Futures and Promises:

Lessons in Concurrency Learned at Tumblr

QCon NY 2012

tumblr.

Overview

- Tumblr stats
- Macro, Mecro and Micro Concurrency
- Platform History
- Motherboy and the Dashboard
- Lessons Learned
- Q & A

Monthly page views

18,000,000,000

600M page views per day

60-80M new posts per day

Peak rate of 50k requests & 2k posts per second

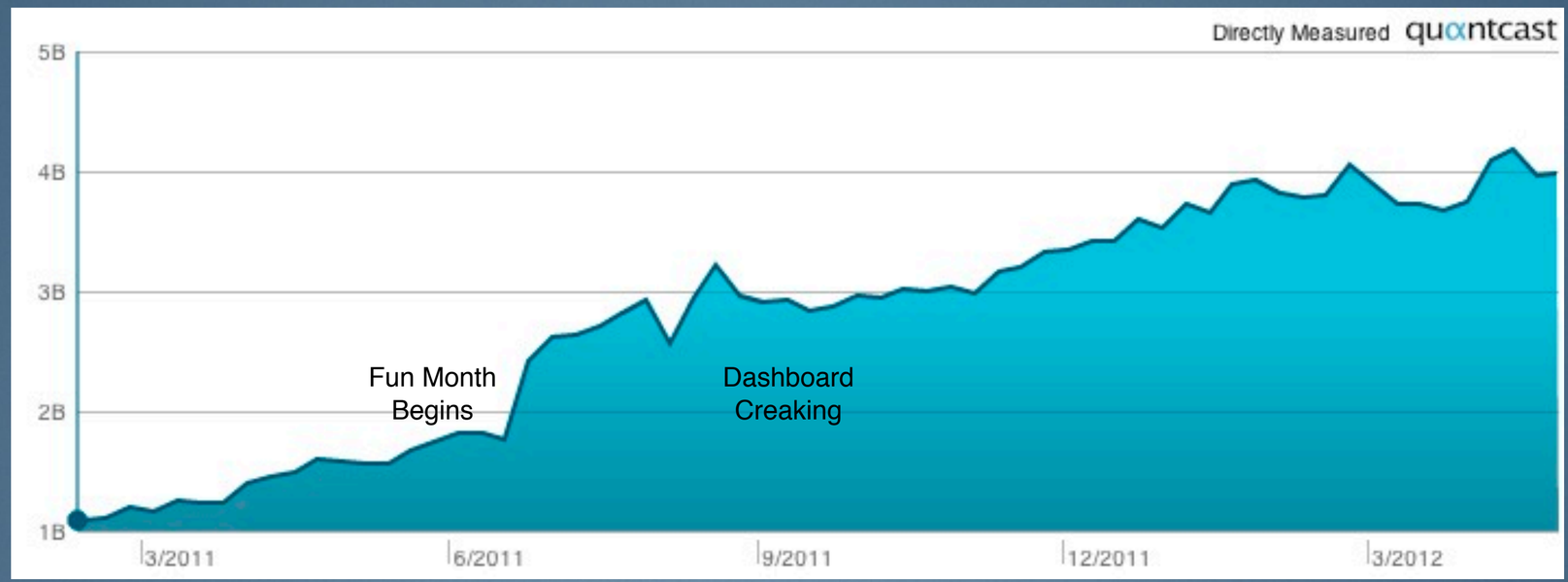
Totals: 22B posts, 53M blogs, 45M users

24 in Engineering (1 CAE, 7 PLAT, 5 SRE, 7 SYS, 4 INFRA)

More than 50% of traffic is international

Traffic growth

Weekly Page Views

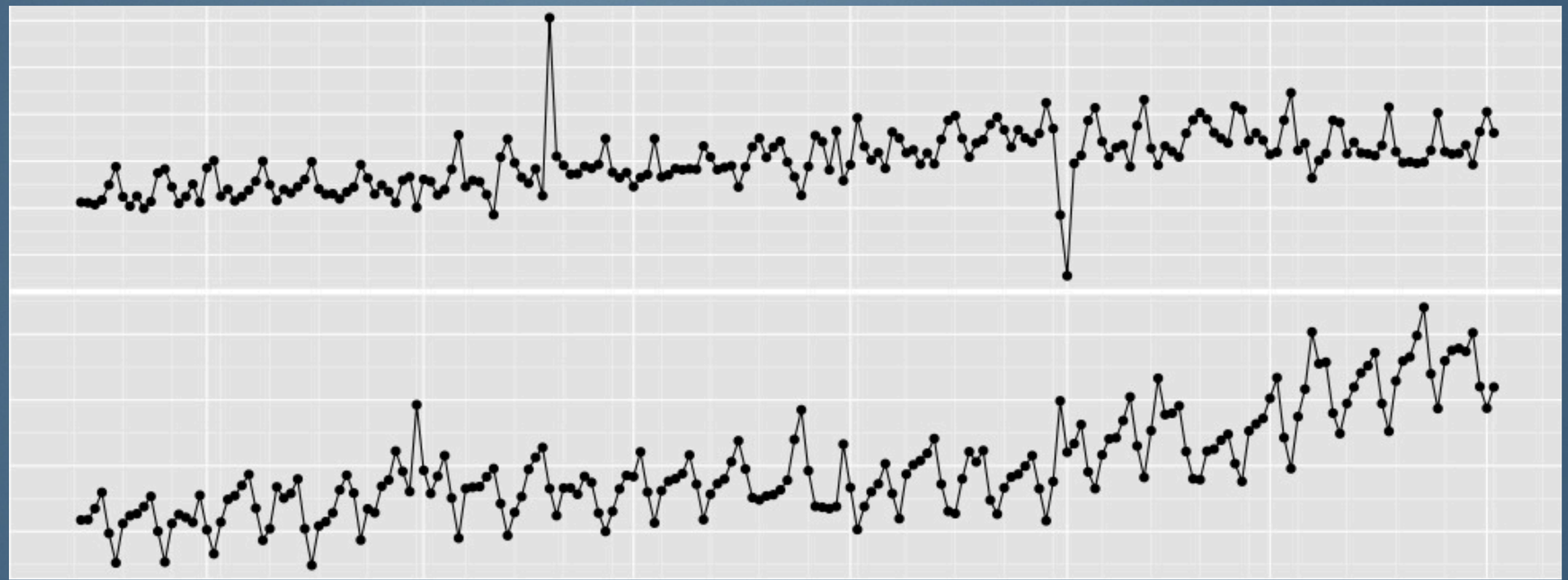


People

Posts and followers drive growth

Posts

Average Followers



Concurrency Styles

Macro

- Team
- Routing
- Load Balancers
- Servers

Micro

- Couroutines
- Event loop based
- Event based actors
- STM

Mecro (In Between)

- Shared-nothing (LAMP)
- Thread based actors, Threads in general
- Processes

Platform History

LAMP Stack

Sharded MySQL

C/HTTP Services

2007 2008 2009 2010 2011 2012

Scala/Thrift Services

2007 to mid-2010 Monolithic PHP Application

- Only two-four developers through mid-2010, made sense
- Started out at rackspace, eventually moved to The Planet (we still have stuff at rackspace)
- Lots of memcache
- Functional database pools
- Hired first ops guy
- Grew to 400 or so servers, team of 4-5 doing development

mid-2010 - early-2011 MySQL Sharding, Services

- Hired a few software engineers (5 through ~April-2011) and a couple more ops guys
- Post dataset outgrew single database instance
- Started doing single dataset MySQL sharding for posts, postref
- Implemented libevent based C/HTTP service for providing unique post ID's
- Implemented libevent based C/HTTP service for handling notifications (staircar)
- The Planet merged with SoftLayer, we stayed. 800 servers.

mid-2011 Scala+Thrift Services

- Hired a few more software engineers (20 total through ~September-2011) and few more ops guys
- More post shards (15), too many functional pools (24)
- Rolled out first Scala/Thrift based service (motherboy)
- We migrated between SoftLayer datacenters after running out of power

mid-2011 to now - Distributed Systems

- Hired a few more engineers (32 total across all teams)
- Many post shards (45), 12 functional pools
- Rolled out many more Scala/Thrift based services (gob, parmesan, ira, buster, wentworth, oscar, george, indefatigable, collins, fibr)
- Started evaluating go as a simpler alternative to some backend services
- Driving more through Tumblr firehose (parmesan)
- We started building out Tumblr owned and operated POP's to support traffic growth. 1200 servers.

Dashboard

Issues

- 70% of traffic is destined for the dashboard
- No dashboard persistence, scatter-gather
- Rendered on-demand
- Not especially cachable
- Lack of persistence ➜ difficult to add features

Time-Based MySQL Sharding

- Current shard always ‘hot’
- Poor fault isolation
- Single threaded replication
- Write concurrency
- Slave lag causes inconsistent dashboard views

Motherboy

Motivations

- Awesome Arrested Development reference
- The dashboard is going to die (load)
- Inconsistent dashboard experience

Goals

- Durability
- Availability, fault-isolation
- Consistency
- Multi-datacenter tenancy
- Features (read/unread, tags, etc)

Non-Goals

- Absolute ordering of posts across dashboards
- 100% availability for a cell

Motherboy Architecture

Goals

- Users have a persistent dashboard, stored in their inbox
- Selective materialization of dashboard when appropriate (ala feeding frenzy)
- Users partitioned into cells, users home to an inbox within a cell
- Inbox writes are asynchronous and distributed
- Inboxes are eventually consistent

Failure Handling

- Reads can be fulfilled on-demand from any cell
- Writes can catch up when a cell is back online

Motherboy Data

Assumptions

- 60M posts per day
- 500 followers on average
- 2k posts/second ➜ 1M writes/second
- 40 bytes per row, 24 byte row key
- Compression factor of 0.4
- Replication factor of 3.5 (3 plus scratch space)

Data Set Growth (Posts*Followers)

	Rows	Size
Day	30 Billion	447 GB
Week	210 Billion	3.1 TB
Month	840 Billion	12.2 TB
Year	10 Trillion	146 TB

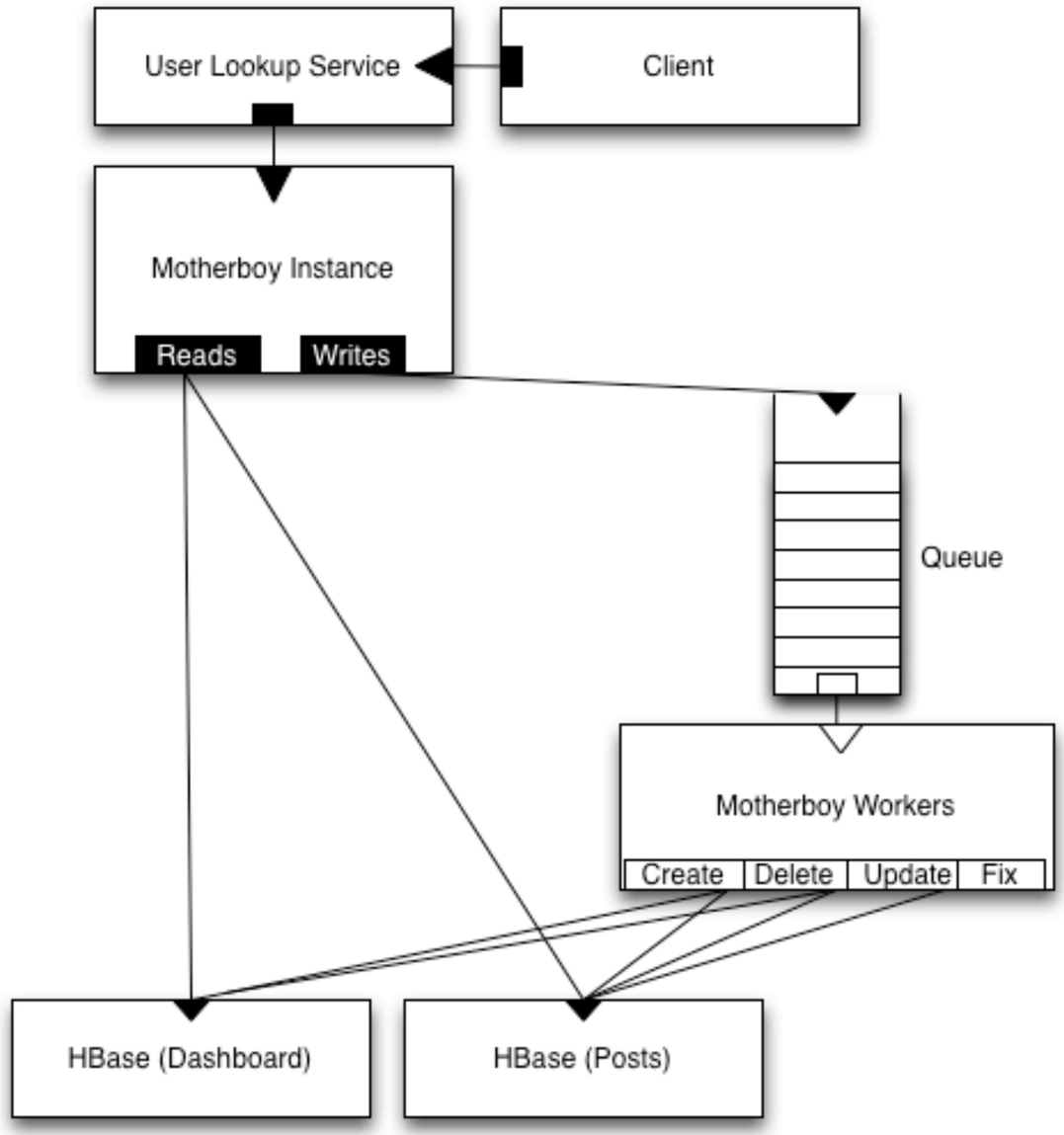
Data Set Growth (Replicated)

	Size
Day	1.5 TB
Week	10.7 TB
Month	42.8 TB
Year	513 TB

Motherboy 0 - Getting our feet wet

Implementation

- Two JVM Processes
  - Server - accept writes (and reads) from clients via thrift, put on write queue. Finagle event loop
  - Worker - process writes from queue, store in HBase. Scala 2.8 actors
- 6 node HBase cluster
- 1 server process, 6 worker processes (one on each datanode)
- User lookup service dictates cell to interact with
- **Goal**: Understand the moving pieces


User Lookup Service
Client
Motherboy Instance
Reads
Writes
Queue
Motherboy Workers
Create
Delete
Update
Fix
HBase (Dashboard)
HBase (Posts)

Motherboy 0 Takeaways

Overall

- Poor automation ➜ Hard to recover in event of failure
- No test infrastructure ➜ Hard to evaluate effects of performance tuning
- Write (client) times unpredictable, variable
- Poor software performance

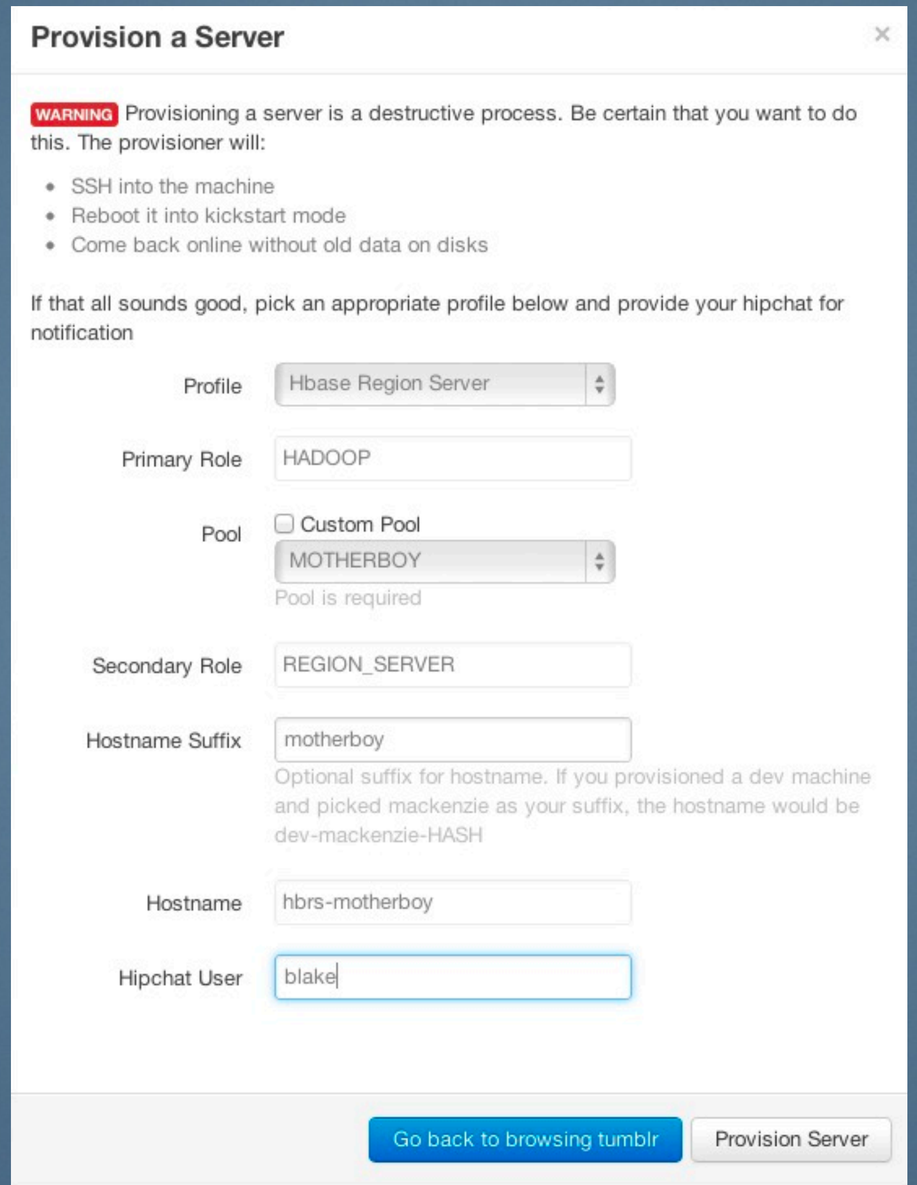
Concurrency

- Thread management was problematic due to client variability
- Network played a huge role in performance, blips would create a large backlog
- Context-switching was killing us, too many threads working on each node
- Actor tuning in 2.8 wasn't well documented
- Thread per actor + thread per connection was a lot of overhead. All IO bound!

Motherboy 1 Preparation

- Build out fully automated cluster provisioning - handle hardware failures more quickly
- Build out a cluster monitoring infrastructure - respond to failures more quickly
- Build out cluster trending/visualization - forecast capacity issues or MTBF

Automated provisioning



Automated monitoring

Current Network Status
Last Updated: Sun Apr 29 02:30:24 EDT 2012
Updated every 90 seconds
Nagios® Core™ 3.3.1 - www.nagios.org
Logged in as *tumblr*

View Status Overview For All Host Groups
View Service Status Detail For This Host Group
View Host Status Detail For This Host Group
View Status Summary For This Host Group
View Status Grid For This Host Group

Host Status Totals

Up	Down	Unreachable	Pending
15	0	0	0

All Problems	All Types
0	15

Service Status Totals

Ok	Warning	Unknown	Critical	Pending
156	0	5	6	1

All Problems	All Types
11	168

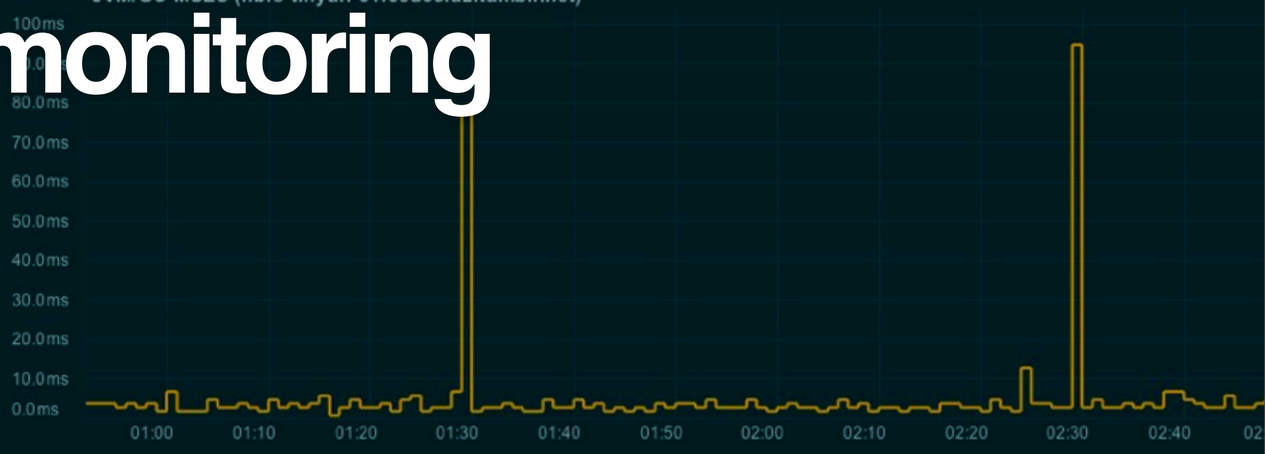
Service Overview For Host Group 'POOL:URL_SHORTENER'

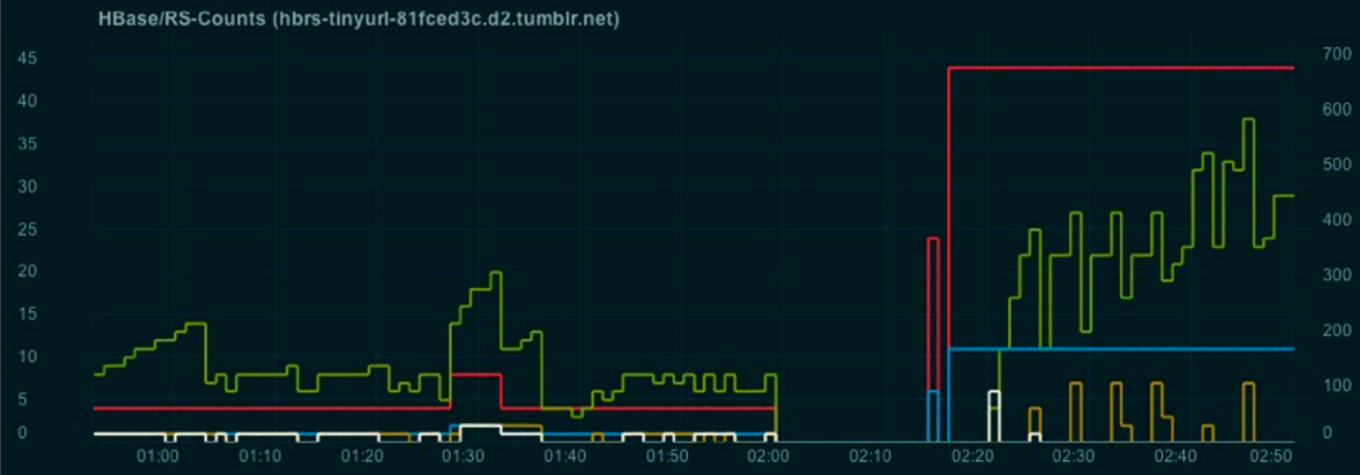
POOL:URL_SHORTENER (POOL:URL_SHORTENER)

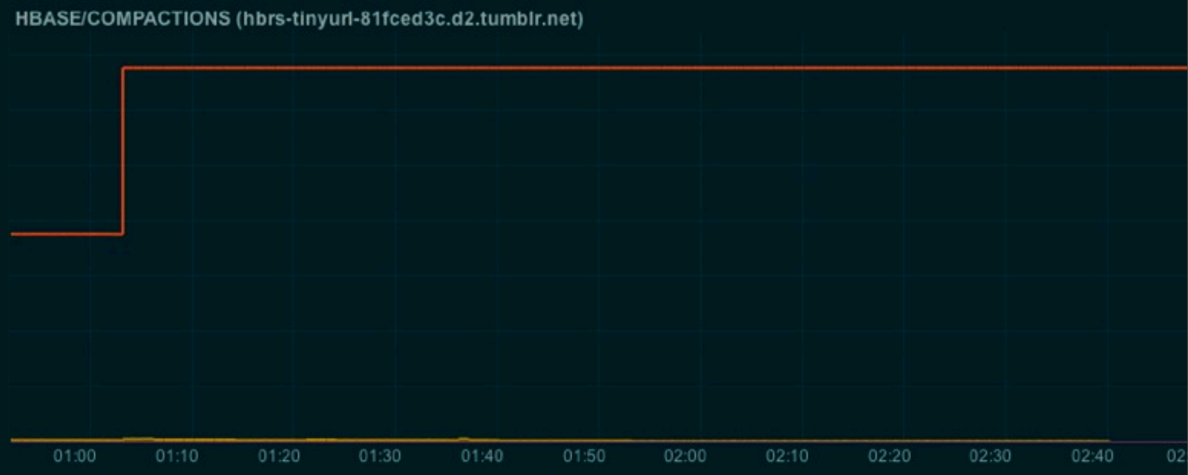
Host	Status	Services	Actions
hbm-tinyurl-1fa64c4c.d2.tumblr.net	UP	10 OK	
hbrs-tinyurl-32d119dc.d2.tumblr.net	UP	10 OK	
hbrs-tinyurl-3e0a1105.d2.tumblr.net	UP	10 OK	
hbrs-tinyurl-51a00c8d.d2.tumblr.net	UP	10 OK	
hbrs-tinyurl-7c82f19e.d2.tumblr.net	UP	10 OK	
hbrs-tinyurl-81fced3c.d2.tumblr.net	UP	10 OK	
hbrs-tinyurl-fd3bf333.d2.tumblr.net	UP	10 OK	
nn-tinyurl-b29d0515.d2.tumblr.net	UP	10 OK 1 CRITICAL	

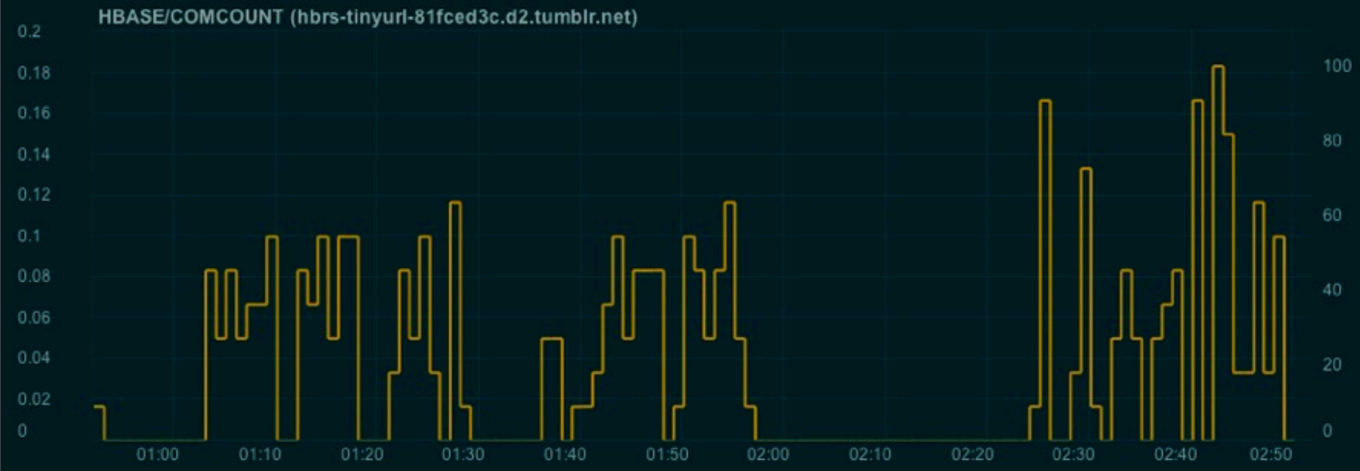
#fibr

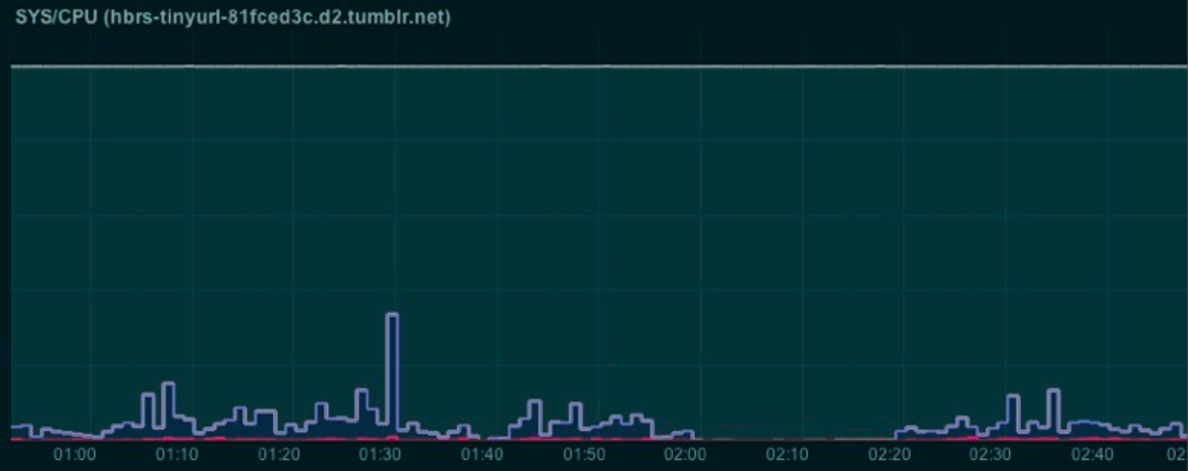
Automated monitoring











Unify Process Interface

Management

- Processes all started, stopped the same way
- No servlet container, wrapped with daemon
- Deployment with capistrano
- Processes look the same to ops/dev
- HTTP interface to manage processes

Monitoring & Trending

- Common stats
- App specific stats
- HTTP graphs, access

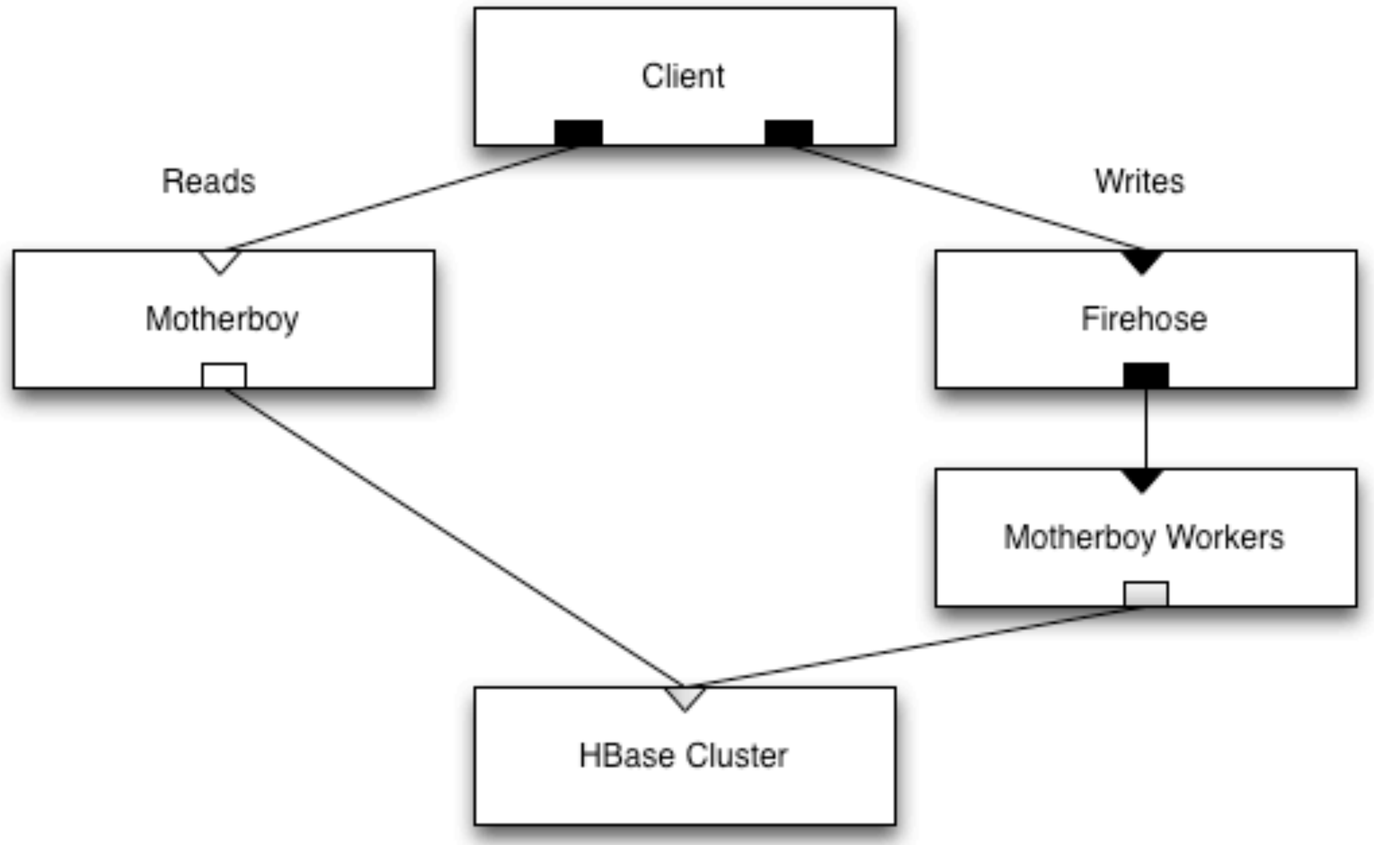
Motherboy 1 Prep Takeaways

- Unified monitoring/trending interfaces across all apps
- 10k data points per second, 500k at peak
- PHP application stats via Thrift to collectd
- JVM stats via Ostrich/OpenTSDB plugin
- 1200 servers reporting via collectd
- 864M data points per day to 10 node OpenTSDB cluster

Motherboy 1 - Try Again

Implementation

- Three JVM Processes
  - Firehose - accept writes from clients via thrift, put on persistent queue. Finagle event loop
  - Server - accept reads from clients via thrift. Finagle event loop
  - Worker - consume from firehose, store in HBase. Finagle service, fixed size thread pool
- 10 node HBase cluster
- 10 server process, 6 worker processes (one on each datanode)
- Dropped the discovery mechanism for now, replace with LB and routing map
- **Goal**: Optimize for performance


Client
Reads
Writes
Motherboy
Firehose
Motherboy Workers
HBase Cluster

Motherboy 1 Takeaways

HBase Tuning

- Disable major compactions
- Disable auto-splits, self manage
- `regionserver.handler.count`
- `hregion.max.filesize`
- `hregion.memstore.mslab.enabled`
- `block.cache.size`
- Table design is super important, a few bytes matter

Concurrency

- IO bound workloads continued to make thread management problematic
- Monitoring much better, but indicates we have over provisioned servers
- Eliminating actors simplified troubleshooting and tuning

Overall

- Still too hard to test tuning changes
- Failure recovery is manual and time consuming
- Distributed failures now a possibility, hard to track

Motherboy 2 Preparation

- **Goal:** Make testing easy
- Fixture data
- Load test tools
- Historical data, look for regressions
- Create a baseline!
- Test different versions, patches, schema, compression, split methods, configurations

Testing setup

Overview

- Standard test cluster: 6 RS/DN, 1HM, 1NN, 3ZK
- Drive load via separate 23 node Hadoop cluster
- Test dataset created for each application
- Results recorded and annotated in OpenTSDB

Motherboy Testing

- 60M posts, 24GB LZO compressed
- 51 mappers
- Map posts into tuple of Long's - 24 byte row key and 16 bytes of data in single CF
- Workers write to HBase cluster as fast as possible
- 10k users, 51 mappers make dashboard requests as fast as possible

Sample test results

Scenario	Test Time	Posts per Second	Avg. Request Time	Avg. CPU Load
Baseline	2:49:46	5801.4	3.7ms	15%
Handler Count = 100	2:39:57	6157.5	3.4ms	12%
Disabled Auto Flush	1:58:53	8284.5	3.4ms	7%
Pre-Created Regions	30:08	32684.3	2.2ms	3%

Conclusions

- More region servers, better throughput
- Less round-trips, better throughput
- Not earth shattering
- If experimenting is easy, people will do it

Tests must be designed for intended workload

Tracing Setup

Problem

- Tiered services create interesting failure scenarios
- Client -> A -> B (DEATH), Client says A failed

Toolbox

- We used a lot of scribe and thrift
- We use a lot of HBase/Hadoop

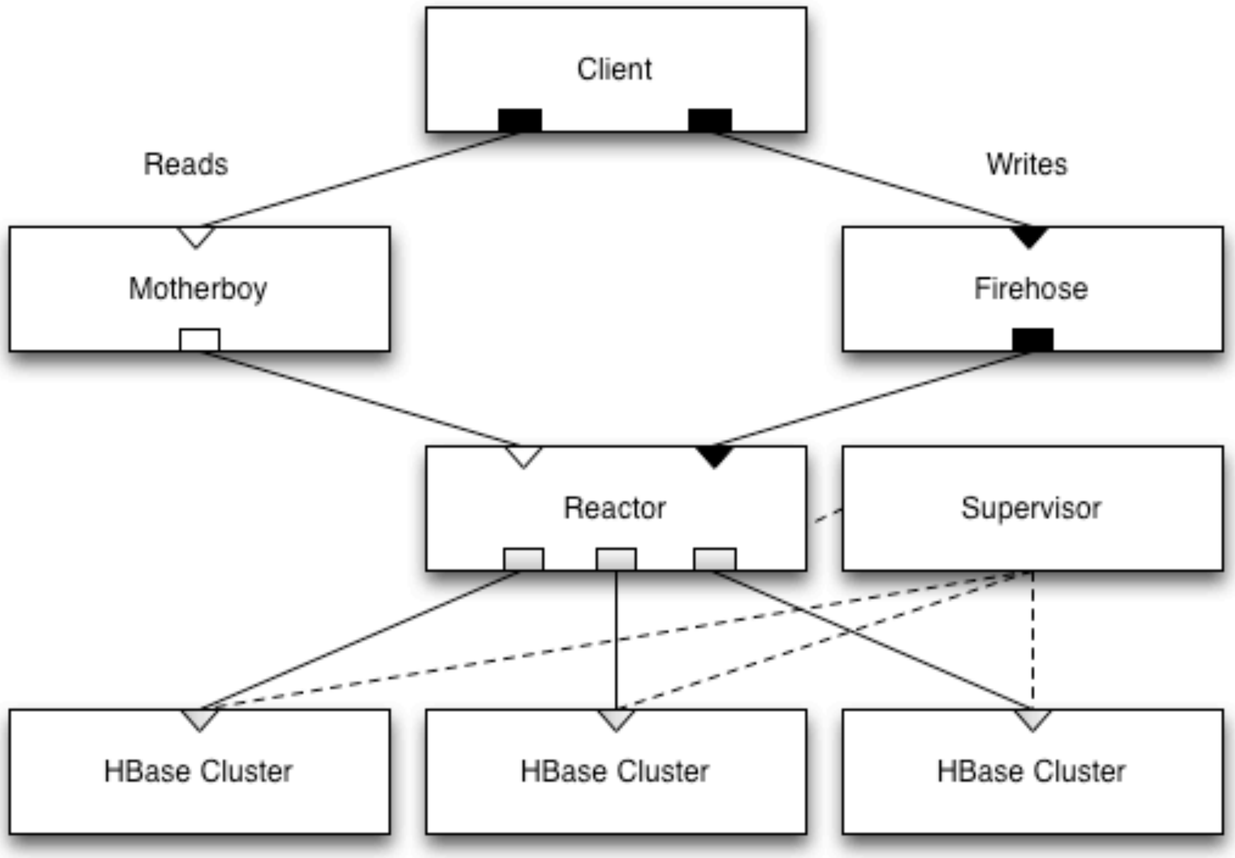
Solution

- Dapper style tracing for services
- Emitted via scribe
- Aggregated into HBase cluster

Motherboy 2 - In progress

Implementation

- Four JVM Processes
  - Firehose - accept writes from clients via thrift, put on persistent queue. Finagle event loop
  - Server - accept reads from clients via thrift. Finagle event loop
  - Reactor - manages talking to different backends, sharding. Elastic pool of finagle services
  - Supervisor - manages reactors to published SLA, brings up/tears-down reactors to meet SLA
- Elastic worker pool, configurable backend persistence
- Dropped the discovery mechanism, no longer needed
- **Goal**: Into production


Client
Reads
Writes
Motherboy
Firehose
Reactor
Supervisor
HBase Cluster
HBase Cluster
HBase Cluster

Lessons learned

1. There are many concurrency mechanisms available, use the appropriate tool for the job
2. The value of automation can't be understated
3. At scale, nothing works as advertised
4. Operational tooling is the single most important thing you can do up front
5. Where in the stack you make concurrency choices impacts who can help tune, deploy, monitor, etc

Questions?

Blake Matheny: bmatheny@tumblr.com