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, 2010
- Previous companies: Inktomi, Apple, c|net
Operations

- Support the site and the developers
- Make it performant
- Capacity Planning (metrics-driven)
- Configuration Management
- Improve existing architecture
What changed since 2009?

- Specialized services for social graph storage, shards
- More efficient use of Apache
- Unicorn (Rails)
- More servers, more LBs, more humans
- Memcached partitioning - dedicated pools+hosts
- More process, more science.
>165M Users

source: blog.twitter.com
700M Searches/Day

source: twitter.com internal, includes api based searches
90M Tweets per day
(~1000 Tweets/sec)

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

3,085 TPS
Lakers Win!
#newtwitter is an API client
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.
- This is a continual process.
UNIX friends fail at scale

- Cron
  - Add NTP, and many machines executing the same thing cause “micro” outages across the site.

- Syslog
  - Truncation, data loss, aggregation issues

- RRD
  - Data rounding over time
Operations Mantra

Find Weakest Point

Metrics + Logs + Science = Analysis

Friday, November 12, 2010
Operations Mantra

Find Weakest Point -> Take Corrective Action

Metrics + Logs + Science = Analysis

Process

Friday, November 12, 2010
Operations Mantra

Find Weakest Point

Metrics + Logs + Science = Analysis

Take Corrective Action

Process

Move to Next Weakest Point

Repeatability
Sysadmin 2.0 (Devops)

- Don’t be a just a sysadmin anymore.
- Think of Systems management as a programming task (puppet, chef, cfengine...)
- No more silos, or lobbing things over the wall
- We’re all on the same side. Work Together!
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!”

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
Profiling

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

Curve-fitting for capacity planning
(R, fityk, Mathematica, CurveFit)

status_id

unsigned int (32 bit)
Twitpocalypse

signed int (32 bit)
Twitpocalypse

$r^2=0.99$
Configuration Management

- Start automated configuration management EARLY in your company.
- Don’t wait until it’s too late.
- Twitter started within the first few months.
Puppet

- Puppet + SVN
- Hundreds of modules
- Runs constantly
- Post-Commit idiot checks
- No one logs into machines
- Centralized Change
loony

- Accesses central machine database (MySQL)
- Python, Django, Paraminko SSH
- Ties into LDAP
- Filter and list machines, find asset data
- On demand changes with *run*
Murder

- Bittorrent based replication for deploys (Python w/libtorrent)
- ~30-60 seconds to update >1k machines
- Uses our machine database to find destination hosts
- Legal P2P
Issues with Centralized Management

- Complex Environment
- Multiple Admins
- Unknown Interactions
- Solution: 2nd set of eyes.
Process through Reviews

Review Board beta

Summary: publish review: dns change to point search round robin to backlink interfaces
Updated 4 days, 2 hours ago

Submitter: Josh Fraser
Branch:
Bugs:
Change Number: None

Reviewers
Groups: operations
People: jayed, jeremy, jin, rudy, joh
Repository: twitter-ops

Description:
publish review: dns change to point search round robin to backlink interfaces

Testing Done:

Ship it!

John Adams

I think this is ok, please make sure internal search doesn't explode.
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
Scribe

- Twitter patches
  - LZO compression and Hadoop (HDFS) writing
- Useful for logging lots of data
- Simple data model, easy to extend
- Log locally, then scribe to aggregation nodes
Hadoop for Ops

- Once the data’s scribed to HDFS you can:
  - Aggregate reports across thousands of servers
  - Produce application level metrics
  - Use map-reduce to gain insight into your systems.
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!
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.
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
request flow

Load Balancers

Apache

Rails (Unicorn)

FlockDB

Kestrel

Memcached

MySQL

Cassandra

Monitoring

Daemons

Mail Servers
Many limiting factors in the request pipeline

**Apache**
- Worker Model
- MaxClients
- TCP Listen queue depth

**Rails** (unicorn)
- 2:1 oversubscribed to cores

**Varnish** (search)
- # threads

**Memcached**
- # connections

**MySQL**
- # db connections

Friday, November 12, 2010
Unicorn Rails Server

- Connection push to socket polling model
- Deploys without Downtime
- Less memory and 30% less CPU
- Shift from ProxyPass to Proxy Balancer
  - mod_proxy_balancer lies about usage
- Race condition in counters patched
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
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.
Decomposition

- Take application and decompose into services
- Admin the services as separate units
- Decouple the services from each other
Asynchronous Requests

- Executing work during the web request is expensive
- The request pipeline should not be used to handle 3rd party communications or back-end work.
  - Move work to queues
  - Run daemons against queues
Thrift

- Cross-language services framework
- Originally developed at Facebook
- Now an Apache project
- Seamless operation between C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, Smalltalk, OCaml (phew!)
Kestrel

- Works like memcache (same protocol)
- SET = enqueue  |  GET = dequeue
- No strict ordering of jobs
- No shared state between servers
- Written in Scala. Open Source.
Daemons

- Many different types at Twitter.
- # of daemons have to match the workload
- Early Kestrel would crash if queues filled
- “Seppaku” patch
- Kill daemons after n requests
- Long-running leaky daemons = low memory
Daemons

- 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

- Gizzard sharding framework
- Billions of edges
- MySQL backend
- Open Source

http://github.com/twitter/gizzard
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 “real time”, but still 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, as
- 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

- We have many MySQL servers
- Increasingly used more and more as key/value store
- Many instances spread out through the Gizzard sharding framework
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)
Key Points

- Databases not always the best store.
- Instrument everything.
- Use metrics to make decisions, not guesses.
- Don’t make services dependent
- Process asynchronously when possible
Questions?
Thanks!

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