Scaling Up JGit

Shawn Pearce, Google

Git Go *Fast*

(live demo)

Reduced Latency

Client Operation	Distance Behind	Classic Algorithm	Hybrid Bitmaps
Clone	*(all)*	37,530 ms	82 ms
Fetch	1 commit	75 ms	107 ms
Fetch	10 commits	456 ms	341 ms
Fetch	100 commits	449 ms	337 ms
Fetch	1,000 commits	2,229 ms	189 ms
Fetch	10,000 commits	2,177 ms	254 ms
Fetch	100,000 commits	14,340 ms	1,655 ms

Bitmap Indexes

Clone and Fetch Performance

- Typically dominated by "Counting: ..." phase
- Cost is linear with complexity and age of repository
- Linux kernel: 2.9M objects, 100% server CPU, 60s counting time

Bitmap Indexes

- Pre-computed reachability data for commits in a pack file
- Linux kernel bitmap data: 2.9 MiB (for 559 MiB pack, 78 MiB idx)
- <80ms counting time

- Auxiliary data ignored by git-core
- In JGit master, ships in JGit 3.0 and Gerrit Code Review 2.6

Bitmap Structure

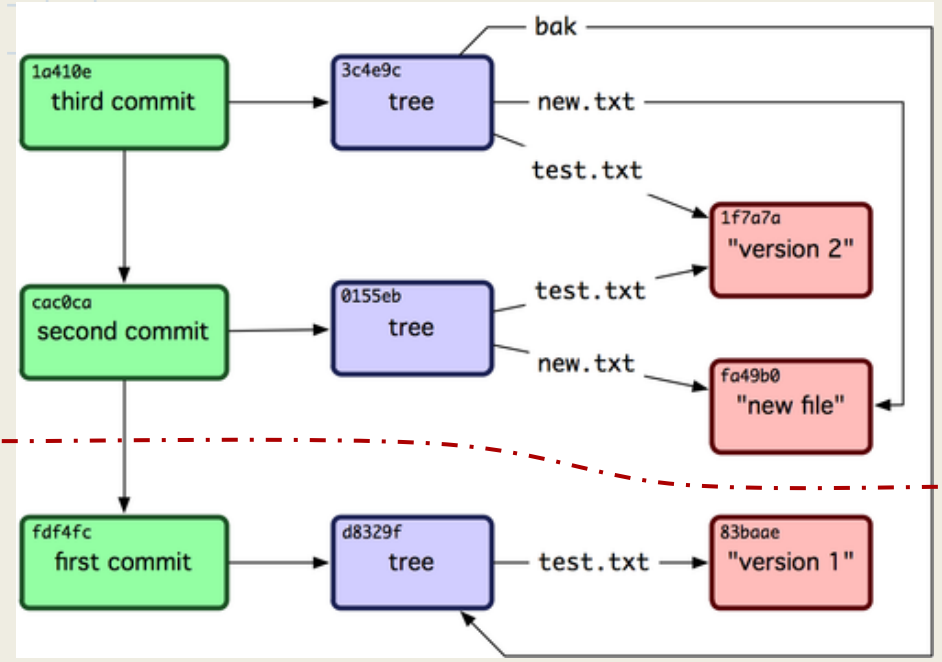


Image credit: *ProGit*, by Scott Chacon

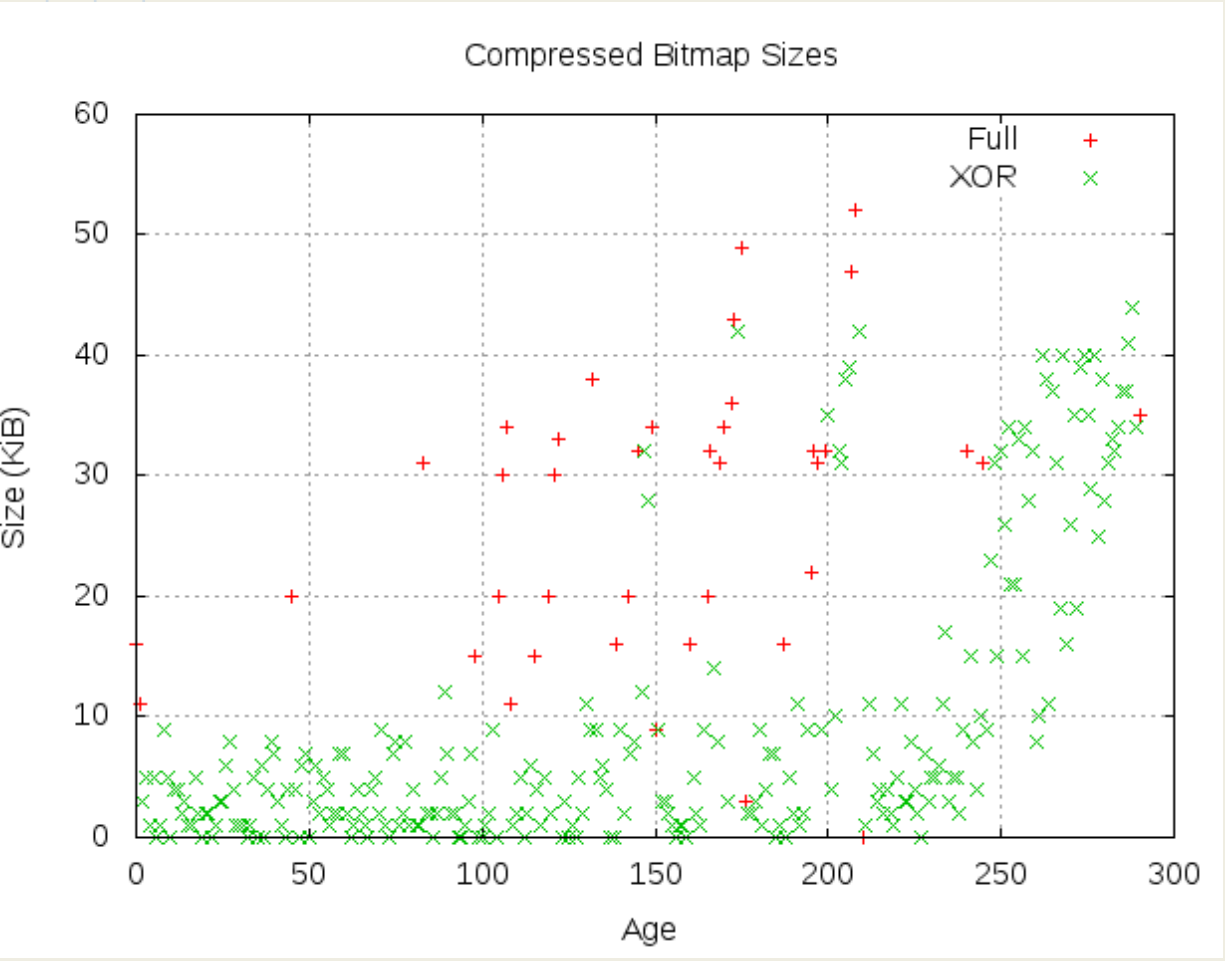
	1a410e	cac0ca	fdf4fc	3c439c	0155eb	d8329f	1f7a7a	fa49b0	83baae
1a410e	1	1	1	1	1	1	1	1	1
cac0ca	0	1	1	0	1	1	1	1	1
fdf4fc	0	0	1	0	0	1	0	0	1

- Assign bits by order of objects in pack file
- Objects to send = `want AND_NOT have`

- Bitmaps **are** compressible
- EWAH bitmap, variant of RLE

Daniel Lemire, Owen Kaser, Kamel Aouiche,
Sorting improves word-aligned bitmap indexes,
Data & Knowledge Engineering, Volume 69, Issue 1, 2010

Compressed Bitmap Sizes



	Full	XOR	Both
Bitmaps	39	252	291
Total Size	1 MiB	2.4 MiB	3.4 MiB
Avg. Size	26.8 KiB	9.8 KiB	12.1 KiB
Bits / Byte	98	267	217

2.7M bits in a bitmap
60% space savings with XOR
~1 ms CPU to recreate full version

Git Hosting At Google

	Repositories	Size	Traffic
Android Open Source Project android.googlesource.com	~490	19.4G *largest: 2.3G*	2.5M req./day 1.4T+5.0T data/day
Public Mirrors {apache,eclipse,gnu,kernel,webkit} .googlesource.com	~1,730	259.2G *largest: 4.4G*	0.8M req./day 0.8T data/day
"Other Stuff"	???	??? *largest: 147.2G*	???
Total	???	???	??? 360+485 Mbps 5200 peak qps

Managed and served using JGit and Gerrit Code Review.
Data does not include code.google.com Git support.

(Data Center+CDN)

JGit Storage Systems

Local Filesystem

- On-disk compatible with git-core
- Works out of the box
- Widely deployed (EGit, Gerrit Code Review, ...)

DFS ("Distributed File System")

- Stored format *not* compatible with git-core
- Requires shared filesystem and database
- Different buffer cache implementation *(hard references)*

- Abstract implementation (~8300 lines, 11 abstract methods)
- Only two known deployments

Local Filesystem Storage

References

- Text files mapping branch name (`refs/heads/master`) to SHA-1

`$GIT_DIR/refs/heads/master` and `$GIT_DIR/packed-refs`

Loose Objects

- Individually compressed object (file) contents
- Created as new objects enter the repository

`$GIT_DIR/objects/ab/cdef...{38}`

Packs

- Heavily delta compressed, multiple objects per file
- Generated by `git repack` or `git gc`
- Over time, replaces loose objects

`$GIT_DIR/objects/pack/pack-abcdef...{40}.{pack,idx}`

Scaling Local Filesystem

Repositories
- Difficult to manage 450 repositories under a directory tree
- Hard to query "who needs to repack?"
- Hard to replicate to multiple servers (post-receive hook + push?)

References
- Loose references are expensive to read; writes always make loose

Objects
- Loose objects are expensive to read; expensive for fetch and clone
- Constant repacking consumes server CPU time, disk IO time, RAM
- Limited number of cores available per motherboard

Server
- Single server can only scale so much CPU
- Git servers on NFS still eventually bottleneck on the NFS server

DFS Storage at Google

- Reference Database
  - Google Bigtable
  - One row per reference

```
b6:gerrit/gerrit\001refs/heads/master
  ref:                                              @ 2013-03-15 03:02:07
    object_id: "626334ded051865024d2ddf985ac939515fe791c"
    is_peeled: true
    name: "refs/heads/master"

b6:gerrit/gerrit\001refs/heads/stable
  ref:                                              @ 2012-02-28 21:12:19
    object_id: "dc225268721fec39010c9faeeb5980805b5e9bb7"
    is_peeled: true
    name: "refs/heads/stable"

b6:gerrit/gerrit\001refs/tags/v2.5.2
  ref:                                              @ 2013-02-12 02:13:14
    object_id: "2aee634f6e69646a55b76a28202a463224094ccd"
    is_peeled: true
    peeled_id: "b9ac2439c45862fecb232ded8fa70b47f2c48945"
    name: "refs/tags/v2.5.2"
```

- Range scan `[b6:gerrit/gerrit\001 ... b6:gerrit/gerrit\002)` replaces packed-refs
- Atomic compare-and-swap handled by Bigtable

DFS Storage at Google

Object Storage

No loose objects, everything is stored as a pack.
pack v2 and idx v2 file formats *(minor incompatible change in pack v2 header)*

Two categories of packs:

small: pack <= 2 MiB and idx <= 2 MiB
Mostly recent pushes, merges created by Gerrit Code Review
Stored in Bigtable

everything else: typically results of Git GC
Stored in archive files in GFS *(think ZIP or TAR without compression)*

GFS is not like ext2/3/4
Slower access
Everything is over the network
Prefers a few big files over many small files

DFS Storage at Google

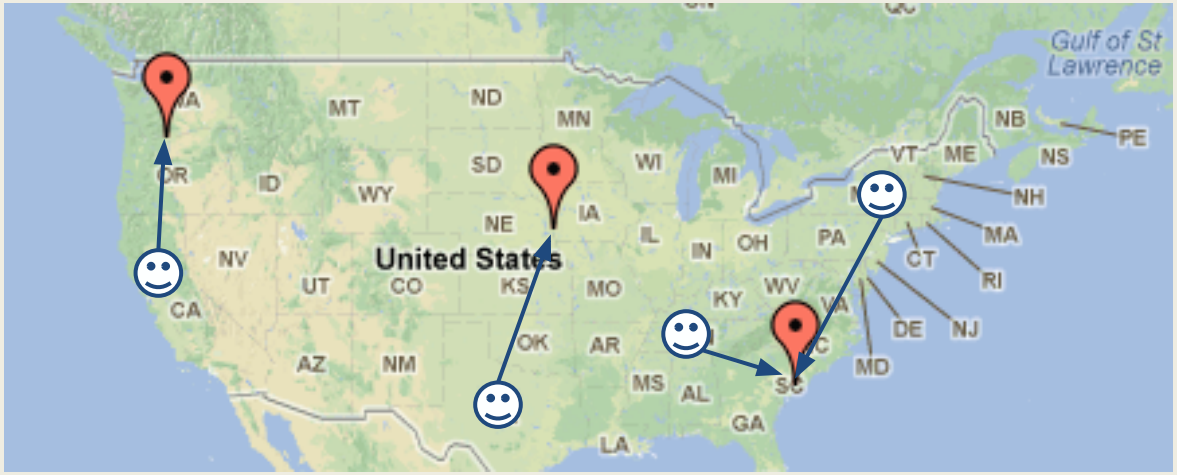
Pack Listing

- Google Bigtable
- Stored in a single row per repository; one cell per pack
- Replaces `readdir("$GIT_DIR/objects/pack")`

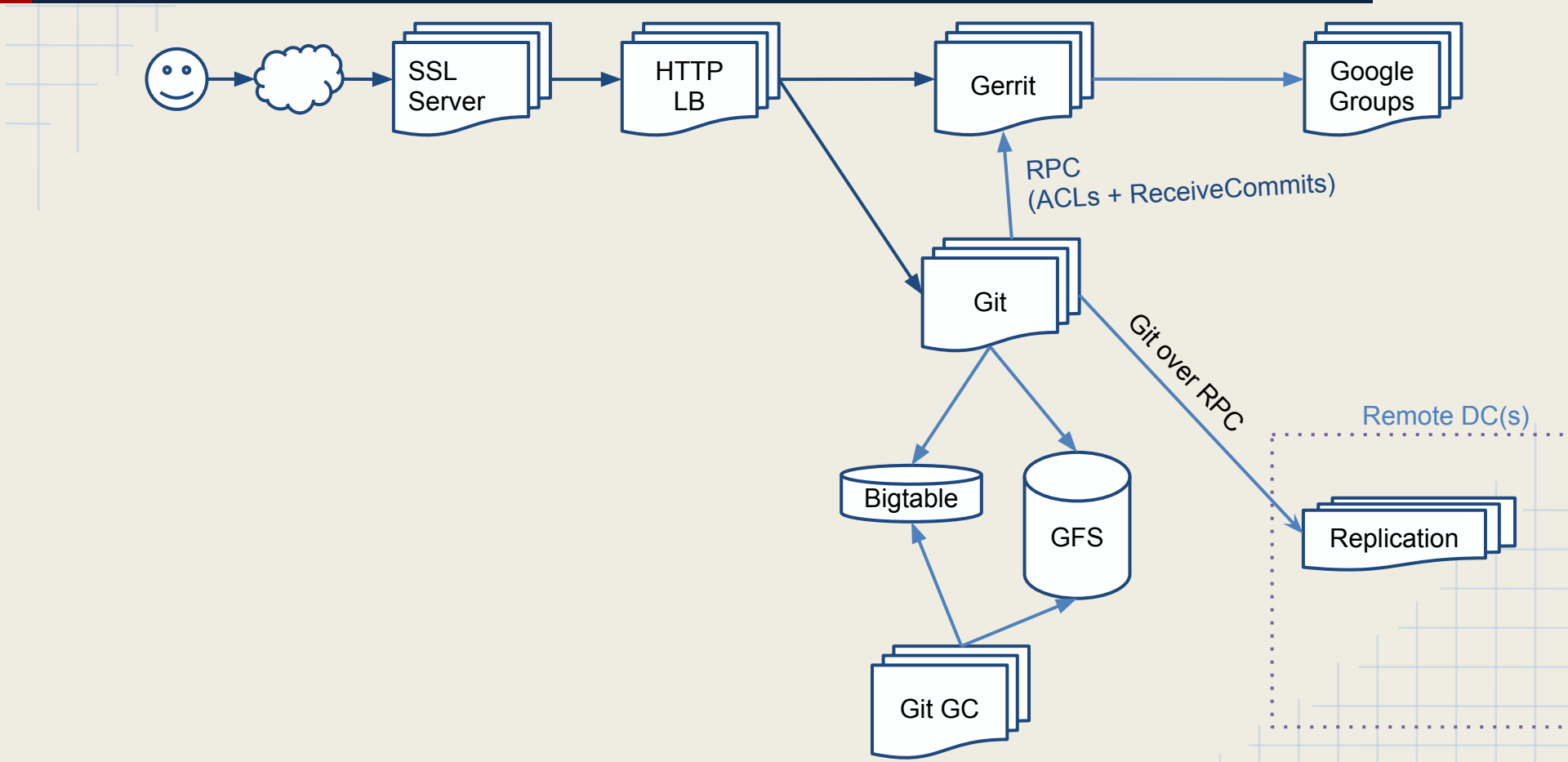
```
b6:gerrit/gerrit
  pack-info:54d/8                                    @ 2013-03-15 09:31:50
    source: UNREACHABLE_GARBAGE
    pack   < offset: 65536    length: 2788596            >
    idx    < offset: 2854132  length: 239744  version: 2 >

  pack-info:ab2/E.gar/0                              @ 2013-03-15 09:31:50
    source: GC
    pack   < offset: 65536    length: 16924041           >
    idx    < offset: 16989577 length: 2611960 version: 2 >
    bitmap < offset: 19601537 length: 100360             >

  pack-info:ab2/E.gar/1                              @ 2013-03-15 09:31:50
    source: GC
    pack   < offset: 19726336 length: 17725063           >
    idx    < offset: 37451399 length: 2950844 version: 2 >
```

- Packs created at the same time share the same file in the filesystem.
- Offset and length provide seek hint within the archive.

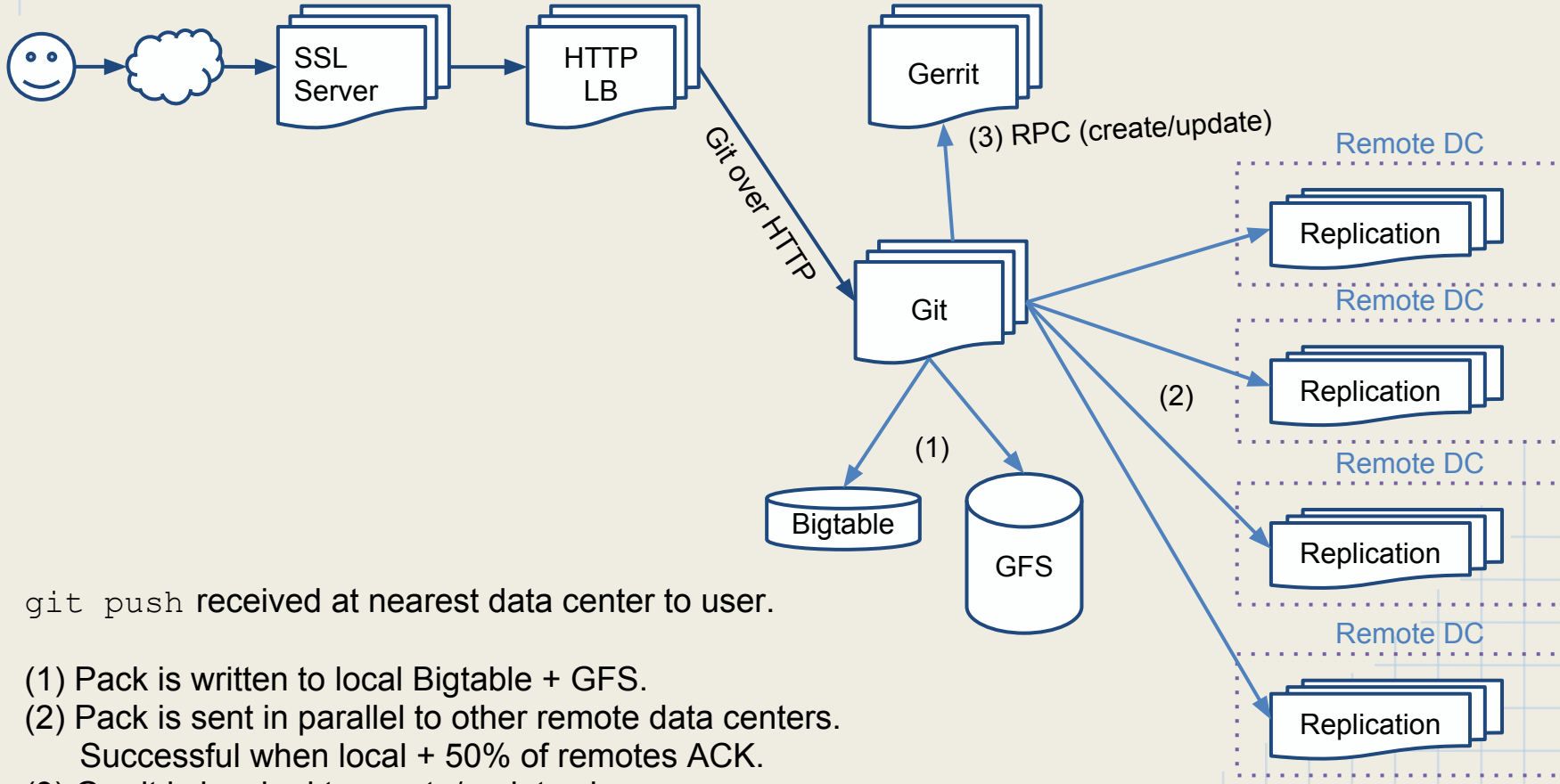
Distributed Hosting



Git on Google's DFS Storage



Handling Git Push



`git push` received at nearest data center to user.

(1) Pack is written to local Bigtable + GFS.
(2) Pack is sent in parallel to other remote data centers.
Successful when local + 50% of remotes ACK.
(3) Gerrit is invoked to create/update changes.

Remaining remote data centers catch up "soon".

Configuring Big JGit Servers

Buffer cache is important:

-Xmx8g # gobs of JVM heap

```
# for `jgit daemon` or gerrit.config
[core]
    packedGitLimit = 4g             # around 50% of JVM heap
    packedGitOpenFiles = 8192       # don't forget to configure ulimit
    streamFileThreshold = 2047m     # avoids pathological inflate
    deltaBaseCacheLimit = 50m

# applies to `jgit gc`
[pack]
    bigFileThreshold = 20m          # don't delta compress big binaries
    indexVersion = 2                # use index v2
```

Future JGit work

- Optional hard references in buffer cache
- Optional per-thread delta base cache
- Remove streaming delta support *(too slow)*

Thank You

Shawn Pearce sop@google.com

eclipse.org/jgit
code.google.com/p/gerrit