Inside LiveJournal's Backend

or,

“holy hell that's a lot of hits!”

July 2004

Brad Fitzpatrick
brad@danga.com

Danga Interactive
danga.com / livejournal.com

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/1.0/ or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.
The Plan

- LiveJournal overview
- Scaling history
- Perlbal
  - load balancer
- memcached
  - distributed caching
- MogileFS
  - distributed filesystem
Before we begin...

• Question Policy
  – Anytime... interrupt!

• Pace
  – told to go slow
  – too bad
  – too much
  – hold on tight

http://www.danga.com/words/
LiveJournal Overview

- college hobby project, Apr 1999
- blogging, forums
- aggregator, social-networking ('friends')
- 3.9 million accounts; ~half active
- 50M+ dynamic page views/day. 1k+/s at peak hours
- why it's interesting to you...
  - 90+ servers, working together
  - lots of MySQL usage
  - lots of failover
  - Open Source implementations of otherwise commercial solutions
LiveJournal Backend
(as of a few months ago)
Backend Evolution

• From 1 server to 90+....
  – where it hurts
  – how to fix

• Learn from this!
  – don't repeat my mistakes
  – can implement much of our design on a single server
One Server

- shared server (killed it)
- dedicated server (killed it)
  - still hurting, but could tune it
  - learned Unix pretty quickly
  - CGI to FastCGI
- Simple
One Server - Problems

- Site gets slow eventually.
  - reach point where tuning doesn't help
- single point of failure
- Need servers
  - start “paid accounts”
Two Servers

- Paid account revenue buys:
  - Kenny: 6U Dell web server
  - Cartman: 6U Dell database server
    - bigger / extra disks
- Network simple
  - 2 NICs each
- Cartman runs MySQL on internal network
Two Servers - Problems

- Two points of failure
- No hot or cold spares
- Site gets slow again.
  - CPU-bound on web node
  - need more web nodes...
Four Servers

- Buy two more web nodes (1U this time)
  - Kyle, Stan
- Overview: 3 webs, 1 db
- Now we need to load-balance!
  - Kept Kenny as gateway to outside world
  - mod_backhand amongst 'em all
mod_backhand

- web nodes broadcasting their state
  - free/busy apache children
  - system load
  - ...
- internally proxying requests around
  - network cheap
Four Servers - Problems

• Points of failure:
  – database
  – kenny (but could switch to another gateway easily when needed, or used heartbeat, but we didn't)

• Site gets slow...
  – IO-bound
  – need another database server ...
  – ... how to use another database?
Five Servers
introducing MySQL replication

- We buy a new database server
- MySQL replication
- Writes to Cartman (master)
- Reads from both
Replication Implementation

- `get_db_handle() : $dbh`
  - existing
- `get_db_reader() : $dbr`
  - transition to this
  - weighted selection
- permissions: slaves select-only
  - mysql option for this now
- be prepared for replication lag
  - easy to detect in MySQL 4.x
  - user actions from $dbh, not $dbr
More Servers

- Site's fast for a while,
- Then slow
- More web servers,
- More database slaves,
- ...
- IO vs CPU fight
- BIG-IP load balancers
  - cheap from usenet
  - LVS would work too
  - nowadays: wackamole

Chaos!
Where we're at...
Problems with Architecture

or,
“This don't scale...”

- Slaves upon slaves doesn't scale well...
  - only spreads reads

w/ 1 server

500 reads/s
200 writes/s

w/ 2 servers

250 reads/s
200 write/s
250 reads/s
200 write/s
Eventually...

- databases eventual consumed by writing

```plaintext
3 r/s
400 write/s
3 r/s
400 write/s
3 r/s
400 write/s
3 r/s
400 write/s
3 r/s
400 write/s
3 r/s
400 write/s
3 r/s
400 write/s
3 r/s
400 write/s
3 r/s
400 write/s
3 r/s
400 write/s
3 r/s
400 write/s
```
Not to mention,

- Database master is point of failure
- Reparenting slaves on master failure tricky at best
  - (without downtime)
Spreading Writes

- Our database machines already did RAID
- We did backups
- So why put user data on 6+ slave machines? (~12+ disks)
  - overkill redundancy
  - wasting time writing everywhere
Introducing User Clusters

- Already had `get_db_handle()` vs `get_db_reader()`
- Specialized handles:
- Partition dataset
  - can't join. don't care. never join user data w/ other user data
- Each user assigned to a cluster number
- Each cluster has multiple machines
  - writes self-contained in cluster (writing to 2-3 machines, not 6)
User Cluster Implementation

- $u = LJ::load_user(“brad”)
  - hits global cluster
  - $u object contains its clusterid
- $dbcm = LJ::get_cluster_master($u)
  - writes
  - definitive reads
- $dbcr = LJ::get_cluster_reader($u)
  - reads
User Clusters

SELECT userid, clusterid FROM user WHERE user='bob'

userid: 839
clusterid: 2

OMG i like
totally hate
my parents
they just
dont
understand me
and i h8 the
world omg lol
rofl *! :^-^

add me as a
friend!!!

• almost resembles today's architecture
User Cluster Implementation

- per-user numberspaces
  - can't use AUTO_INCREMENT
  - avoid it also on final column in multi-col index: (MyISAM-only feature)
    - CREATE TABLE foo (uid INT, postid INT AUTO_INCREMENT, PRIMARY KEY (userid, postid))
- moving users around clusters
  - balancing disk IO
  - balance disk space
  - monitor everything
    - cricket
    - nagios
    - ...whatever works
- Our library on top of DBI
  - GPL; not packaged anywhere but our cvs
- Returns handles given a role name
  - master (writes), slave (reads)
  - directory (innodb), ...
  - cluster<n>{,slave,a,b}
  - Can cache connections within a request or forever
- Verifies connections from previous request
- Realtime balancing of DB nodes within a role
  - web / CLI interfaces (not part of library)
  - dynamic reweighting when node down
Where we're at...
Points of Failure

- 1 x Global master
  - lame
- $n$ x User cluster masters
  - $n$ x lame.
- Slave reliance
  - one dies, others reading too much

Solution?
Master-Master Clusters!

- two identical machines per cluster
  - both “good” machines
- do all reads/writes to one at a time, both replicate from each other
- intentionally only use half our DB hardware at a time to be prepared for crashes
- easy maintenance by flipping active node
- backup from inactive node
Master-Master Prereqs

- failover can't break replication, be it:
  - automatic
    - be prepared for flapping
  - by hand
    - probably have other problems if swapping, don't need more breakage
- fun/tricky part is number allocation
  - same number allocated on both pairs
  - avoid AUTO_INCREMENT
  - cross-replicate, explode.
  - do your own sequence generation w/ locking, 3rd party arbitrator, odd/even, etc...
Cold Co-Master

- Inactive pair isn't getting reads
- After switching active machine, caches full, but not useful (few min to hours)
- Switch at night, or
- Sniff reads on active pair, replay to inactive guy

Cold cache, sad.

7A

Clients

7B

Hot cache, happy.
Summary Thus Far

- Dual BIG-IPs (or LVS+heartbeat, or..)
- ~40 web servers
- 1 “global cluster”:
  - non-user/multi-user data
  - what user is where?
  - master-slave (lame)
    - point of failure; only cold spares
    - pretty small dataset (<4 GB)
      - future: MySQL Cluster!
        - in memory, shared-nothing, 99.999% uptime
- bunch of “user clusters”:
  - master-slave (old ones)
  - master-master (new ones)
- ...

Static files...
Dynamic vs. Static Content

- **static content**
  - images, CSS
  - TUX, epoll-thttpd, etc. w/ thousands conns
  - boring, easy

- **dynamic content**
  - session-aware
    - site theme
    - browsing language
  - security on items
  - deal with heavy (memory hog) processes
  - exciting, harder
Misc MySQL Machines

Diagram showing a network of servers and services, including:
- User
- Internet
- Internap
- Secure Servers
- BigIP
- Proxy Web
- NetApp
- Web Servers
- Mail
- MEMCACHE
- GLOBAL CLUSTER
- Cartman
- Chef
- Santa
- Green
- Ribeye
- Big Boned
- Syndication

Key:
- web request
- secure request
- mail request
- userpic request
- conditional
- database
- user data
- Pool of Peers
- Pool of Master/Slaves
MyISAM vs. InnoDB

- We use both
- MyISAM
  - fast for reading xor writing,
  - bad concurrency, compact,
  - no foreign keys, constraints, etc
  - easy to admin
- InnoDB
  - ACID
  - good concurrency
    - long slow queries while updates continue
    - directory server
- Mix-and-match.
Postfix & MySQL

- 4 postfix servers
  - load balance incoming connections w/ BIG-IP
  - each runs tiny MySQL install
    - replicates one table (email_aliases)
- Incoming mail uses mysql map type
  - To: brad@livejournal.com
  - SELECT email FROM email_aliases WHERE alias='brad@livejournal.com'
- Don't have rebuild huge DBM files every few minutes
Logging to MySQL

- mod_perl logging handler
- new table per hour
  - MyISAM
- Apache access logging off
  - diskless web nodes, PXE boot
  - apache error logs through syslog-ng
- INSERT DELAYED
  - increase your insert buffer if querying
- minimal/no indexes
  - table scans are fine
- background job doing log analysis/rotation
Load Balancing!
Load Balancing Problem
Overview

- slow clients (hogging mod_perl/php)
  - even DSL/Cable is “slow”
  - need to spoon-feed clients
    - who will buffer?
- heterogeneous hardware and response latencies
  - load balancing algorithms
  - unlucky, clogged nodes
- dealing with backend failures
- The “Listen Backlog Problem”
  - is proxy/client talking to kernel or apache?
- live config changes
Two proxy / load balancing layers

- **1: IP-level proxy**
  - little or no buffering
    - 1 or 2 machines
      - hot spare, stateful failover
    - finite memory
    - Gbps+ switching
- **2: HTTP-level proxy**
  - more machines
  - buffer here
Proxy layer 1: IP-level

• Options:
  – Commercial:
    • BIG-IP, Alteon, Foundry, etc, etc...
  – Open Source:
    • Linux Virtual Server, Wackamole*

• load balance methods:
  – round robin, weighted round robin
  – least connections

• some have L7 capabilities
  – useful, but still need another proxy layer...
Proxy layer 2: HTTP-level

- Options:
  - mod_proxy
    - “typical” setup with mod_perl
    - to one host by default
    - mod_rewrite + external map program (prg:) with mod_proxy dest ([P])
      - broadcast Apache free/idle status from Apache scoreboard
      - flakey
    - “proxy connect error” to clients
  - pound
  - mod_backhand
  - Squid
  - plb (pure load balancer)
- Frustrated, needy, we wrote our own...
Perlbal

- Perl
- uses Linux 2.6's epoll
- single threaded, event-based
- console / HTTP remote management
  - live config changes
- handles dead nodes
- static webserver mode
  - sendfile(), async stat() / open()
- plug-ins
  - GIF/PNG altering
- ...

Perlbal: Persistent Connections

- persistent connections
  - perlbal to backends (mod_perls)
  - know exactly when a connection is ready for a new request
    - keeps backends busy
    - connection known good
      - tied to mod_perl, not kernel

- verifies new connections
  - one new pending connect per backend
  - verifies backend connection
    - OPTIONS request w/ keep-alive
    - response quick for apache

- multiple queues
  - free vs. paid user queues
Perlbal: cooperative large file serving

- large file serving w/ mod_perl bad...
  - buffering
- internal redirects
  - to URLs (plural) or file path
    - (hence Perlbal's web server mode)
  - client sees no HTTP redirect
- The path:
  - Perlbal advertises "X-Proxy-Capability: reproxy" to backend
  - backend (mod_perl) does path trans & auth, sees proxy capability, sends URL/path back in header, not response
    - let mod_perl do hard stuff, not push bytes around
Internal redirect picture

1. HTTP request
2. HTTP request w/ X-Proxy-Capabilities: reproxy
4. Request
5. Response
6. Merged Response (3's headers, 5's body)
MogileFS: distributed filesystem

- looked into Lustre, GFS, scared of in-development status
- MogileFS main ideas:
  - files belong to classes
    - classes: minimum replica counts (thumbnails == 1)
  - track what devices (disks) files are on
    - states: up, temp_down, dead
  - keep replicas on devices on different hosts
    - Screw RAID! (for this, for databases it's good.)
  - multiple tracker databases
    - all share same MySQL cluster database
  - big, cheap disks (12 x 250GB SATA in 3U)
  - dumb storage nodes
MogileFS components

- **clients**
  - small, simple Perl library
  - FUSE filesystem driver (unfinished)
- **trackers**
  - interface between client protocol and MySQL Cluster
- **MySQL Cluster**
  - in memory, multiple machines
- **Storage nodes**
  - NFS or HTTP transport
    - [Linux] NFS *incredibly* problematic
    - HTTP transport is Perlbal with PUT & DELETE enabled
Large file GET request
Caching!
Caching

- caching's key to performance
- can't hit the DB all the time
  - MyISAM: major r/w concurrency problems
  - InnoDB: good concurrency
    - not as fast as memory
  - MySQL has to parse your queries all the time
    - better with new MySQL 4.1 binary protocol

Where to cache?
- mod_perl caching (address space per apache child)
- shared memory (limited to single machine, same with Java/C#/Mono)
- MySQL query cache: flushed per update, small max size
- HEAP tables: fixed length rows, small max size
memcached
http://www.danga.com/memcached/

- our Open Source, distributed caching system
- run instances wherever there's free memory
- no “master node”
- clients distribute requests
- In use by:
  - LiveJournal, Slashdot, Wikipedia, Meetup, mail systems, etc...
- protocol simple and XML-free; clients for:
  - perl, java, php(x3), python, ruby, C(?)...
How memcached works

- requests hashed out amongst instance "buckets"
  - CRC32("key") = 383472 % num_buckets = 6
  - bucket 23 ... server 10.1.0.23: send: "key" = "value"

3 hosts, 7 buckets; 512 MB = 1 bucket (arbitrary)

<table>
<thead>
<tr>
<th>10.1.0.18</th>
<th>1024 MB; buckets 0-1</th>
<th>weather = dismal</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.1.0.20</td>
<td>2048 MB; buckets 2-5</td>
<td>tu:29323 = 1091029955</td>
</tr>
<tr>
<td>10.1.0.23</td>
<td>512 MB; bucket 6</td>
<td>key = value</td>
</tr>
</tbody>
</table>
memcached – speed

- C  
  - prototype Perl version proved concept, too slow
- async IO, event-driven, single-threaded
- libevent (epoll, kqueue, select, poll...)  
  - run-time mode selection
- lockless, refcounted objects
- slab allocator  
  - glibc malloc died after 7~8 days  
    - variable sized allocations, long life = difficult  
    - slabs: no address space fragmentation ever.
- O(1) operations  
  - hash table, LRU cache
- multi-server parallel fetch (can't do in DBI)
LiveJournal and memcached

- 10 unique hosts
  - none dedicated
- 28 instances (512 MB = 1 bucket)
- 30 GB of cached data
- 90-93% hit rate
  - not necessarily 90-93% less queries:
    - FROM foo WHERE id IN (1, 2, 3)
    - would be 3 memcache hits; 1 mysql query
  - 90-93% potential disk seeks?
- 12 GB machine w/ five 2GB instances
  - left-over 'big' machines from our learn-to-scale-out days
What to Cache

- Everything?
- Start with stuff that's hot
- Look at your logs
  - query log
  - update log
  - slow log
- Control MySQL logging at runtime
  - can't
    - (been bugging them)
  - sniff the queries! Net::Pcap
- count
  - add identifiers: SELECT /* name=foo */
Caching Disadvantages

- more code
  - using
  - populating
  - invalidating
  - easy, if your API is clean

- conceptually lame
  - database should do it
    - kinda.
    - database doesn't know object lifetimes
      - putting memcached between app and DB doesn't work

- more stuff to admin
  - but memcached is easy
  - one real option: memory to use
MySQL Persistent Connection Woes

- connections == threads == memory
- max threads
  - limit max memory
- with 10 user clusters:
  - Bob is on cluster 5
  - Alice on cluster 6
  - Do you need Bob's DB handles alive while you process Alice's request?
- Major wins by disabling persistent conns
  - still use persistent memcached conns
  - db hits are rare
  - mysql conns quick (opposed to, say, Oracle)
Software Overview

- BIG-IPs
- Debian
  - Linux 2.4
  - Linux 2.6 (epoll)
- mod_perl
- MySQL
- Perlbal
- MogileFS
Questions?

![Diagram with various components and labels such as User, Internet, Internap, Secure Servers, BigIP, Mail, Proxy Web, NetApp, Web Servers, MemoCache, Global Cluster, and User Clusters.]

- mog1
- mog2
- mog..
Thank you!

Questions to...
brad@danga.com

Slides linked off:
http://www.danga.com/words/