Operations at Twitter

John Adams
Twitter Operations
John Adams / @netik

- Early Twitter employee
- Lead engineer: Application Services (Apache, Unicorn, SMTP, etc...)
- Keynote Speaker: O’Reilly Velocity 2009
- Previous companies: Inktomi, Apple, c|net
What changed since Velocity ’09?

- Specialized services for social graph storage
- More efficient use of Apache
- Unicorn (Rails)
- More servers, more LBs, more humans
- Memcached partitioning - dedicated pools+hosts
- More process, more science.
210 employees

sharding humans is difficult.
160K
Registered Apps

source: twitter.com internal
700M Searches/Day

source: twitter.com internal, includes api based searches
65M Tweets per day
(~750 Tweets/sec)

source: twitter.com internal
2,940 TPS
Japan Scores!

3,085 TPS
Lakers Win!
Operations

- Support the site and the developers
- Make it performant
- Capacity Planning (metrics-driven)
- Configuration Management
- Improve existing architecture and plan for future
Nothing works the first time.

- Scale site using best available technologies
- Plan to build everything more than once.
- Most solutions work to a certain level of scale, and then you must re-evaluate to grow.
- We’re doing this now.
Operations Mantra

Find Weakest Point

Metrics + Logs + Science = Analysis
Operations Mantra

Find Weakest Point → Take Corrective Action

Metrics + Logs + Science = Analysis
Operations Mantra

Find Weakest Point

Take Corrective Action

Move to Next Weakest Point

Metrics + Logs + Science = Analysis

Process

Repeatability
Monitoring

- Twitter graphs and reports critical metrics in as near to real time as possible
- If you build tools against our API, you should too.
- Use this data to inform the public
  - dev.twitter.com - API availability
  - status.twitter.com
Sysadmin 2.0

- Don’t be a “systems administrator” anymore.
- Combine statistical analysis and monitoring to produce meaningful results
- Make decisions based on data
Profiling

- Low-level
- Identify bottlenecks inside of core tools
  - Latency, Network Usage, Memory leaks
- Methods
  - Network services: tcpdump + tcpdstat, yconalyzer
  - Introspect with Google perftools
Data Analysis

- Instrumenting the world pays off.
- “Data analysis, visualization, and other techniques for seeing patterns in data are going to be an increasingly valuable skill set. Employers take notice!”

 Rails

- Front-end (Scala/Java back-end)
- Not to blame for our issues. Analysis found:
  - Caching + Cache invalidation problems
  - Bad queries generated by ActiveRecord, resulting in slow queries against the db
  - Garbage Collection issues (20-25%)
- Replication Lag
Analyze

- Turn data into information
- Where is the code base going?
- Are things worse than they were?
  - Understand the impact of the last software deploy
  - Run check scripts during and after deploys
- Capacity Planning, not Fire Fighting!
Logging

- Syslog doesn’t work at high traffic rates
- No redundancy, no ability to recover from daemon failure
- Moving large files around is painful
- Solution:
  - Scribe to HDFS with LZO Compression
Dashboard

- “Criticals” view
- Smokeping/MRTG
- Google Analytics
- Not just for HTTP 200s/SEO
- XML Feeds from managed services
Whale Watcher

- Simple shell script, Huge Win
- Whale = HTTP 503 (timeout)
- Robot = HTTP 500 (error)
- Examines last 60 seconds of aggregated daemon / www logs
- “Whales per Second” > $W_{\text{threshold}}$
- Thar be whales! Call in ops.
Change Management

- Reviews in Reviewboard
- Puppet + SVN
  - Hundreds of modules
  - Runs constantly
- Reuses tools that engineers use
Deploy Watcher

Sample window: 300.0 seconds
First start time:
Mon Apr  5 15:30:00 2010 (Mon Apr  5 08:30:00 PDT 2010)
Second start time:
Tue Apr  6 02:09:40 2010 (Mon Apr  5 19:09:40 PDT 2010)

PRODUCTION APACHE: ALL OK
PRODUCTION OTHER: ALL OK
WEB049 CANARY APACHE: ALL OK
WEB049 CANARY BACKEND SERVICES: ALL OK
DAEMON031 CANARY BACKEND SERVICES: ALL OK
DAEMON031 CANARY OTHER: ALL OK
Deploys

- Block deploys if site in error state
- Graph time-of-deploy along side server CPU and Latency
- Display time-of-last-deploy on dashboard
- Communicate deploys in Campfire to teams

^^ last deploy times ^^
Feature “Darkmode”

- Specific site controls to enable and disable computationally or IO-Heavy site function
- The “Emergency Stop” button
- Changes logged and reported to all teams
- Around 90 switches we can throw
- Static / Read-only mode
subsystems
loony

- Central machine database (MySQL)
- Python, Django, Paraminko SSH
  - Paraminko - Twitter’s OSS SSH Library
- Ties into LDAP
- When data center sends us email, machine definitions built in real-time
- On demand changes with run
Murder

- Bittorrent based replication for deploys (Python w/libtorrent)
- ~30-60 seconds to update >1k machines
- Gets work list from loony
- Legal P2P
memcached

- Network Memory Bus isn’t infinite
- Evictions make the cache unreliable for important configuration data (loss of darkmode flags, for example)
- Segmented into pools for better performance
- Examine slab allocation and watch for high use/eviction rates on individual slabs using *peep*. Adjust slab factors and size accordingly.
request flow

Load Balancers

Apache

Rails (Unicorn)

Flock  Kestrel  Memcached

MySQL  Cassandra

Monitoring  Daemons  Mail Servers
Unicorn Rails Server

- Connection push to socket polling model
- Deploys without Downtime
- Less memory and 30% less CPU
- Shift from ProxyPass to Proxy Balancer
  - Apache’s not better than ngnix.
  - It’s the proxy.
Asynchronous Requests

- Inbound traffic consumes a worker
- Outbound traffic consumes a worker
- The request pipeline should not be used to handle 3rd party communications or back-end work.
- Move long running work to daemons when possible.
Kestrel

- Works like memcache (same protocol)
- \textbf{SET} = enqueue  \mid \textbf{GET} = dequeue
- No strict ordering of jobs
- No shared state between servers
- Written in Scala.
Daemons

• Many different types at Twitter.
• Old way: One Daemon per type
• New Way: One Daemon, many jobs
• Daemon Slayer
• A Multi Daemon that does many different jobs, all at once.
Flock DB

- Shard the social graph through Gizzard
- Billions of edges
- MySQL backend
- Open Source (available now)
Disk is the new Tape.

- Social Networking application profile has many $O(n^y)$ operations.
- Page requests have to happen in < 500mS or users start to notice. Goal: 250-300mS
- Web 2.0 isn’t possible without lots of RAM
- What to do?
Caching

- We’re the real-time web, but lots of caching opportunity
- Most caching strategies rely on long TTLs (>60 s)
- Separate memcache pools for different data types to prevent eviction
- Optimize Ruby Gem to libmemcached + FNV Hash instead of Ruby + MD5
- Twitter largest contributor to libmemcached
Caching

• “Cache Everything!” not the best policy
• Invalidating caches at the right time is difficult.
• Cold Cache problem; What happens after power or system failure?
• Use cache to augment db, not to replace
MySQL Challenges

- Replication Delay
  - Single threaded replication = pain.
- Social Networking not good for RDBMS
  - N x N relationships and social graph / tree traversal - we have FlockDB for that
- Disk issues
  - FS Choice, *noatime*, scheduling algorithm
Database Replication

- Major issues around users and statuses tables
- Multiple functional masters (FRP, FWP)
- Make sure your code reads and writes to the write DBs. Reading from master = slow death
  - Monitor the DB. Find slow / poorly designed queries
  - Kill long running queries before they kill you (mkill)
In closing...

- Use configuration management, no matter your size
- Make sure you have logs of everything
- Plan to build everything more than once
- Instrument everything and use science.
- Do it again.
Thanks!

- We support and use Open Source
- [http://twitter.com/about/opensource](http://twitter.com/about/opensource)
- Work at scale - We’re hiring.
- @jointheflock