Pendant des années, plusieurs approches de synchronisation de contenu ont été suggérés pour utiliser les services d'annuaire LDAP. Ces approches sont inadéquates pour un ou plusieurs des raisons suivantes :

- Impossibilité de s'assurer d'un niveau de convergence raisonnable
- Impossibilité de détecter que la convergence ne peut pas être achevée (sans rechargement)
- Nécessite que le serveur maintienne un historique des changements passés dans le DIT et/ou les méta-informations
- Nécessite que le serveur maintienne un état de synchronisation sur un base par client ; et/ou
- sont trop bavard

Sync Operation fournis une convergence éventuelle du contenu synchronisé quand cela est possible et, sinon, une notification qu'un rechargement complet est nécessaire. Sync Operation ne nécessite par d'agrément pré-arrangé de synchronisation.

Sync Operation ne nécessite pas que les serveurs maintiennent ou utilisent un historique des changements passés dans le DIT ou de méta-informations. Cependant, les serveurs peuvent maintenir et utiliser des historiques pour réduire le nombre de messages générés et pour réduire leur taille. Comme il n'est pas toujours faisable de maitenir et d'utiliser des historiques, l'opération peut être implémentée en utilisant un approche purement basé sur l'état. Sync Operation permet d'utiliser soit l'approche state-based, soit l'approche history-based pour balancer l'a taille de l'historique et la quantité du trafic. Sync Operation permet également de combiner l'utilisation de ces 2 approches.

Sync Operation ne nécessite par que les serveur maintiennent d'état de synchronisation par client, cependant, les serveurs peuvent maitenir et utiliser un tel état d'information pour réduire le nombre de messages générés et la taille de tels messages.

Un mécanisme de synchronisation peut être considéré trop bavard quand le trafic de synchronisation n'est pas raisonnable. Le trafic de Sync Operation est limité par la taille des mises à jours ou des nouvelles entrées et le nombre d'entrées inchangées dans le contenu. L'opération est conçue pour éviter des échanges complet de contenu, même quand les informations d'historique disponible au serveur est insuffisant pour déterminer l'état du client. L'opération est également conçue pour éviter la transmission d'information d'historique hors contenus, vu que sa taille n'est pas limitée par le contenu et il n'est pas toujours faisable de transmettre de telles informations d'historique dû à des raisons de sécurité.

Utilisation

Sync Operation est conçu pour être utilisé dans les applications qui nécessitent une synchronisation de contenu éventuellement convergent. Une fois l'étape de chaque synchronisation de l'opération terminée, toutes les informations pour construire une copie client synchronisée a été fournie au client ou le client a été notifié qu'un rechargement complet est nécessaire. Excepté pour les incohérence transitoires dus au traitement simultané du serveur, la copie client est un reflet précis du contenu maintenu par le serveur. Les inconsistances transitoires seront résolues par des opérations de synchronisation ultérieurs.

Des utilisations possibles incluent :

- Les applications de services page blanche
- Les moteurs de méta-informations

- Les services de proxy
- La réplication maître-esclave entre des serveurs d'annuaire hétérogènes.

Ce protocole n'est pas conçus pour être utilisé avec des applications nécessitant une consistance de données transactionnelles. Vu que ce protocole transfère toutes les valeurs visibles des entrées appartenant au contenu courant au lieu d'un changement delta, ce protocole n'est pas approprié pour des applications à bande passante limitée ou des déploiements.

Vue d'ensemble

Cette section fournie une vue d'ensemble sur la manière dont Sync Operation peut être utilisé pour maintenir une copie client synchronisée d'un fragment du DIT.

Polling for Changes (refreshOnly)

Pour obtenir sa copie cliente initiale, le client envoie un requête Sync : une requête Search avec le contrôle Sync Request et mode à refreshOnly. Le serveur retourne le contenu correspondant au critère de recherche, Additionnellement, avec chaque entrée retournée, le serveur fournis un contrôle d'état de synchronisation indiquant l'état. Ce contrôle contient l'UUID de l'entrée. Le contenu initial est suivi par un SearchResultDone avec un contrôle Sync Done. Le Sync Done Contrôle fournis un syncCookie, qui représente un état de session.

Pour interroger les mises à jours pour la copie client, le client renouvelle le Sync Operation avec le syncCookie précédemment retourné. Le serveur détermine quel contenu doit être à retourner comme si l'opération était une opération de recherche normale. Cependant, en utilisant le syncCookie, le serveur envoie les copies des entrées qui ont changé avec un Sync State Control indiquant l'état add. Pour chaque entrée changée, tous les attribut (modifiés ou non) appartenant au contenu sont envoyés.

Le serveur peut effectuer une des 2 phases de synchronisations, ou les 2, qui sont distinguées par la manière de synchroniser les entrées supprimées du contenu : Les phases présent et delete. Quand le serveur utilise une seule phase pour l'étape de rafraîchissement, chaque phase est marquée comme terminée par un SearchResultDone avec un Sync Done Control. Une phase present est identifiée par une valeur refreshDeletes à FALSE dans le contrôle Sync Done. Une phase delete est identifiée par une valeur refreshDeletes à TRUE. La phase present peut être suivie par une phase delete. Les 2 phases sont délimitées par un message refreshPresent Sync Info avec la valeur refreshDone à FALSE. Dans le cas où les 2 phases sont utilisée, la phase present est utilisée pour apporte la copie client à jours après laquelle la phase delete peut commencer.

Dans la phase present, le serveur envoie une entrée vide (ex : pas d'attributs) avec un Sync State Control indiquant l'état présent pour chaque entrée inchangée.

La phase delete peut être utilisée quand le serveur peut déterminer avec certitude quelles entrées dans la précédente copie cliente ne sont plus présent dans le contenu et le nombre de telles entrées est inférieur ou égal au nombre d'entrées inchangées. Dans le mode delete, le serveur envoie une entrée vide avec un Sync State Control indiquant l'état delete pour chaque entrée qui n'est plus dans le contenu, au lieu de retourner une entrée vide avec l'état present pour chaque entrée présente.

Le serveur peut envoyer des messages syncIdSet Sync Info Messages contenant le jeu d'UUID des entrées présentes non changées et les entrées supprimées, au lieu d'envoyer plusieurs messages individuels. Il refreshDeletes et syncIdSet sont FALSE, les UUID des entrées présente inchangées sont contenus dans le jeu syncUUID ; si refreshDeletes de syncIdSet est à TRUE, l'UUID des entrées supprimées dans le contenu sont dans le jeu syncUUID. Un cookie optionnel peut être inclus dans le syncIdSet pour représenter l'état du contenu après avoir synchronisé la présence ou l'absence des entrées contenus dans le jeu syncUUID.

La copie synchronisée du fragment du DIT est construit par le client.

Si refreshDeletes de syncDoneValue est FALSE, la nouvelle copie inclue toutes les entrées changées retournées par le Sync Operation ré-émis, aussi bien que toutes les entrées non changées identifiée comme étant présentes par le Sync Operation ré-émis, mais dont le contenu est fournis par le Sync Operation précédent. Les entrées inchangées non identifiées qui sont présentées sont supprimées du contenu client. Ils ont été soit supprimés, déplacés, ou sortis du scope.

Si refreshDeletes de syncDoneValue est TRUE, la nouvelle copie inclus toutes les entrées changées retournées par le Sync Operation ré-émis, aussi bien que toutes les autres entrées de la copie précédente excepté pour celles qui ont été identifiées comme ayant été supprimés du contenu.

Le client peut, plus tard, re-demander les changements pour cette copie cliente synchronisée.

Listening for Changes (refreshAndPersist)

Le Polling for changes peut être coûteux en terme de serveur, client et ressources réseaux. Le mode refreshAndPersist permet des mises à jours active des entrées changées dans le contenu.

En sélectionnant le mode RefreshAndPersist, le client demande que le serveur envoie les mises à jours des entrées qui ont été changées après que le refresh initial aie été déterminé. Au lieu d'un SearchResultDone comme dans le polling, le serveur envoie un Sync Info Message au client indiquant que l'étape refresh est complet et qu'il peut ainsi entrer en mode persist. Une fois reçu ce Sync Info Message, le client va construire une copie synchronisée comme décris précédemment.

Le serveur peut ainsi envoyer des notifications de changement en résultat de la requête de recherche Sync originale, qui maintenant reste persistante dans le serveur. Pour les entrées à ajouter au contenu retourné, le serveur envoie un SearchResultEntry (avec les attributs) avec un contrôle Sync State indiquant l'état add. Pour les entrées à supprimer du contenu, le serveur envoie un SearchResultEntry ne contenant aucun attribut et un contrôle Sync State indiquant l'état delete. Pour les entrées à modifier dans le contenu retourné, le serveur envoie un SearchResultEntry (avec les attributs) avec un contrôle Sync State indiquant l'état modify. Une fois la modification d'une entrée, tous les attributs (modifiés et non modifiés) appartenant au contenu sont envoyés.

Noter que renommer une entrée du DIT peut causer en état add où l'entrée est renommée dans le contenu, un état delete où l'entrée est sortie du contenu, et un état modify où l'entrée reste dans le contenu. Noter également qu'une modification d'une entrée du DIT peut causer un état add, delete ou modify dans le contenu.

Une fois une notification de changement reçue, le client met à jours sa copie du contenu.

Si le serveur désire mettre à jours de syncCookie durant l'étape persist, il peut inclure le syncCookie et un Sync State Control ou un Sync Info Message retourné.

L'opération persiste jusqu'à annulation par le client ou terminées par le serveur. Un contrôle Sync Done devrait être attaché au message SearchResultDone pour fournir un nouveau syncCookie.

Élements de Sync Operation

Sync Operation est définis comme extension de l'opération ldap search où le DUA envoie un message SearchRequest avec un contrôle Sync Request et le DSA répond avec 0 ou plusieurs message SearchResultEntry, chacun avec un contrôle Sync State ; 0 ou plusieurs messages intermédiaires Sync Info, et un message SearchResultDone avec un contrôle Sync Done.

Pour permettre aux clients de découvrir le support pour cette opération, les serveurs implémentant cette opération devaient publier **1.3.6.1.4.1.4203.1.9.1.1** comme valeur dans **supportedControl** du rootDSE.

SyncUUID

Le type de données syncUUID est un OCTET STRING maintenant un identifiant unique 16-octets

```
syncUUID ::= OCTET STRING (SIZE(16))
```

syncCookie

Le syncCookie est un outil de notation pour indiquer que, bien que le type syncCookie est encodé en OCTET STRING, sa valeur est une valeur opaque contenant des informations sur la session de synchronisation est son état. Généralement, les informations de session incluent un hash des paramètres de l'opération que le serveur impose de ne pas changer et les informations de synchronisation incluent un numéro de séquence d'envoie (log), un numéro de séquence de changement, ou un timestamp. Pour simplifier les descriptions, le term "aucun cookie" réfère soit à un cookie null, soit à un cookie avec un état de synchronisation pré-initialisé

```
syncCookie ::= OCTET STRING
```

Sync Request Control

Le contrôle Sync Request est un contrôle ldap où controlType est **1.3.6.1.4.1.4203.1.9.1.1** et controlValue, un OCTET STRING, contient un BER de syncRequestValue :

```
syncRequestValue ::= SEQUENCE {
 mode ENUMERATED {
  - 0 unused
  refreshOnly (1),
  - 2 reserved
  refreshAndPersist (3)
 },
 cookie syncCookie OPTIONAL,
 reloadHint BOOLEAN DEFAULT FALSE
}
```

Sync State Control

Le contrôle Sync State est un contrôle ldap où controlType est **1.3.6.1.4.1.4203.1.9.1.2** et controlValue, un OCTET STRING, contient un BER de syncStateValue :

```
syncStateValue ::= SEQUENCE {
 state ENUMERATED {
  present (0),
  add (1),
  modify (2),
  delete (3)
 },
 entryUUID syncUUID,
 cookie syncCookie OPTIONAL
}
```

Le contrôle Sync State est seulement applicable aux messages SearchResultEntry et SearchResultReference.

Sync Done Control

Le contrôle Sync Done est un contrôle ldap où controlType est **1.3.6.1.4.1.4203.1.9.1.3** et controlValue, un OCTET STRING, contient un BER de syncDoneValue :

```
syncDoneValue ::= SEQUENCE {
```

```
 cookie syncCookie OPTIONAL,
 refreshDeletes BOOLEAN DEFAULT FALSE
}
```

Le contrôle Sync Done est seulement applicable au message SearchResultDone.

Sync Info Message

Le message Sync Info est un message de réponse intermédiaire où responseName est **1.3.6.1.4.1.4203.1.9.1.4** et responseValue contient un BER de syncInfoValue :

```
syncInfoValue ::= CHOICE {
 newcookie [0] syncCookie,
 refreshDelete [1] SEQUENCE {
  cookie syncCookie OPTIONAL,
  refreshDone BOOLEAN DEFAULT TRUE
 },
 refreshPresent [2] SEQUENCE {
  cookie syncCookie OPTIONAL,
  refreshDone BOOLEAN DEFAULT TRUE
 },
 syncIdSet [3] SEQUENCE {
  cookie syncCookie OPTIONAL,
  refreshDeletes BOOLEAN DEFAULT FALSE,
  syncUUIDs SET OF syncUUID
 }
}
```

Sync Result Codes

Le resultCode ldap suivant est définis :

```
e-syncRefreshRequired (4096)
```

Content Synchronization

Sync Operation est invoqué quand le client envoie un message SearchRequest avec un contrôle Sync Request.

L'absence d'un cookie ou d'un état de synchronisation dans un cookie indique une requête pour un contenu initial, alors que la présence d'un cookie représentant un état d'une copie client indique un requête pour une mise à jours du contenu.

Le mode est soit refreshOnly soit refreshAndPersist. refreshOnly consiste seulement d'une étape refresh, alors que refreshAndPersist consiste d'une étape refresh suivi d'une étape persist.

Synchronization Session

Une séquence de Sync Operations où le dernier cookie retourné par le serveur pour une opération est fournie par le client dans la prochaine opération est dite appartenant à la même session de synchronisation.

Le client doit spécifier les même paramètre de contrôle de contenu dans chaque Search Request de la session. Le client devrait également fournir chaque demande Sync d'une session sous la même association authentification et autorisation avec des des protections et intégrité équivalent. Si le serveur ne reconnaît pas le cookie ou si la requête est faite sous des associations différentes ou à des protections non-équivalentes, le serveur devrait retourner le contenu initial comme si aucun cookie n'avait été fournis ou retourner un contenu vide avec le code de résultat e-syncRefreshRequired. La décision entrée le retour du contenu initial et le retour du contenu vide avec le code de résultat e-syncRefreshRequired peut être basé sur le reloadHint dans le contrôle de requête Sync du client. Si le serveur reconnait le cookie comme représentant un état de synchronisation initial ou vide de la copie cliente, le serveur devrait retourner le contenu initial.

Une session de synchronisation peut s'étendre sur plusieurs sessions entre le client et le serveur. Le client devrait fournir chaque demande Sync d'une session du même serveur.

Détermination du contenu

Le contenu à fournir est déterminé par les paramètre de la requête search, comme décrit dans la rfc4511, et possiblement d'autres contrôles. Les même paramètres de contenu devraient être utilisés dans chaque requête Sync d'une session. Si un contenu différent est demandé et que le serveur n'est pas en mesure de traiter la requête, le serveur devrait retourner le contenu initial comme si aucun cookie n'avait été retourné ou retourner un contenu vide avec le code e-syncRefreshRequired. La décision entrée les 2 peut être basé sur reloadHint du contrôle Sync Request du client.

Le contenu peut ne pas nécessairement inclure toutes les entrées ou références qui seraient retournées par une opération de recherche normale, ni, pour les entrées inclues, tous les attributs retournés par une recherche normale. Quand le serveur ne peut pas fournir de synchronisation pour un attribut pour un jeu d'entrées, le serveur doit traiter tous les composants de filtre qui matchent ces attributs comme indéfinis et ne doit pas retourner ces attributs dans les réponses SearchResultEntry.

Les serveurs devraient supporter la synchronisation pour tous les attributs utilisateurs non-collectifs pour toutes les entrées. Le serveur peut également retourner des références de continuation aux autres serveurs et à lui-même. Ce dernier est permis vu que le serveur peut partitionner les entrées qu'il maintient dans des contextes de synchronisation séparés. Le client peut poursuivre toutes ou certaines continuations, chacune comme une session de synchronisation de contenu séparés.

Mode refreshOnly

Une requête Sync avec le mode refreshOnly et sans cookie est un sondage pour un contenu initial. Une requête Sync avec le mode refreshOnly et avec un cookie représentant un état de synchronisation est un sondage pour une mise à jours de contenus.

Initial Content Poll

Une fois reçu la requête, le serveur fournis le contenu initial en utilisant un jeu de 0 ou plusieurs messages SearchResultEntry et SearchResultReference suivs par un message SearchResultDone.

Chaque message SearchResultEntry devrait inclure un contrôle Sync State avec l'état add, un entryUUID contenant l'UUID de l'entrée, et aucun cookie. Chaque message SearchResultReference devrait inclure un contrôle Sync State avec l'état add, un entryUUID contenant l'UUID associé avec là référence (normalement l'UUID de l'objet référant nommé associé), et aucun cookie. Le message searchResultDone devrait inclure un contrôle Sync Done avec refreshDeletes à FALSE.

Une valeur resultCode à success indique que l'opération a été complétée avec succès. Sinon, le resultCode indique la nature de l'échec. Le serveur peut retourner le resultCode e-syncRefreshRequired dans le sondage de contenu initial s'il est sûre de le faire quand il n'est pas en

mesure d'effectuer l'opération. reloadHint est mis à FALSE dans le message SearchRequest nécessitant l'initial content poll.

Si l'opération est un succès, un cookie représentant l'état synchronisé de la copie cliente devrait être retournée pour être utilisé dans d'autres Sync Operations.

Content Update Poll

Une fois la requête reçue, le serveur fournis la mise à jours du contenu en utilisant un jeu de 0 ou plusieurs messages SearchResultEntry et SearchResultReference suivi par un message SearchResultDone. Le serveur doit :

a) Fournir la séquence des messages nécessaire pour une convergence éventuelle de la copie cliente au contenu de la copie du serveur

b) Traiter la requête comme demande de contenu initial (ignore le cookie ou l'état de synchronisation)

c) Indiquer que la convergence incrémentale n'est pas possible en retournant e-syncRefreshRequired

d) Retourner un resultCode autre que success ou e-syncRefreshRequired

Un Sync Operation peut consister d'une simple phase present, une simple phase delete, ou une phase present suivi par une phase delete.

Dans chaque phase, pour chaque entrée ou référence qui a été ajouté au contenu ou a été changé depuis le Sync Operation précédent indiqué par le cookie, le serveur retourne un message SearchResultEntry ou SearchResultReference, respectivement, chacun avec un contrôle Sync State consistant de l'état add, un entryUUID contenant l'UUID de l'entrée ou de la référence, et aucun cookie. Chaque message SearchResultEntry représente l'état courant d'une entrée changée. Chaque message searchResultReference représente l'état courant d'une référence changée.

Dans la phase present, pour chaque entrée qui n'a pas été changée depuis le Sync Operation précédent, un SearchResultEntry vide est retourné dont l'objectName reflète le DN courant de l'entrée, dont le champ attributes est vide, et dont le contrôle Sync State consiste de l'état present, un entryUUID contenant l'UUID de l'entrée, et aucun cookie. Pour chaque référence qui n'a pas été changé depuis le Sync Operation précedent, un SearchResultReference vide contenant une sequence de LDAPURL vide est retournée avec un contrôle Sync State consistant le l'état present, un entryUUID contenant l'UUID de l'entrée, et aucun cookie. Aucun message n'est envoyé pour les entrées ou références qui ne sont plus dans le contenu.

De multiples entrées vides avec un contrôle Sync State à l'état present devraient être fusionnés en un ou plusieurs message Sync Info de valeur syncIdSet avec refreshDeletes à FALSE. syncUUID contient un jeu d'UUID d'entrées et références inchangées depuis le dernier Sync Operation. syncUUID peut être vide. Le message Sync Info de syncIdSet peut contenir un cookie pour représenter l'état du contenu après avoir effectué la synchronisation des entrées dans le jeu.

Dans la phase delete, pour chaque entrée qui n'est plus dans le contenu, le serveur retourne un SearchResultEntry dont l'objectName reflète un DN passé de l'entrée ou vide, dont le champ attributes est vide, et dont le contrôle Sync State a un état delete, un entryUUID contenant l'UUID de l'entrée supprimée, et aucun cookie. Pour chaque référence qui n'est plus dans le contenu, un SearchResultReference contenant une séquence de LDAPURL vide est retourné avec un contrôle Sync State à l'état delete, un entryUUID contenant l'UUID de la référence supprimée, et aucun cookie.

De multiples entrées vides avec un contrôle Sync State à l'état delete devraient être fusionnés en un ou plusieurs message Sync Info de valeur syncIdSet avec refreshDelete à TRUE. syncUUID contient un jeu d'UUID d'entrées et références supprimées du contenu depuis le dernier Sync Operation. syncUUID peut être vide. Le message Sync Info de syncIdSet peut contenir un cookie pour représenter l'état du contenu après avoir effectué la synchronisation des entrées dans le jeu.

Quand une phase present est suivie par une phase delete, le 2 phases sont délimitées par un message Sync Info contenant syncInfoValue de refreshPresent, qui peut contenir un cookie représentant l'état après avoir complété la phase present. Le refreshPresent contient refreshDone, qui est toujours FALSE dans le mode refreshOnly parce qu'il est suivi par une phase delete.

Si un Sync Operation consiste en un simple phase, chaque phase et par conséquent le Sync Operation sont marqués comme terminés par un message SearchResultDone avec le contrôle Sync Done, qui devrait contenir un cookie représentant l'état du contenus après avoir complété le Sync Operation. Le contrôle Sync Done contient refreshDeletes, qui est à FALSE pour la phase present et à TRUE pour la phase delete.

Si un Sync Operation consiste d'une phase present suivie par une phase delete, le Sync Operation est marqué comme terminé à la fin de la phase delete par un message SearchResultDone avec le contrôle Sync Done, qui devrait contenir un cookie représentant l'état du contenu après avoir complété le Sync Operation. Le contrôle Sync Done contient refreshDeletes, qui est à TRUE.

Le client peut spécifier s'il préfère recevoir un contenu initial en fournissant reloadHint à TRUE ou recevoir un e-syncRefreshRequired en fournissant reloadHint à FALSE (donc absent), dans ce cas le serveur détermine s'il est impossible ou inéfficace d'achever la convergence éventuelle en continuant le thread de synchronisation incrémentale courant.

Une valeur resultCode de success indique que l'opération est complétée avec succès. Un resultCode de e-syncRefreshRequired indique qu'un refresh complet ou partiel est nécessaire. Sinon, le resultCode indique la nature de l'erreur. Un cookie est fournis dans le contrôle Sync Done pour l'utiliser dans les Sync Operations suivantes pour une synchronisation incrémentale.

Mode refreshAndPersist

Une requête Sync avec le mode refreshAndPersist demande le contenu initial ou la mise à jours courante (durant l'étape refresh) suivie par des notifications de changement (durant l'étape persist).

Étape refresh

Le contenu refresh est fourni comme décris précédemment, excepté que le refresh du contenu complété avec succès est indiqué en envoyant un message Sync Info de refreshDelete ou refreshPresent avec une valeur refreshDone à TRUE au lieu d'un message SearchResultDone avec un resultCode success. Un cookie devrait être retourné dans le message Sync Info pour représenter l'état du contenu après avoir firi l'état refresh du Sync Operation.

Étape persist

Des notifications de changement sont fournis durant l'étape persist. Lorsqu'un changement est fait dans le DIT, le serveur notifie le client des changements du contenu. Les mises à jours du DIT peut causer des entrées et références qui sont ajoutées, supprimées ou modifiées dans le contenu.

Là où des mises à jours du DIT implique qu'une entrée a été ajoutée au contenu, le serveur fournis un message SearchResultEntry qui représente l'entrée telle qu'elle apparaît dans le contenu. Le message devrait inclure un contrôle Sync State avec l'état add, un entryUUID contenant l'UUID de l'entrée et un cookie optionnel.

Là où des mises à jours du DIT implique qu'une référence est ajoutée au contenu, le serveur fournis un message SearchResultReference qui représente la référence dans le contenu. Le message devrait inclure un contrôle Sync State avec l'état add, un entryUUID contenant l'UUID associée avec la référence, et un cookie optionnel.

Là où des mises à jours du DIT implique qu'une entrée a été modifiée dans le contenu, le serveur fournis un message SearchResultEntry qui représente l'entrée telle qu'elle apparaît dans le contenu. Le message devrait inclure un contrôle Sync State avec l'état modify, un entryUUID contenant l'UUID de l'entrée, et un cookie optionnel.

Là où des mises à jours du DIT implique qu'une référence est modifiée dans le contenu, le serveur fournis un message SearchResultReference qui représente la référence dans le contenu. Le message devrait inclure un contrôle Sync State avec l'état modify, un entryUUID contenant l'UUID associée avec la référence, et un cookie optionnel.

Là où des mises à jours du DIT implique qu'une entrée a été supprimée du contenu, le serveur fournis un message SearchResultEntry sans attributs. Le message devrait inclure un contrôle Sync State avec l'état delete, un entryUUID contenant l'UUID de l'entrée, et un cookie

optionnel.

Là où des mises à jours du DIT implique qu'une référence est supprimée du contenu, le serveur fournis un message SearchResultReference avec un LDAPURL vide. Le message devrait inclure un contrôle Sync State avec l'état delete, un entryUUID contenant l'UUID associée avec la référence, et un cookie optionnel.

plusieurs entrées vides avec un contrôle Sync State delete devraient être fusionnés en un ou plusieurs message Sync Info de valeur syncIdSet avec refreshDeletes à TRUE. syncUUID contient un jeu d'UUID des entrées et références qui ont été supprimés du contenu. Le message Sync Info de syncIdSet peut contenir un cookie pour représenter l'état du contenu après avoir effectué la synchronisation des entrées dans le jeu.

Avec chacun de ces messages, le serveur peut fournir un nouveau cookie pour être utilisé dans les Sync Operation suivants. Additionnellement, le serveur peut également retourner des messages Sync Info avec un newCookie pour fournir un nouveau cookie. Le client devrait utiliser le dernier cookie qu'il a reçu du serveur pour les Sync Operation suivants.

Paramètres de requête de recherche

Comme vu plus haut, le client devrait spécifier les même paramètres de contrôle de contenu dans chaque requête de recherche de la session. Tous les champs du message SearchRequest sont considérés comme des paramètres de contrôle de contenu excepté pour sizeLimit et timeLimit.

baseObject

Comme avec une opération de recherche normale, les étapes refresh et persist ne sont pas isolés des changement du DIT. Il est possible que l'entrée référé par le baseObject soit supprimé, renommé, ou déplacé. Il est également possible que l'objet alias utilisé pour trouver l'entrée référée par le baseObject soit changé de telle sorte que le baseObject réfère à une entrée différente.

Si le DIT est mis à jours durant le traitement du Sync Operation d'une manière qui cause le baseObject à ne plus référer à une entrée ou dans une manière qui change l'entrée auquel il fait référence, le serveur devrait retourner un resultCode approprié, tel que noSuchObject, aliasProblem, aliasDereferencingProblem, referral, ou e-syncRefreshRequired.

derefAliases

Cette opération ne supporte pas le déréférencement d'alias durant la recherche. Le client doit spécifier neverDerefAliases ou derefFindingBaseObj pour le paramètre derefAliases du SearchRequest. Le serveur devrait traiter les autres valeurs (ex : derefInSearching, derefAlways) comme des erreurs de protocole.

sizeLimit

sizeLimit s'applique seulement aux entrées (sans regarder leur état dans le contrôle Sync State) retourné durant l'opération refreshOnly ou l'étape refresh de l'opération refreshAndPersist.

timeLimit

pour un Sync Operation, le timeLimit s'applique à toute l'opération. Pour une opération refreshAndPersist, e timeLimit s'applique seulement à l'étape refresh incluant la génération du message Sync Info avec une valeur refreshDone à TRUE.

filtre

Le client devrait éviter les filtres qui s'appliquent aux valeurs des attributs utilisés par le serveur comme maintenant les meta-informations.

objectName

Le Sync Operation utilise les valeurs entryUUID fournis dans le contrôle Sync State comme clé primaire aux entrées. Le client doit utiliser ces entryUUID pour corréler les messages de synchronisation.

Dans certaines circonstances, le DN retourné peut ne pas refléter le DN courant de l'entrée. En particulier, quand l'entrée a été supprimée du contenu, le serveur peut fournir un DN vide si le serveur ne désire pas informer du DN courant de l'entrée(ou, si supprimé du DIT, le dernier DN de l'entrée).

Noter également que le DN de l'entrée peut être vue comme une méta-information (voir plus bas).

Annuler le Sync Operation

Les serveurs doivent implémenter l'opération ldap Cancel et supporter l'annulation des opérations de synchronisation. Pour annuler le Sync Operation en cour, le client fournis une opération ldap Cancel.

Si à un moment le serveur n'est plus en mesure de continuer l'opération, le serveur devrait retourner un SearchResultDone avec un resultCode indiquant la raison de l'arrêt de l'opération

Que ce soit le client ou le serveur qui initie la fin, le serveur peut fournir un cookie dans le contrôle Sync Done pour l'utiliser dans les Sync Operation suivants.

Refresh Required

Pour accomplir la synchronisation à convergence éventuelle, le serveur peut terminer le Sync Operation dans les étapes refresh ou persist en retournant un resultCode e-syncRefreshRequired au client. Si aucun cookie n'est fournis, un refresh complet est nécessaire. Si un cookie représentant un état synchronisé est fournis dans cette réponse, un refresh incrémental est nécessaire.

Le serveur peut choisir de fournir une copie complète dans l'étape refresh (ex : en ignorant le cookie ou l'état de synchronisation représenté dans le cookie) au lieu de fournir un refresh incrémentale pour accomplir la convergence éventuelle.

La décision entre le retour du contenu initial et le retour du resultCode e-syncRefreshRequired peut être basé sur le reloadHint dans le contrôle Sync Request du client.

Dans le cas de l'étape persist, le serveur retourne le resultCode e-syncRefreshRequired au client pour indiquer que le client doit ré-émettre un nouveau Sync Operation pour obtenir une copie synchronisée du contenu. Si aucun cookie n'est fournis, un refresh complet est nécessaire. Si un cookie représentant un état synchronisé est fournis, un refresh incrémental est nécessaire.

Le serveur peut également retourner un e-syncRefreshRequired s'il détermine qu'un refresh serait plus efficace qu'envoyer tous les messages requis pour la convergence.

Noter que le client peut recevoir un ou plusieurs messages SearchResultEntry, SearchResultReference, et/ou des Sync Info avant qu'il reçoive un message SearchResultDone avec le resultCode e-syncRefreshRequired.

Considérations diverses

Le serveur doit s'assurer que le nombre de messages générés pour refresh le contenu client n'excède pas le nombre d'entrées présentes dans le contenu. Bien qu'il n'y ait pas de pré-requis pour les serveurs de maintenir des informations d'historique, si le serveur a un historique suffisant pour déterminer avec certitude quelles entrées dans la copie client ne sont plus présentes dans le contenu et que le nombre de telles entrées est inférieur ou égal au nombre d'entrées inchangées, le serveur devrait générer des messages delete au lieu de message d'entrées présentes.

Quand la quantité d'historique maintenu dans le serveur n'est pas suffisant pour que les clients puissent les refreshOnly peut fréquents, c'est généralement que le serveur a des informations d'historique incomplets.

Le serveur ne devrait pas recourir à un reload complet quand les informations d'historique ne sont pas suffisants pour générer des message d'entrées supprimées. Le serveur devrait générer soit des messages d'entrée présentes uniquement ou des messages d'entrées présentes suivis par les messages d'entrée supprimées pour ramener la copie client à l'état courant. Dans le dernier cas, les messages d'entrées présentes apporte la copie cliente à un état couvert par les informations d'historique maintenus dans le serveur.

Le serveur devrait maintenir suffisamment (courant et historique) d'information d'état (tel qu'un horodatage de dernière modification à l'échelle du contexte) pour déterminer si aucun changement n'a été fait dans le contexte depuis que le refresh a été fournis et, quand aucun changement n'a été fait, ne génère aucun message d'entrée supprimée au lieu de messages présents.

Le serveur ne devrait pas utiliser les informations d'historique quand son utilisation ne réduit pas le trafic de synchronisation ou quand sont utilisation peut exposer des informations sensibles non permises par le client.

L'implémenteur du serveur devrait également considérer les problèmes de bavardage qui s'étendent sur plusieurs opérations de synchronisation d'une session. Comme noté plus haut, le serveur peut retourner e-syncRefreshRequired s'il détermine qu'un reload sera plus efficace que continuer l'opération courante. Si reloadHint dans le Sync Request est TRUE, le serveur peut initier un reload sans diriger le client pour demander un reload.

Le serveur devrait transférer un nouveau cookie fréquemment pou éviter d'avoir à transférer des informations déjà fournies au client. Même où les changement du DIT n'impliquent pas de changements dans la synchronisation du contenu à transférer, il peut être avantageux de fournir un nouveau cookie en utilisant un message Sync Info. Cependant, le serveur devrait éviter de surcharger le client ou le réseau avec des messages Sync Info.

Durant le mode persist, le serveur devrait fusionner les messages de mises à jours de la même entrée. Le serveur peut retarder la génération de la mise à jours d'une entrée en anticipation de changements ultérieurs que cette entrée qui pourraient être fusionnés. La longueur du delai devrait être suffisamment long pour permettre de fusionner les demandes de mises à jours émises mais suffisamment court pour que l'inconsistance transitoire induite par le délai soit corrigé.

Le serveur devrait utilise le message Sync Info syncIdSet quand il y a plusieurs messages delete ou present pour réduire la quantité de trafic de synchronisation.

Noter également qu'il peut y avoir plusieurs clients intéressés par le changement particulier, et que ces serveurs qui tentent de tous les desservir en même temps peuvent causer des congestion sur le réseau. Les problèmes de congestion sont intensifiés quand les changements nécessitent un gros transfert pour chaque client intéressé. Les implémenteurs et déployeurs de serveur devraient prendre en compte les étapes pour empêcher et gérer la congestion du réseau.

Multiplexer les opérations

Le protocole LDAP permet de multiplexer les opérations dans une simple session LDAP. Les clients ne devraient pas maintenir plusieurs sessions LDAP avec le même serveur. Les serveurs devraient s'assurer que les réponses des opérations traité concurrentiellement sont entrelacés de façon équitable.

Les clients devraient combiner les Sync Operation dont le jeu de résultat est largement chevauché. Les clients ne devraient pas combiner les Sync Operation dont les jeux de résultat sont largement non-chevauchés. Cela s'assure que dans le cas d'une réponse e-syncRefreshRequired peut être limitée à quelques jeux de résultats.

Entry DN

Le DN de l'entrée est construite depuis son RDN et le DN de son parent, il est dont souvent vu comme une méta-information. renommer ou déplacer à un nouveau supérieur, le DN de l'entrée est changé, ce changement ne devrait pas, par lui-même, causer de message de synchronisation à envoyer pour cette entrée. Cependant, Si le renommage ou le déplacement ajoute ou supprime l'entrée du contenu, les messages de synchronisation appropriés devraient être générés pour l'indiquer au client.

Quand un serveur traite le DN de l'entrée comme méta-information, le serveur devrait soit :

- Évaluer tous MatchingRuleAssertions à TRUE si matche une valeur d'un attribut de l'entrée, sinon Undefined, ou
- Évaluer tous MatchingRuleAssertions avec dnAttributes à TRUE comme Undefined.

Le dernier choix est offert pour simplifier l'implémentation de serveurs.

Attributs Opérationnels

Lorsque les valeurs d'un attribut opérationnel sont déterminés par les valeurs non maintenues dans l'entrée, l'attribut opérationnel ne devrait pas supporter la synchronisation de cet attribut.

Les serveurs devraient supporter la synchronisation des attributs opérationnels suivants : createTimestamp, modifyTimestamp, creatorsName, modifiersName. Les serveurs peuvent supporter la synchronisation d'autres attributs opérationnels.

Attributs collectifs

La modification d'un attribut collectif affecte généralement le contenu de plusieurs entrées, qui sont les membres de la collection. Il n'est pas efficace d'inclure les valeurs des attributs collectifs visible dans les entrées de la collection, vu que la modification d'un simple attribut collectif nécessite la transmission de plusieurs SearchResultEntry (un pour chaque entrées dans la collection que la modification affecte)

Les serveurs ne devraient pas synchroniser les attributs collectifs apparaissant dans les entrées d'une collection. Les serveur peuvent supporter la synchronisation des attributs collectifs apparaissant dans les attributs collectif des subentries.

Accès et autres contrôle administratifs

Les entrées sont communément sujets à accès et autres contrôles administratifs. Bien que des portions d'information de stratégie gouvernant une entrée particulière peuvent être maintenus dans l'entrée, les informations de stratégies sont souvent maintenus ailleurs. À

cause de cela, les changement d'information de stratégie rendent plus difficile la vérification de convergence éventuel durant la synchronisation incrémentale.

Lorsqu'il n'est pas possible ou infaisable de générer des changement de contenu résultant d'un changement d'information de stratégie, les serveurs peuvent opter pour retourner e-syncRefreshRequired ou pour traiter le Sync Operation comme requête de contenu initial.

Intéraction avec d'autres contrôles

Le Sync Operation peut être utilisé avec :

- ManageDsaIT
- Subentries Control

ManagesDsaIT

Le Contrôle ManageDsaIT indique que l'opération agit sur le DIT et force les objets spéciaux à être traités comme des entrées.

Subentries Control

Le contrôle Subentries est utilisé avec l'opération de recherche pour contrôler la visibilité des entrées et des subentries qui sont dans le scope. Quand utilisé avec le Sync Operation, le contrôle subentries et d'autres facteurs sont utilisés pour déterminer si une entrée ou un subentry apparait dans le contenu.

Shadowing

Certains serveurs peuvent maintenir des copies shadow des entrées qui peuvent être utilisée pour répondre à des requêtes search et compare. De tels serveurs peuvent également supporter les requêtes de synchronisation de contenu.

Shadowing

Bien qu'un client peut connaître plusieurs serveurs qui sont capables d'être utilisé pour obtenir un contenu d'annuaire particulier, un client ne devrait pas assumer qui chacun de ces serveurs est capable de continuer une session de synchronisation de contenu. Le client devrait fournir chaque requête Sync d'une session Sync au même serveur.

Cependant, via le nom de domaine ou les redirections d'adresse IP ou d'autres techniques, plusieurs serveurs physiques peuvent être fait pour apparaître comme un serveur logique à un client. Seuls les serveurs qui sont également capable en ce qui concerne leur support de Sync Operation et qui maintienne également des copies complètes des entrées devraient être faits pour apparaître comme un serveur logique. En particulier, chaque serveur physique agissant comme un serveur logique devrait être également capable de continuer une synchronisation de contenu basé sur des cookies fournis par un autre serveur physique sans nécessiter un full reload. Parce qu'il n'y pas de mécanisme de shadowing ldap standard, la spécification de la manière d'implémenter cette capacité est laissé à de futures documents.

Noter qu'il peut être difficile pour le serveur de déterminer avec certitude quel contenu a été fournis au client par un autre serveur, spécifiquement dans les environnement Shadowing qui permet aux évènements shadowing d'être fusionnés. Pour ces serveurs, l'utilisation de la phase delete peut ne pas être applicable.

Considérations de sécurité

Pour maintenir une copie synchronisée du contenu, un client doit supprimer les informations de sa copie du contenu comme décris plus haut. Cependant, le client peut maintenir les connaissances des informations qui lui sont communiquées par le serveur séparé de sa copie du contenu utilisé pour la synchronisation. La gestion de ces connaissances est hors du scope de ce document. Les serveurs devraient faire attention à ne pas communiquer les informations du contenu que le client n'a pasle droit de connaître.

Bien que les informations fournies par une série de Sync Operation refreshOnly est similaire à celles fournies par une série d'opération de recherche, l'étape persist peur communiquer des informations additionnelles. Un client peut être capable de discerner les informations sur la séquence d'opérations de mises à jours particulière qui à causé le changement de contenu.

Les implémenteurs devraient prendre des précautions contre le contenu de cookie malicieux, incluant les cookies malformés ou les cookies valides utilisé avec différentes protections et/ou associations de sécurité pour tenter d'obtenir un accès non autorisé aux informations. Les serveurs peuvent inclure une signature numérique dans le cookie pour détecter la falsification.

L'opération peut être la cible d'attaque DOS direct. Les implémenteurs devraient fournir des protections contre ça. Les serveurs peuvent placer le contrôle d'accès ou d'autres restrictions contre l'utilisation de cette opération.

Noter que même les petites mises à jours peuvent causer une quantité significative de trafic. Un utilisateur pourrait abuser de ses privilèges de mise à jours pour monter un DOS indirecte.

Considérations d'implémentation basé sur CSN

Cet appendix discute de l'implémentation de l'opération ldap de synchronisation de contenu associé avec l'approche basé sur le numéro de séquence de changement.

Cet approche est ciblée pour l'utilisation dans des serveurs qui ne maintiennent pas d'information d'historique sur les changements faits dans le DIT et ainsi doit s'assurer de l'état de l'annuaire courant et un état de synchronisation minimal embraqué dans le Sync Cookie. Les serveurs qui implémentent des informations d'historique devraient considérer d'autres approches qui exploitent ces informations d'historique.

Un CSN est effectivement un timestamp qui a la granularité suffisante pour s'assurer que la relation de précedence dans le temps de 2 mises à jours sur le même objet peut être déterminer. Les CSN ne doivent pas être confondus avec les Commit Sequence Number ou Commit Log Record Number. Un Commit Sequence Number permet de déterminer comment 2 envoies (sur le même objet ou différents) sont liés l'un à l'autre dans le temps. Un Change Sequence Number associé avec des entrées différentes pour être envoyés dans le désordre. Dans le reste de cet appendix, le terme CSN réfère toujours au Change Sequence Number.

Dans ces approches, le serveur ne maintient pas seulement un CSN pour chaque entrée de l'annuaire mais maintient également une valeur qui l'on appel le context CSN. Ce context CSN est le plus grand entry CSN envoyé qui n'est pas supérieur à une entry CSN exceptionnelle (non validée) pour toutes les entrées dans le contexte de l'annuaire. Les valeurs du context CSN sont utilisée dans les valeurs syncCookie comme indicateurs d'état de synchronisation.

Vu que les opérations de recherche ne sont pas isolées des mises à jours d'annuaire individuels et les opérations de mises à jours individuels ne peuvent pas assumer être sérialisés, on ne peut pas assumer que le contenu retourné incorpore chaque changement dont le CSN est inférieur ou égal au plus grand entryCSN dans le contenu. Le contenu incorpore tous les changements dont les CSN sont inférieurs ou égal au contextCSN avec de traiter la recherche. Le contenu peut également incorporer un sous-jeu de changements dont le CSN est supérieur au contextCSN avant chaque traitement de recherche mais inférieur ou égal au contextCSN après le traitement de la recherche. Le contenu n'incorpore pas les changements dont le CSN est supérieur au contextCSN après le traitement de la recherche.

Une implémentation de serveur simple pourrait utiliser la valeur du contextCSN avec de traiter la recherche pour indiquer l'état. Une telle implémentation embarquerai cet valeur dans chaque syncCookie retourné. On appel çà le cookie CSN. Quant un refresh est demandé, le serveur va simplement générer des messages de mises à jours pour tous les entrées dans le contenu dont le CSN est supérieur au cookieCSN fournis et générer des messages present pour toutes les autres entrées dans le contenu. Cependant, si le contextCSN courant est le même que le cookieCSN, le serveur devrait générer 0 updates et 0 delete et indiquer un refreshDeletes à TRUE, vu que l'annuaire n'a pas changé.

L'implémentation devrait également considérer l'impact des changements des méta-informations, tel que les contrôles d'accès, qui affecte la détermination du contenu. Un approche est pour le serveur de maitenir un CSN meta. Ce metaCSN serait mis à jours quand une méta-information affectant la détermination du contenu a été changée. Si la valeur du metaCSN est supérieur au cookieCSN, le serveur devrait ignorer le cookie et traiter la demande comme requête initial pour le contenu.

Additionnellement, les serveurs peuvent vouloir maintenir des information d'historique par session pour réduire le nombre de messages nécessaires à transférer durant les refresh incrémentales. Spécifiquement, un serveur pourrait enregistrer les informations sur les entrées qui ont quitté le scope d'une session sync déconnectée et l'utiliser ultérieurement pour générer des informations delete au lieu de messages present.

Quand les informations d'historique sont tronqués, le CSN de la dernière information d'historique tronquée peut être enregistrée comme CSN tronquée de l'information d'historique. Le CSN tronqué peut être utilisé pour déterminer si la copie client peut être couvert par les informations d'historique en les comparant à l'état de synchronisation contenu dans le cookie fournis par le client.

quand il y a un grand nombre de sessions, il peut être censé de maintenir un tel historique seulement pour le clients sélectionnés. De même, les serveurs utilisant cette approche nécessitent de considérer les problèmes de consommation de ressource pour s'assurer de performance raisonnable et éviter les abus. Il peut être approprié de restreindre ce mode d'opération par stratégie.