Reftable
Faster Git-as-a-Database

Han-Wen Nienhuys
Agenda

- NoteDB, or Git-as-a-Database
- Ref storage - the Problem
- Reftable: All Hail The King
- Demo
- Outlook
What is a review?

- Code - refs/changes/12345/20
- Metadata
  - Status = active
  - PatchSet = 20

Create refs/changes/12345/21
- Advance refs/heads/master
- Metadata
  - Status = merged
  - Patchset = 21

- Code - refs/changes/12345/20
- Metadata
  - Status = abandoned
  - Patchset = 20
A Tale of Two Storage Systems

**NoteDb**: store metadata in Git branches
- refs/changes/12345/20
- refs/changes/12345/meta

**Reasons**: with two storage systems:
- Consistent operations?
- Concurrent writes?
- Takeout?
- Backup?
- Two systems to maintain, tune, etc.

**NoteDb occasion:**

Google:

“Hey Gerrit team, we’re shutting down your SQL database January 1, 2018.

Rewrite all your glue code, or else.”
NoteDb solves atomicity woes

- Code - refs/changes/12345/20
- Metadata - refs/changes/12345/meta
  - Status = active
  - PatchSet = 20

- Create refs/changes/12345/21
- Advance refs/heads/master
- Advance refs/change/12345/meta
  - Status = merged
  - Patchset = 21

Atomic ref update

SUBMIT (rebase)
Git branch access patterns

Read:
- Random access:
  - refs/changes/12345/meta
  - refs/heads/master
- Prefix search:
  - refs/heads/*
  - refs/changes/12345/*

Size:
- Chromium src: 1.7M branches
- Android pfb: 1.8M branches
Branch storage, v1: “loose refs”

File: `.git/refs/heads/master`

- Random access:
  - Open, Read/Write, Close
- Prefix search: `refs/changes/*`
  - Open, Read, Close directory
  - Recurse
- Space: 4096 bytes / ref, 1 file / ref

File system limitations:
- master/bla vs master
- MASTER vs master

$O(1)$, but system call overhead
Branch storage, v2: “packed refs”

File: `.git/packed-refs`

refs/heads/a abc123abc..
refs/heads/b 456abc123..
...
refs/heads/z 789def456..

- Random access
  - $O(\log N)$ in memory
- Prefix search
  - $O(\log N)$ in memory
- Space
  - 20 bytes + name / ref
- Write
  - Rewrite file, $O(N)$
Branch storage, v3: “packed refs” + “loose refs”

- Combine packed & loose
- Loose ref overrides packed-ref
- Compact occasionally

Performance
  - Packed-refs + a few reads

Space:
  - 20 bytes + name / ref

Write
  - write 1 file, O(1)

Difficult to understand

Deletion is O(N)
Atomicity in the file system

Lock = create single file
Transaction = rename file to destination

- Recompact loose refs
- Update ref data in memory
- Write new packed-refs file

Cost: O(N)
A NEW HOPE
Fix all of this

- We only need a format for fast reading
- Fast writing
  - transactions are small
  - Write just the delta
  - Merge delta on read
  - Compact regularly
- List of tables is a single file
  - can provide transactions

```
Merged table
master: 666666
stable: def4678
...
```
```
table-1.ref
.. master: abc1234
next: 77a2fde
stable: def4678
...
```
```
table-2.ref
master: 666666
```
```
table-3.ref
next: delete
```

A NEW HOPE
Storage likes to use blocks
- 4kb default
- 64kb at Google

Store keys sorted
Prefix compression for keys
Index key restarts

<table>
<thead>
<tr>
<th>'R' 4096</th>
<th>0refs/changes/12341</th>
<th>abc...</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>2</td>
<td>a2bc...</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>ab3c...</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>a4bc...</td>
</tr>
<tr>
<td>16</td>
<td>5</td>
<td>ab5c...</td>
</tr>
<tr>
<td>16</td>
<td>6</td>
<td>abc6...</td>
</tr>
<tr>
<td>0refs/heads/master</td>
<td>abc6...</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>stable</td>
<td>def7...</td>
</tr>
<tr>
<td>0refs/tags/v1.0</td>
<td>a1b3...</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>b1b3...</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>a2b3...</td>
</tr>
</tbody>
</table>

03 restarts
Block indexes

- Index minimizes number of seeks
- Index block holds
  - Key => Block offset
- Index blocks have same layout
  - Prefix compression!
- Large tables will have multiple levels
Fix other gripes too

- Store Reflog in “log” blocks
  - No more dir/file conflicts
  - RefDB + Reflog updates atomically
- Store SHA1 => ref mapping
  - Fast inverse lookups
  - Needed for visibility checks
  - Needed for Gerrit patch upload

Chromium push performance
### Ref storage in Bigtable at Google

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>android:wifi:changes/1000</td>
<td>a1b2c3...</td>
</tr>
<tr>
<td>android:wifi:changes/1001</td>
<td>ff7dcb...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Encryption**

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>android:NduoVksXjobBg</td>
<td>uoDCiroSd...</td>
</tr>
<tr>
<td>android:qX+uqdhzMCs7</td>
<td>9i37ZDCM...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Reftable history

- Aug 2017: Shawn Pearce introduces reftable format
- Dec 2017: reftable deployed at Google
- Nov 2019: JGit support in FileRepository
Demo

- Demo
- Measurements for write rate (synthetic)
  - 1ms/update (SSD, Linux/Mac)
  - 20 ms/update (NFS)
- Gerrit benchmark:
  - 1700 changes, SSD storage
  - Reftable
  - Packed-refs: 123ms / createchange (median)
  - Reftable: 71ms / createchange (median)
Outlook

- JGit: [https://git.eclipse.org/r/c/146568/](https://git.eclipse.org/r/c/146568/)
- Library: [https://github.com/google/reftable](https://github.com/google/reftable)
  - Go - full implementation of (de)deserialization
  - C - the same; reflog storage missing
  - Plan: integrate into git-core
- CGit doesn’t support it yet
  - Hooks plugin?
- Ref storage is transparent to Gerrit
  - Go back and forth