Billions of Hits: Scaling Twitter

John Adams
Twitter Operations
John Adams  @netik

- Early Twitter employee (mid-2008)
- Lead engineer: Outward Facing Services (Apache, Unicorn, SMTP), Auth, Security
- Keynote Speaker: O’Reilly Velocity 2009, 2010
- Previous companies: Inktomi, Apple, c|net
Growth.
752%

2008 Growth

source: comscore.com - (based only on www traffic, not API)
1358%

2009 Growth

source: comscore.com - (based only on www traffic, not API)
12th
most popular

source: alexa.com (global ranking)
55M Tweets per day

(640 TPS/sec, 1000 TPS/sec peak)

source: twitter.com internal
600M

Searches/Day

source: twitter.com internal

Wednesday, May 5, 2010
Operations

- What do we do?
  - Site Availability
  - Capacity Planning (metrics-driven)
  - Configuration Management
  - Security
  - Much more than basic Sysadmin
What have we done?

- Improved response time, reduced latency
- Less errors during deploys (Unicorn!)
- Faster performance
- Lower MTTD (Mean time to Detect)
- Lower MTTR (Mean time to Recovery)
- We are an advocate to developers
Operations Mantra

Find Weakest Point

Take Corrective Action

Move to Next Weakest Point

Metrics + Logs + Science = Analysis

Process

Repeatability

Wednesday, May 5, 2010
Make an **attack** plan.

<table>
<thead>
<tr>
<th>Symptom</th>
<th>Bottleneck</th>
<th>Vector</th>
<th>Solution</th>
</tr>
</thead>
</table>

Wednesday, May 5, 2010
Make an **attack** plan.

<table>
<thead>
<tr>
<th>Symptom</th>
<th>Bottleneck</th>
<th>Vector</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Congestion</td>
<td>Network</td>
<td>HTTP Latency</td>
<td>More LB’s</td>
</tr>
<tr>
<td>Timeline Delay</td>
<td>Storage</td>
<td>Update Delay</td>
<td>Better algorithm</td>
</tr>
<tr>
<td>Status Growth</td>
<td>Database</td>
<td>Delays</td>
<td>Flock Cassandra</td>
</tr>
<tr>
<td>Updates</td>
<td>Algorithm</td>
<td>Latency</td>
<td>Algorithms</td>
</tr>
</tbody>
</table>

Wednesday, May 5, 2010
Finding Weakness

- Metrics + Graphs
  - Individual metrics are irrelevant
  - We aggregate metrics to find knowledge
- Logs
- SCIENCE!
Monitoring

- Twitter graphs and reports critical metrics in as near real time as possible
- If you build tools against our API, you should too.
  - RRD, other Time-Series DB solutions
  - Ganglia + custom gmetric scripts
- dev.twitter.com - API availability
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!

Wednesday, May 5, 2010
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!”

A New World for Admins

- You’re not just a sysadmin anymore
- Analytics - Graph what you can
- Math, Analysis, Prediction, Linear Regression
- For everyone, not just big sites.
- You can do *fantastic* things.
Forecasting

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

unsigned int (32 bit)
Twitpocolypse

status_id

signed int (32 bit)
Twitpocolypse

$r^2 = 0.99$
### External API Dashboard

![API Status Dashboard](http://dev.twitter.com/status)

#### Current Performance and Availability History

<table>
<thead>
<tr>
<th>Service / Website</th>
<th>Apr 14</th>
<th>Apr 13</th>
<th>Apr 12</th>
<th>Apr 11</th>
<th>Apr 10</th>
<th>Apr 9</th>
<th>Apr 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>account/verify_credentials</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>direct_messages</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>favorites</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>friends/ids</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>lists</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>search</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>statuses/home_timeline</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>statuses/home_timeline (OAuth 1.0a)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>statuses/retweets</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>stream</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>users/search</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>users/show</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

- ✔️ Service is operating normally
- 🔄 Performance issues
- 🚨 Service disruption
- 🕵️ Informational message

---

周三，五月五日，2010年

[http://dev.twitter.com/status](http://dev.twitter.com/status)
What’s a Robot?

- Actual error in the Rails stack (HTTP 500)
- Uncaught Exception
- Code problem, or failure / nil result
- Increases our exception count
- Shows up in Reports
- We’re on it!
What’s a Whale?

- HTTP Error 502, 503
- Twitter has a hard and fast five second timeout
- We’d rather fail fast than block on requests
- Death of a long-running query (mkill)
- Timeout
Whale Watcher

- Simple shell script,
  - MASSIVE WIN by @ronpepsi
- 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
WEB0049 CANARY APACHE: ALL OK
WEB0049 CANARY BACKEND SERVICES: ALL OK
DAEMON0031 CANARY BACKEND SERVICES: ALL OK
DAEMON0031 CANARY OTHER: ALL OK
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 60 switches we can throw
- Static / Read-only mode
unicorn

- A single socket Rails application Server
- Workers pull worker, vs. Apache pushing work.
- Zero Downtime Deploys (!)
- Controlled, shuffled transfer to new code
- Less memory, 30% less CPU
- Shift from mod_proxy_balancer to mod_proxy_pass
  - HAProxy, Nginx wasn’t any better. really.
Rails

- Mostly only for front-end and mobile
- Back end mostly Java, Scala, and Pure Ruby
- Not to blame for our issues. Analysis found:
  - Caching + Cache invalidation problems
  - Bad queries generated by ActiveRecord, resulting in slow queries against the db
  - Queue Latency
  - Replication Lag
memcached

- memcached isn’t perfect.
- Memcached SEGVs hurt us early on.
- Evictions make the cache unreliable for important configuration data (loss of darkmode flags, for example)
- Network Memory Bus isn’t infinite
- Segmented into pools for better performance
Loony

- Central machine database (MySQL)
- Python, Django, Paraminko SSH
- Paraminko - Twitter OSS (@robey)
- Ties into LDAP groups
- When data center sends us email, machine definitions built in real-time
Murder

- @lg rocks!
- Bittorrent based replication for deploys
- ~30-60 seconds to update >1k machines
- P2P - Legal, valid, Awesome.
Kestrel

- @robey
- Works like memcache (same protocol)
- \texttt{SET} = enqueue \, | \, \texttt{GET} = dequeue
- No strict ordering of jobs
- No shared state between servers
- Written in Scala.
Asynchronous Requests

- Inbound traffic consumes a unicorn worker
- Outbound traffic consumes a unicorn worker
- The request pipeline should not be used to handle 3rd party communications or back-end work.
- Reroute traffic to daemons
Daemons

- Daemons touch every tweet
- Many different daemon types at Twitter
- Old way: One daemon per type (Rails)
  - New way: Fewer Daemons (Pure Ruby)
- Daemon Slayer - A Multi Daemon that could do many different jobs, all at once.
Disk is the new Tape.

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

- We’re the real-time web, but lots of caching opportunity. You should cache what you get from us.
- 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 now largest contributor to libmemcached
MySQL

- Sharding large volumes of data is hard
- Replication delay and cache eviction produce inconsistent results to the end user.
- Locks create resource contention for popular data
MySQL Challenges

- Replication Delay
- Single threaded. Slow.
- Social Networking not good for RDBMS
  - N x N relationships and social graph / tree traversal
- Disk issues (FS Choice, noatime, scheduling algorithm)
Relational Databases not a Panacea

- Good for:
  - Users, Relational Data, Transactions

- Bad:
  - Queues. Polling operations. Social Graph.

- You don’t need ACID for everything.
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)
Flock

- Scalable Social Graph Store
- Sharding via Gizzard
- MySQL backend (many.)
- 13 billion edges, 100K reads/second
- Recently Open Sourced
Cassandra

- Originally written by Facebook
- Distributed Data Store
- @rk’s changes to Cassandra Open Sourced
- Currently double-writing into it
- Transitioning to 100% soon.
Lessons Learned

• Instrument everything. Start graphing early.
• Cache as much as possible
• Start working on scaling early.
• Don’t rely on memcache, and don’t rely on the database
• Don’t use mongrel. Use Unicorn.
Join Us!
@jointheflock
Q&A
Thanks!

- @jointheflock
- http://twitter.com/jobs
- Download our work
  - http://twitter.com/about/opensource