LiveJournal: Behind The Scenes
Scaling Storytime

June 2007
USENIX

Brad Fitzpatrick
brad@danga.com

danga.com / livejournal.com / sixapart.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.

http://danga.com/words/
The plan...

- Refer to previous presentations for more details...
  - http://danga.com/words/
- Questions anytime! Yell. Interrupt.
- Part 0:
  - show where talk will end up
- Part I:
  - What is LiveJournal? Quick history.
  - LJ’s scaling history
- Part II:
  - explain all our software,
  - explain all the moving parts
LiveJournal Backend: Today
(Roughly.)

BIG-IP
- bigip1
- bigip2

perlbal (httpd/proxy)
- proxy1
- proxy2
- proxy3
- proxy4
- proxy5

mod_perl
- web1
- web2
- web3
- web4
- ...
- webN

Memcached
- mc1
- mc2
- mc3
- mc4
- ...
- mcN

Global Database
- master_a
- master_b
- slave1
- slave2
- ...
- slave5

User DB Cluster 1
- uc1a
- uc1b

User DB Cluster 2
- uc2a
- uc2b

User DB Cluster 3
- uc3a
- uc3b

User DB Cluster N
- ucNa
- ucNb

Job Queues (xN)
- jqNa
- jqNb

MogileFS Database
- mog_a
- mog_b

Mogile Trackers
- tracker1
- tracker3

Mogile Storage Nodes
- sto1
- sto2
- ...
- sto8

gearmand
- gearmand1
- gearmandN

"workers"
- gearwrkN
- theschwkn

djabberd
- djabberd
- djabberd

mod_perl
- web1
- web2
- web3
- web4
- ...
- webN

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

- college hobby project, Apr 1999
- 4-in-1:
  - blogging
  - forums
  - social-networking (“friends”)
  - aggregator: “friends page”
    - “friends” can be external RSS/Atom
- 10M+ accounts
- Open Source!
  - server,
  - infrastructure,
  - original clients,
Stuff we've built...
(all production, open source)

- memcached
  - distributed caching
- MogileFS
  - distributed filesystem
- Perlbal
  - HTTP load balancer, web server, swiss-army knife
- gearman
  - LB/HA/coalescing low-latency function call “router”
- TheSchwartz
  - reliable, async job dispatch system
- djabberd
  - the super-extensible everything-is-a-plugin mod_perl/qpsmtpd/ Eclipse of XMPP/Jabber servers
- OpenID
  - federated identity protocol

http://danga.com/words/
“Uh, why?”

- NIH? (Not Invented Here?)
- Are we reinventing the wheel?
Yes.

- We build wheels.
  - ... when existing suck,
  - ... or don’t exist.
Yes.

- We build wheels.
  - ... when existing suck,
  - ... or don’t exist.
Yes.

- We build wheels.
  - ... when existing suck,
  - ... or don’t exist.
Yes.

• We build wheels.
  - ... when existing suck,
  - ... or don’t exist.

(Yes, arguably tires. sshh..)
Part I
Quick Scaling History
Quick Scaling History

• 1 server to hundreds...

• you can do all this with just 1 server!
  - then you’re ready for tons of servers, without pain
  - don’t repeat our scaling mistakes
Terminology

- **Scaling:**
  - **NOT:** “How fast?”
  - **But:** “When you add twice as many servers, are you twice as fast (or have twice the capacity)?”
- **Fast still matters,**
  - 2x faster: 50 servers instead of 100...
    - that’s some good money
  - but that’s not what scaling is.

http://danga.com/words/
Terminology

• “Cluster”
  - varying definitions... basically:
  - making a bunch of computers work together for some purpose
  - what purpose?
    • load balancing (LB),
    • high availability (HA)

• Load Balancing?
• High Availability?
• Venn Diagram time!
  - I love Venn Diagrams
LB vs. HA

Load Balancing

High Availability
LB vs. HA

Load Balancing
- round-robin DNS
- data partitioning
- ...

High Availability
- http reverse proxy
- wackamole
- ...
- LVS heartbeat
- cold/warm/hot spare
- ...

http://danga.com/words/
Favorite Venn Diagram

Times When I’m Truly Happy

Times When I’m Wearing Pants
One Server

- Simple:

![Diagram showing a single server with 'Internet', 'mysql', and 'apache' components connected]
Two Servers

http://danga.com/words/
Two Servers - Problems

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

- 3 webs, 1 db
- Now we need to load-balance!
  - LVS, mod_backhand, whackamole, BIG-IP, Alteon, pound, **Perlbal**, etc, etc..
Four Servers - Problems

- Now I/O bound...
- ... how to use another database?
Five Servers
introducing MySQL replication

- We buy a new DB
- MySQL replication
- Writes to DB (master)
- Reads from both
More Servers

Chaos!
Where we're at....

BIG-IP
bigip1
bigip2

mod_proxy
proxy1
proxy2
proxy3

mod_perl
web1
web2
web3
web4
...
web12

Global Database
master
slave1
slave2
...
slave6

http://danga.com/words/
Problems with Architecture

or, “This don't scale…”

- DB master is SPOF
- Adding slaves doesn't scale well...
  - only spreads reads, not writes!

![Diagram showing scalability issues](http://danga.com/words/)
Eventually...

- databases eventual only writing

http://danga.com/words/
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!
Partition your data!

- Spread your databases out, into “roles”
  - roles that you never need to join between
    - different users
    - or accept you'll have to join in app
- Each user assigned to a numbered HA cluster
- Each cluster has multiple machines
  - writes self-contained in cluster (writing to 2-3 machines, not 6)
User Clusters

http://danga.com/words/
SELECT userid, clusterid FROM user WHERE user='bob'
SELECT userid, clusterid FROM user WHERE user='bob'

userid: 839
clusterid: 2
SELECT userid,
    clusterid
FROM user
WHERE user='bob'

userid: 839
clusterid: 2

SELECT ....
FROM ...
WHERE userid=839 ...
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!!!

http://danga.com/words/
Details

- per-user numberspaces
  - don't use AUTO_INCREMENT
  - PRIMARY KEY (user_id, thing_id)
  - so:
- Can move/upgrade users 1-at-a-time:
  - per-user “readonly” flag
  - per-user “schema_ver” property
  - user-moving harness
    - job server that coordinates, distributed long-lived user-mover clients who ask for tasks
  - balancing disk I/O, disk space

http://danga.com/words/
Shared Storage (SAN, SCSI, DRBD...)

- Turn pair of InnoDB machines into a cluster
  - looks like 1 box to outside world. floating IP.
- One machine at a time mounting fs, running MySQL
- Heartbeat to move IP, {un,}mount filesystem, {stop,start} mysql
  - filesystem repairs,
  - innodb repairs,
  - don’t lose any committed transactions.
- No special schema considerations
- MySQL 4.1 w/ binlog sync/flush options
  - good
  - The cluster can be a master or slave as well
Shared Storage: DRBD

- Linux block device driver
  - “Network RAID 1”
  - Shared storage without sharing!
  - sits atop another block device
  - syncs w/ another machine's block device
    - cross-over gigabit cable ideal. network is faster than random writes on your disks.
- InnoDB on DRBD: HA MySQL!
  - can hang slaves off HA pair,
  - and/or,
  - HA pair can be slave of a master

http://danga.com/words/
MySQL Clustering Options: Pros & Cons

- No magic bullet...
  - Master/Slave
    - doesn’t scale with writes
  - Master/Master
    - special schemas
  - DRBD
    - only HA, not LB
  - MySQL Cluster
    - special-purpose
  - ....

- lots of options!
  - :)
  - :( 

http://danga.com/words/
Part II
Our Software
Caching

- caching's key to performance
  - store result of a computation or I/O for quicker future access (classic space/time trade-off)

- Where to cache?
  - mod_perl/php internal caching
    - memory waste (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
  - implements a dictionary ADT, with network API
- run instances wherever free memory
- two-level hash
  - client hashes* to server,
  - server has internal dictionary (hash table)
- no “master node”, nodes aren’t aware of each other
- protocol simple, XML-free
  - clients: c, perl, java, c#, php, python, ruby, ...
- popular, fast
- scalable
Protocol Commands

- set, add, replace
- delete
- incr, decr
  - atomic, returning new value
http://danga.com/words/
10.0.0.100:11211  
1GB

10.0.0.101:11211  
2GB

10.0.0.102:11211  
1GB
<table>
<thead>
<tr>
<th>IP Address</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.100:11211</td>
<td>1GB</td>
</tr>
<tr>
<td>10.0.0.101:11211</td>
<td>2GB</td>
</tr>
<tr>
<td>10.0.0.102:11211</td>
<td>1GB</td>
</tr>
<tr>
<td>IP Address</td>
<td>Memory</td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
</tr>
<tr>
<td>10.0.0.100:11211</td>
<td>1GB</td>
</tr>
<tr>
<td>10.0.0.101:11211</td>
<td>2GB</td>
</tr>
<tr>
<td>10.0.0.102:11211</td>
<td>1GB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number</th>
<th>IP Address</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10.0.0.100:11211</td>
<td>1GB</td>
</tr>
<tr>
<td>1</td>
<td>10.0.0.101:11211</td>
<td>2GB</td>
</tr>
<tr>
<td>2</td>
<td>10.0.0.102:11211</td>
<td>1GB</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Client

$val = $client->get("foo")
$val = $client->get("foo")
CRC32("foo") % 4 = 2
$val = $client->get("foo")
CRC32("foo") % 4 = 2
connect to server[2]  ("10.0.0.101:11211")
$val = $client->get(“foo”)
CRC32(“foo”) % 4 = 2
connect to server[2] (“10.0.0.101:11211”)
Client

$\text{val} = \text{client}\rightarrow\text{get}(\text{“foo”})$

CRC32(“foo”) % 4 = 2

connect to server[2] ("10.0.0.101:11211")

GET foo  (response)
Client hashing onto a memcached node

• Up to client how to pick a memcached node
• Traditional way:
  - CRC32(<key>) % <num_servers>
  - (servers with more memory can own more slots)
  - CRC32 was least common denominator for all languages to implement, allowing cross-language memcached sharing
  - con: can’t add/remove servers without hit rate crashing
• “Consistent hashing”
  - can add/remove servers with minimal <key> to <server> map changes
memcached internals

- libevent
  - epoll, kqueue...
- event-based, non-blocking design
  - optional multithreading, thread per CPU (not per client)
- slab allocator
- referenced counted objects
  - slow clients can’t block other clients from altering namespace or data
- LRU
- all internal operations O(1)
Perlbal
Web Load Balancing

- BIG-IP, Alteon, Juniper, Foundry
  - good for L4 or minimal L7
  - not tricky / fun enough. :-)
- Tried a dozen reverse proxies
  - none did what we wanted or were fast enough
- Wrote Perlbal
  - fast, smart, manageable HTTP web server / reverse proxy / LB
  - can do internal redirects
    - and dozen other tricks
Perlbal

- Perl
  - parts optionally in C with plugins
- single threaded, async event-based
  - uses epoll, kqueue, etc.
- console / HTTP remote management
  - live config changes
- handles dead nodes, smart balancing
- multiple modes
  - static webserver
  - reverse proxy
  - plug-ins (Javascript message bus.....)
- plug-ins
  - GIF/PNG altering, ....
Perlbal: Persistent Connections
Perlbal: Persistent Connections

- perlbal to backends (mod_perls)
  - know exactly when a connection is ready for a new request
    - no complex load balancing logic: just use whatever's free. beats managing “weighted round robin” hell.
- clients persistent; not tied to a specific backend connection
Perlbal: Persistent Connections

- perlbal to backends (mod_perls)
  - know exactly when a connection is ready for a new request
    - no complex load balancing logic: just use whatever's free. beats managing “weighted round robin” hell.
- clients persistent; not tied to a specific backend connection
Perlbal: Persistent Connections

- perlbal to backends (mod_perls)
  - know exactly when a connection is ready for a new request
    - no complex load balancing logic: just use whatever's free. beats managing “weighted round robin” hell.
- clients persistent; not tied to a specific backend connection
Perlbal: Persistent Connections

- perlbal to backends (mod_perls)
  - know exactly when a connection is ready for a new request
    - no complex load balancing logic: just use whatever's free. beats managing “weighted round robin” hell.
- clients persistent; not tied to a specific backend connection
Perlbal: can verify new backend connections

```c
#include <sys/socket.h>
int listen(int sockfd, int backlog);
```

- connects to backends are often fast, but...
  - are you talking to the kernel’s listen queue?
  - or apache? (did apache accept() yet?)
- send OPTIONs request to see if apache is there
  - Apache can reply to OPTIONS request quickly,
  - then Perlbal knows that conn is bound to an apache process, not waiting in a kernel queue
- Huge improvement to user-visible latency!
  - (and more fair/even load balancing)
Perlbal: multiple queues

- high, normal, low priority queues
- paid users -> high queue
- bots/spiders/suspect traffic -> low queue
Perlbal: cooperative large file serving

- large file serving w/ mod_perl bad...
  - mod_perl has better things to do than spoon-feed clients bytes
Perlbal: cooperative large file serving

- internal redirects
  - mod_perl can pass off serving a big file to Perlbal
    - either from disk, or from other URL(s)
  - client sees no HTTP redirect
  - “Friends-only” images
    - one, clean URL
    - mod_perl does auth, and is done.
    - perlbal serves.

http://danga.com/words/
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)

mod_perl

Perlbale

TUX, thttpd, mogstored

TUX, thttpd, mogstored

http://danga.com/
And the reverse...

- Now Perlbal can buffer uploads as well..
  - Problems:
    - LifeBlog uploading
      - cellphones are slow
    - LiveJournal/Friendster photo uploads
      - cable/DSL uploads still slow
    - decide to buffer to “disk” (tmpfs, likely)
      - on any of: rate, size, time
    - blast at backend, only when full request is in
Palette Altering GIF/PNGs

- based on palette indexes, colors in URL, dynamically alter GIF/PNG palette table, then sendfile(2) the rest.
MogileFS
oMgFileS
MogileFS

- our distributed file system
- open source
- userspace
  - based all around HTTP (NFS support now removed)
- hardly unique
  - Google GFS
  - Nutch Distributed File System (NDFS)
- production-quality
  - lot of users
  - lot of big installs

http://danga.com/words/
MogileFS: Why

• alternatives at time were either:
  – closed, non-existent, expensive, in development, complicated, ...
  – scary/impossible when it came to data recovery
    • new/uncommon/ unstudied on-disk formats
• because it was easy
  – initial version = 1 weekend! :
  – current version = many, many weekends :)

http://danga.com/words/
MogileFS: Main Ideas

- files belong to classes, which dictate:
  - replication policy, min replicas, ...
- tracks what disks files are on
  - set disk's state (up, temp_down, dead) and host
- keep replicas on devices on different hosts
  - (default class policy)
  - No RAID!

- multiple tracker databases
- all share same database cluster (MySQL, etc..)
- big, cheap disks
  - dumb storage nodes w/ 12, 16 disks, no RAID
MogileFS components

- clients
- mogilefsd (does all real work)
- database(s) (MySQL, .... abstract)
- storage nodes
MogileFS: Clients

- tiny text-based protocol
- Libraries available for:
  - Perl
    - tied filehandles
    - MogileFS::Client
      - my $fh = $mogc->new_file("key", [[$class], ...])
  - Java
  - PHP
  - Python?
    - porting to $LANG is be trivial
    - future: no custom protocol. only HTTP
- clients don't do database access
MogileFS: Tracker (mogilefsd)

- The Meat
- event-based message bus
- load balances client requests, world info
- process manager
  - heartbeats/watchdog, respawnner, ...
- Child processes:
  - ~30x client interface ("query" process)
    - interfaces client protocol w/ db(s), etc
  - ~5x replicate
  - ~2x delete
  - ~1x fsck, reap, monitor, ..., ...

http://danga.com/words/
Trackers' Database(s)

- Abstract as of Mogile 2.x
  - MySQL
  - SQLite (joke/demo)
  - Pg/Oracle coming soon?
  - Also future:
    - wrapper driver, partitioning any above
      - small metadata in one driver (MySQL Cluster?),
      - large tables partitioned over 2-node HA pairs
- Recommend config:
  - 2xMySQL InnoDB on DRBD
  - 2 slaves underneath HA VIP
    - 1 for backups
    - read-only slave for during master failover window
MogileFS storage nodes (mogstored)

- HTTP transport
  - GET
  - PUT
  - DELETE
- mogstored listens on 2 ports...
  - HTTP. --server={perlbal,lighttpd,...}
    - configs/manages your webserver of choice.
    - perlbal is default. some people like apache, etc
  - management/status:
    - iostat interface, AIO control, multi-stat() (for faster fsck)
- files on filesystem, not DB
  - sendfile()! future: splice()
  - filesystem can be any filesystem
Large file GET request
Large file GET request

Auth: complex, but quick
Large file
GET request

Auth: complex, but quick

Spoonfeeding: slow, but event-based
manaGer
Manager

dispatches work,
but doesn't do anything useful itself. :)}
Gearman

- system to load balance function calls...
  - scatter/gather bunch of calls in parallel,
  - different languages,
  - db connection pooling,
  - spread CPU usage around your network,
  - keep heavy libraries out of caller code,
  - ...
  - ...

http://danga.com/words/
Gearman Pieces

- gearmand
  - the function call router
  - event-loop (epoll, kqueue, etc)
- workers.
  - Gearman::Worker – perl/ruby
  - register/heartbeat/grab jobs
- clients
  - Gearman::Client[::Async] -- perl
    - also Ruby Gearman::Client
  - submit jobs to gearmand
    - opaque (to server) “funcname” string
    - optional opaque (to server) “args” string
    - opt coalescing key
Gearman Picture

gearmand  gearmand  gearmand
Gearman Picture

gearmand → gearmand

can_do("funcA")

can_do("funcA")

can_do("funcB")

Worker

Worker

http://danga.com/words/
Gearman Picture

Client

```
gearmand
  can_do("funcA")
  can_do("funcB")

Worker
  can_do("funcA")

Worker
```

http://danga.com/words/
Gearman Picture

call("funcA")
can_do("funcA")
can_do("funcA")
can_do("funcB")
can_do("funcA")

Client

Worker

Worker
Gearman Picture

call("funcA")
can_do("funcA")
can_do("funcB")
can_do("funcA")

Client

Client

Worker

Worker

http://danga.com/words/
Gearman Picture

call(“funcA”) 
call(“funcB”) 

Client

Client

Worker

Worker

can_do(“funcA”) 
can_do(“funcB”) 
can_do(“funcA”)
Gearman Protocol

- efficient binary protocol
- No XML
- but also line-based text protocol for admin commands
  - telnet to gearmand and get status
  - useful for Nagios plugins, etc
Gearman Uses

• Image::Magick outside of your mod_perls!
• DBI connection pooling (DBD::Gofer + Gearman)
• reducing load, improving visibility
• "services"
  - can all be in different languages, too!
Gearman Uses, cont..

- running code in parallel
  - query ten databases at once
- running blocking code from event loops
  - DBI from POE/Danga::Socket apps
- spreading CPU from ev loop daemons
- calling between different languages,
- ...

http://danga.com/words/
Gearman Misc

• Guarantees:
  - none! hah! :)
    • please wait for your results.
    • if client goes away, no promises
  - all retries on failures are done by client
    • but server will notify client(s) if working worker goes away.
• No policy/conventions in gearmand
  - all policy/meaning between clients <-> workers
• ...

http://danga.com/words/
Sick Gearman Demo

• Don’t actually use it like this... but:

```perl
use strict;
use DMap qw(dmap);
DMap->set_job_servers("sammy", "papag");

my @foo = dmap { "$_ = " . `hostname` } (1..10);

print "dmap says:\n @foo";

$ ./dmap.pl
dmap says:
  1 = sammy
  2 = papag
  3 = sammy
  4 = papag
  5 = sammy
  6 = papag
  7 = sammy
  8 = papag
  9 = sammy
 10 = papag
```

http://danga.com/words/
Gearman Summary

• Gearman is sexy.
  - especially the coalescing
• Check it out!
  - it's kinda our little unadvertised secret
    • oh crap, did I leak the secret?

http://danga.com/words/
TheSchwartz
The Schwartz

- Like gearman:
  - job queuing system
  - opaque function name
  - opaque “args” blob
  - clients are either:
    - submitting jobs
    - workers
- But unlike gearman:
  - **Reliable** job queueing system
  - not low latency
    - fire & forget (as opposed to gearman, where you wait for result)
- *currently* library, not network service
The Schwartz Primitives

- insert job
- “grab” job (atomic grab)
  - for 'n' seconds.
- mark job done
- temp fail job for future
  - optional notes, rescheduling details..
- replace job with 1+ other jobs
  - atomic.
- ...

http://danga.com/words/
• backing store:
  - a database
  - uses Data::ObjectDriver
    • MySQL,
    • Postgres,
    • SQLite,
    • ....

• but HA: you tell it @dbs, and it finds one to insert job into
  - likewise, workers foreach (@dbs) to do work
TheSchwartz uses

- outgoing email (SMTP client)
  - millions of emails per day
  - TheSchwartz::Worker::SendEmail
  - Email::Send::TheSchwartz
- LJ notifications
  - ESN: event, subscription, notification
    - one event (new post, etc) -> thousands of emails, SMSes, XMPP messages, etc...
- pinging external services
- atomstream injection
- ..... 
- dozens of users
- shared farm for TypePad, Vox, LJ
gearmand + TheSchwartz

- gearmand: not reliable, low-latency, no disks
- TheSchwartz: latency, reliable, disks
- In TypePad:
  - TheSchwartz, with gearman to fire off TheSchwartz workers.
    - disks, but low-latency
    - future: no disks, SSD/Flash, MySQL Cluster
djabberd

• Our Jabber/XMPP server
  • powers our “LJ Talk” service
• S2S: works with GoogleTalk, etc
• perl, event-based (epoll, etc)
• done 300,000+ conns
• tiny per-conn memory overhead
  - release XML parser state if possible
• everything is a hook
  - not just auth! like, everything.
    - auth,
    - roster,
    - vcard info (avatars),
    - presence,
    - delivery,
    - inter-node cluster delivery,
  - ala mod_perl, qpsmtpd, etc.

• async hooks
  - hooks phases can take as long as they want before they answer, or decline to next phase in hook chain...
  - we use Gearman::Client::Async
Thank you!

Questions to:
brad@danga.com

Software:
http://danga.com/
http://code.sixapart.com/
Bonus Slides

• if extra time
Data Integrity

- Databases depend on fsync()
  - but databases can't send raw SCSI/ATA commands to flush controller caches, etc
- fsync() almost never works work
  - Linux, FS' (lack of) barriers, raid cards, controllers, disks, ...
- Solution: test! & fix
  - disk-checker.pl
    - client/server
    - spew writes/fsyncs, record intentions on alive machine, yank power, checks.
Persistent Connection Woes

- connections == threads == memory
  - My pet peeve:
    - want connection/thread distinction in MySQL!
    - w/ max-runnable-threads tunable

- max threads
  - limit max memory/concurrency

- DBD::Gofer + Gearman
  - Ask

- Data::ObjectDriver + Gearman