NoSQL²: Store LDAP Data in HBase

Stefan Seelmann
seelmann@apache.org
About me

• Stefan Seelmann
• Freelancer
  – Software development with Java
  – LDAP, Identity Management
• Open Source developer
  – Apache Directory project
  – DataNucleus LDAP store
NoSQL² – Store LDAP Data in HBase

Agenda

- Apache Directory project
- LDAP is NoSQL
- Motivation for HBase backend
- Schema Design: how LDAP data fits into HBase
  - LDAP Information Model
  - LDAP Naming Model
  - LDAP Functional Model
- Why HBase? And why not?
- Status, Future
- Demo
Apache Directory Server

- Directory Server in Java
- Protocols: LDAP, DNS, DHCP, NTP, Kerberos, Change Password
- X.500 ACI, Triggers, Stored Procedures
- Open Group certified LDAPv3 server
- In progress: ChangeLog, Replication (RFC 4533), Configuration in DIT
Apache Directory Studio

- Directory client platform
- Eclipse based
  - Integration into Eclipse
  - Application for Linux (x32/x64), Mac OS X, Windows
- Features:
  - LDAP Browser
  - LDIF Editor
  - Schema Editor
  - Integrated ApacheDS
  - Configuration for ApacheDS
LDAP is NoSQL

- Classical Backends
  - B+tree based: BerkleyDB, FLAIM, JET Blue, JDBM
  - Oracle and IBM use their RDBMS

- Replication
  - Proprietary protocols, now RFC 4533 (content sync)
  - multi-master for high availability
  - master-slave for read-scalability
  - “eventually consistent”

- Partitioning
  - distributed tree
  - for write-scalability
• LDAPCon2009: OpenLDAP and OpenDS projects presented MySQL NDB backend
  – using relations
  – max. DN depth
  – fixed value size
  – no substring matching
• Entry consists of a set of attributes
• Attribute description and one or many values
• Schema:
  – object classes
  – attribute types
  – syntaxes
  – matching rules

```
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Horatio Nelson
uid: hnelson
mail: hnelson@example.com
userPassword:: e1NIQX01ZW42RzZNZXPScm...
jpegPhoto:: /9j/4AAQSkZJRgABAQAAAQABA....
entryUUID: 44444444-4444-4444-4444-444444444444
createTimestamp: 20100506180000Z
...```

LDIF notation
Table „master“

Column family „upAttributes“ for attributes

One row per entry, entryUUID as row key

Additional index for multi-valued attributes

<table>
<thead>
<tr>
<th>master</th>
<th>treeInfo</th>
<th>upAttributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>44444444....</td>
<td>...</td>
<td>objectClass0 -&gt; top</td>
</tr>
<tr>
<td></td>
<td></td>
<td>objectClass1 -&gt; person</td>
</tr>
<tr>
<td></td>
<td></td>
<td>objectClass2 -&gt; organizationalPerson</td>
</tr>
<tr>
<td></td>
<td></td>
<td>objectClass3 -&gt; inetOrgPerson</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cn0 -&gt; Horatio Nelson</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uid0 -&gt; hnelson</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mail0 -&gt; <a href="mailto:hnelson@example.com">hnelson@example.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>userPassword0 -&gt; &lt;bytes&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>jpegPhoto0 -&gt; &lt;bytes&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>entryUUID0 -&gt; 44444444-4444-4444-4444-444444444444</td>
</tr>
<tr>
<td></td>
<td></td>
<td>createTimestamp -&gt; 20100506180000Z</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Naming Model (1)

- Hierarchical structure
- Relative DN: one or multiple attributes of entry
- Distinguished Name: composition of RDNs till root
  - root is right!
  - DN is not stable!

\[
\text{cn}=\text{Horatio Nelson}, \text{ou}=\text{people}, \text{o}=\text{sevenSeas}
\]
### Table "master"
- column family "treeInfo": parentId, upRdn, normRdn
- don't store DN, only RDN and pointer to parent
- used to resolve UUID to DN

<table>
<thead>
<tr>
<th>master</th>
<th>treeInfo</th>
<th>upAttributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000-....</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11111111-....</td>
<td>parentId -&gt; 00000000-....</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>upRdn -&gt; o=sevenSeas</td>
<td></td>
</tr>
<tr>
<td></td>
<td>normRdn -&gt; 2.5.4.10=sevensseas</td>
<td></td>
</tr>
<tr>
<td>22222222-....</td>
<td>parentId -&gt; 11111111-....</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>upRdn -&gt; ou=people</td>
<td></td>
</tr>
<tr>
<td></td>
<td>normRdn -&gt; 2.5.4.11=people</td>
<td></td>
</tr>
<tr>
<td>44444444-....</td>
<td>parentId -&gt; 22222222-....</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>upRdn -&gt; cn=Horatio Nelson</td>
<td></td>
</tr>
<tr>
<td></td>
<td>normRdn -&gt; 2.5.4.3=horatio nelson</td>
<td></td>
</tr>
</tbody>
</table>
• Table "tree"
  – column family „treeInfo“
  – used to resolve DN to UUID

<table>
<thead>
<tr>
<th>tree</th>
<th>treeInfo</th>
<th>normAttributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000...,2.5.4.10=sevenseas</td>
<td>id -&gt; 11111111...,</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>11111111...,2.5.4.11=people</td>
<td>id -&gt; 22222222...,</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>22222222...,2.5.4.3=horatio nelson</td>
<td>id -&gt; 44444444...,</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

• Fast enough for flat trees
• Cache branch entries, with TTL
Functional Model - Write

- **add entry**
  - put()

- **modify entry**
  - get() + apply modifications + checkAndPut()
  - null out deleted attributes, for audit log

- **delete entry**
  - null out attributes
  - no real delete, just mark as deleted, for audit log

- **moddn (move, rename)**
  - just update pointers to parent
Functional Model - Search (1)

- **Parameters**
  - base DN
  - scope: base, one, sub
  - filter
    - equal, substring, presence
    - less than eq, greater than eq
    - and, or, not
    - (uid=hnelson)
    - (&(objectClass=person)(mail=*@example.com) (modifyTimestamp>=20100101)(!(uid=*)))

- **Procedure:**
  - get each entry in scope and check if filter matches
• For luck HBase has scanner with filters
• Normalized attributes in „tree“ table
  – qualifier is attribute description + value
  – SingleColumnQualifierFilter
  – one-level and sub-level count

<table>
<thead>
<tr>
<th>tree</th>
<th>treeInfo</th>
<th>normAttributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>22222222-...,2.5.4.3 =horatio nelson</td>
<td>id -&gt; 44444444-...</td>
<td>0.9.2342.19200300.100.1.1=hnelson -&gt; 0</td>
</tr>
<tr>
<td></td>
<td>oneLevelCount -&gt; 0</td>
<td>0.9.2342.19200300.100.1.3=<a href="mailto:hnelson@example.com">hnelson@example.com</a> -&gt; 0</td>
</tr>
<tr>
<td></td>
<td>subLevelCount -&gt; 0</td>
<td>2.5.4.0=inetorgperson -&gt; 3</td>
</tr>
<tr>
<td></td>
<td>status -&gt; e</td>
<td>2.5.4.0=organizationalperson -&gt; 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.5.4.0=person -&gt; 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.5.4.0=top -&gt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.5.4.3=horatio nelson -&gt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.5.4.35=&lt;bytes&gt; -&gt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
• HBase scanner:
  – base -> UUID as scanner start and stop row
  – filter -> converted to scanner filter
  – scope sub -> recursive scans if one-level count > 0

• Better, but still not optimal
  – too large scan range, e.g. for (uid=foobar)

<table>
<thead>
<tr>
<th>tree</th>
<th>treeInfo</th>
<th>normAttributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>222222222----, 2.5.4.3</td>
<td>id -&gt; 44444444----</td>
<td>0.9.2342.19200300.100.1.1=hnelson -&gt; 0</td>
</tr>
<tr>
<td></td>
<td>oneLevelCount -&gt; 0</td>
<td>0.9.2342.19200300.100.1.3=<a href="mailto:hnelson@example.com">hnelson@example.com</a> -&gt; 0</td>
</tr>
<tr>
<td></td>
<td>subLevelCount -&gt; 0</td>
<td>2.5.4.0=inetorgperson -&gt; 3</td>
</tr>
<tr>
<td></td>
<td>status -&gt; e</td>
<td>2.5.4.0=organizationalperson -&gt; 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.5.4.0=person -&gt; 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.5.4.0=top -&gt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.5.4.3=horatio nelson -&gt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.5.4.35=&lt;bytes&gt; -&gt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Functional Model - Search (4)

- ApacheDS XDBM search engine
  - for B+Tree like backends
  - uses index tables
  - search path optimization, based on candidate count

- Two type of attribute index tables
  - column based
  - row based

- Additional: presence
  - entryUUID as row key
  - check with exists()

<table>
<thead>
<tr>
<th>index_givenname</th>
<th>info</th>
</tr>
</thead>
<tbody>
<tr>
<td>*44444444...</td>
<td>status -&gt; e</td>
</tr>
<tr>
<td>*77777777...</td>
<td>status -&gt; e</td>
</tr>
</tbody>
</table>
• Column based index tables
  – for attributes with (almost) unique values, e.g. uid, mail
  – value as row key, entryUUID as column qualifier
  – eq: just a get to retrieve all entryUUIDs, count is cheap
  – substr: scan, regex filter
    • additional start and stop row for substring initial filters
  – gte: scan, start row
  – lte: scan, stop row

<table>
<thead>
<tr>
<th>index_givenname</th>
<th>info</th>
</tr>
</thead>
<tbody>
<tr>
<td>=cornelius</td>
<td>77777777-... -&gt; e</td>
</tr>
<tr>
<td>=horatio</td>
<td>44444444-... -&gt; e</td>
</tr>
<tr>
<td></td>
<td>CCCCCCCC-... -&gt; e</td>
</tr>
<tr>
<td>=john</td>
<td>99999999-... -&gt; e</td>
</tr>
<tr>
<td></td>
<td>DDDDDDDD-... -&gt; e</td>
</tr>
<tr>
<td>=william</td>
<td>AAAAAAAAA-... -&gt; e</td>
</tr>
</tbody>
</table>
### Functional Model - Search (6)

- **Row based index tables**
  - for often used attribute values, e.g. objectClass:person
  - row key composed of value and entryUUID
  - no fixed length!
  - eq: scan with start and stop row, count more expensive
  - substr: scan, regex filter
    - additional start and stop row for initial filter
  - gte: scan, start row
  - lte: scan, stop row

<table>
<thead>
<tr>
<th>index_objectClass</th>
<th>info</th>
</tr>
</thead>
<tbody>
<tr>
<td>=inetorgperson\00&gt;44444444-</td>
<td>status -&gt; e</td>
</tr>
<tr>
<td>=inetorgperson\00&gt;99999999-</td>
<td>status -&gt; e</td>
</tr>
<tr>
<td>=person\00&gt;44444444-</td>
<td>status -&gt; e</td>
</tr>
<tr>
<td>=person\00&gt;99999999-</td>
<td>status -&gt; e</td>
</tr>
</tbody>
</table>
Why HBase?

- Java API, Thrift, REST, ...
- Embeddable for integration tests
- Soon in Maven repo
- Map/Reduce Jobs
  - bulk import job: write master table only
  - index job: creates tree and index tables
    - also useful for rebuilding indices
  - backup/restore
  - mass modifications (@sun.com -> @oracle.com)
  - analysis, data migration
Why HBase?

• Built-in replication
• Scalability
• Strong consistency
• Sparse, everything is a byte[]
• Versions for audit log
• Transactions
• Atomic increment/decrement/checkAndPut
• Scanner with ranges and filter
• Apache License
• Great community
Who not HBase?

- Heavy to setup
- Real tests require real hardware
- Exceptions in logs
- Security
  - no trusted connection between client and server
  - data isn't stored encrypted
  - no authn/authz
• All LDAP operations work
  – add, modify, delete, moddn, search
  – ApacheDS integration test pass
• Performance
  – Great improvement from HBase 0.20.3 -> 0.20.4
  – only tested pseudo-distributed and on VMware
• TODOs
  – make ApacheDS ready for distributed backends
    • event notification from HBase?
  – caching
  – schema and config partition
- HBase 0.20.4-RC5, standalone
- HBase Explorer 0.2.1
- ApacheDS trunk
  - 3 partitions:
    - o=hbase
    - o=sevenSeas
    - dc=example,dc=com
- Apache Directory Studio Plugin 1.5.3
Resources

• Apache Directory Project
  – http://directory.apache.org/

• Wiki page
  – https://cwiki.apache.org/confluence/display/DIRxSBOX/HBase+Prototype