Counting events reliably with storm & riak

Frank Schröder - eBay Classifieds Group Amsterdam

marktplaats.nl

Classifieds

Admarkt

Pay-per-Click ads for professional sellers

Show ad if relevant and budget available

```
budget =
budget - clicks*cpc;
```

```
if budget == 0
hide(ad);
```

```
func onClick(ad) {
    ad.relevancy++;
}
```

Manage & find ads

Keep track of budget

Analytics & Statistics

ebay classifieds group

Count

clicks & impressions

70-100M/day

@ 2K/sec

accurate

(must not lose a click)

quick

detect fraud & unwanted events

scalable

Additional Requirements

System too slow => add more boxes

Must work across multiple data centers and for multiple customers

Faster decisions & better analytics

A/B testing

Current system (2y old)

Current System

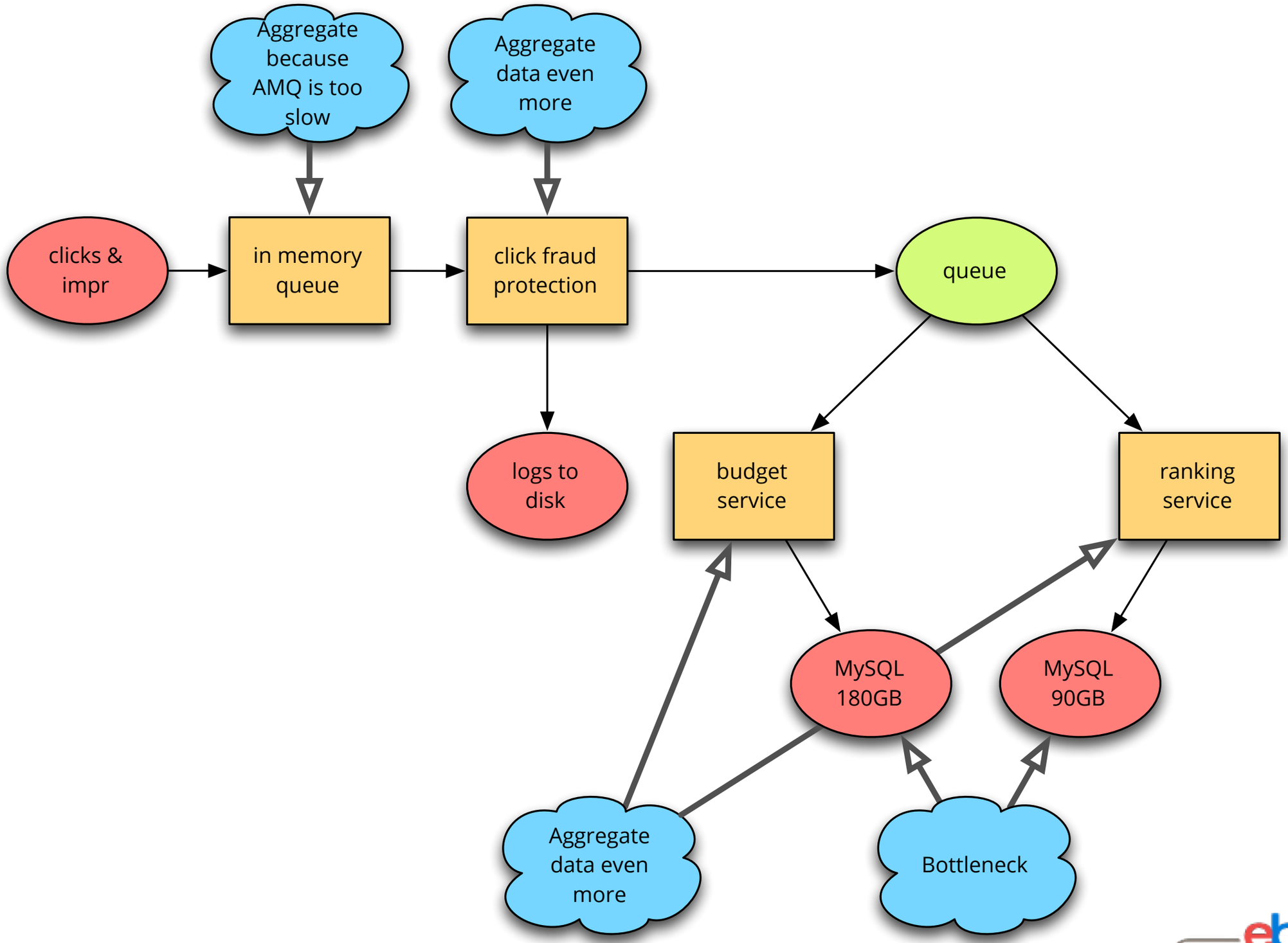
SOA architecture

Queue and aggregate events

Write log records into MySQL

Write log records to disk for archival

ebay classifieds group



ebay classifieds group

To determine the performance of an ad (clicks & impressions) we query the database

```
SELECT SUM(clicks*cpc) WHERE id=123;
```

MySQL has to perform SUM and GROUP BY queries on two large tables (500M rows , 180GB data & idx)

Current System

To keep up with the usage we have to keep the database in RAM

The way we use the database is the bottleneck

Potential Solutions

Better Hardware

Current DB servers already have 270GB RAM. Need 3 per data center

Already seeing random RAM errors

Probably not the end of the line but close to what makes sense

Sharding

Finding a good shard key is difficult (hotspots)

Sharding should be handled by the app

Re-balancing is a pain (different hotspots)

Would need ~60 MySQL servers instead of 3
-> DBA would not be happy
-> Schema changes are a nightmare

Precompute

Precompute all required values

Fast, but less flexible (no ad-hoc queries)

Stop recomputing things we already know,
i.e. yesterdays data

Problems: Bootstrapping, over-/undercounts,
failures, replays, new requirements

Storm & Riak

Architecture

Storm

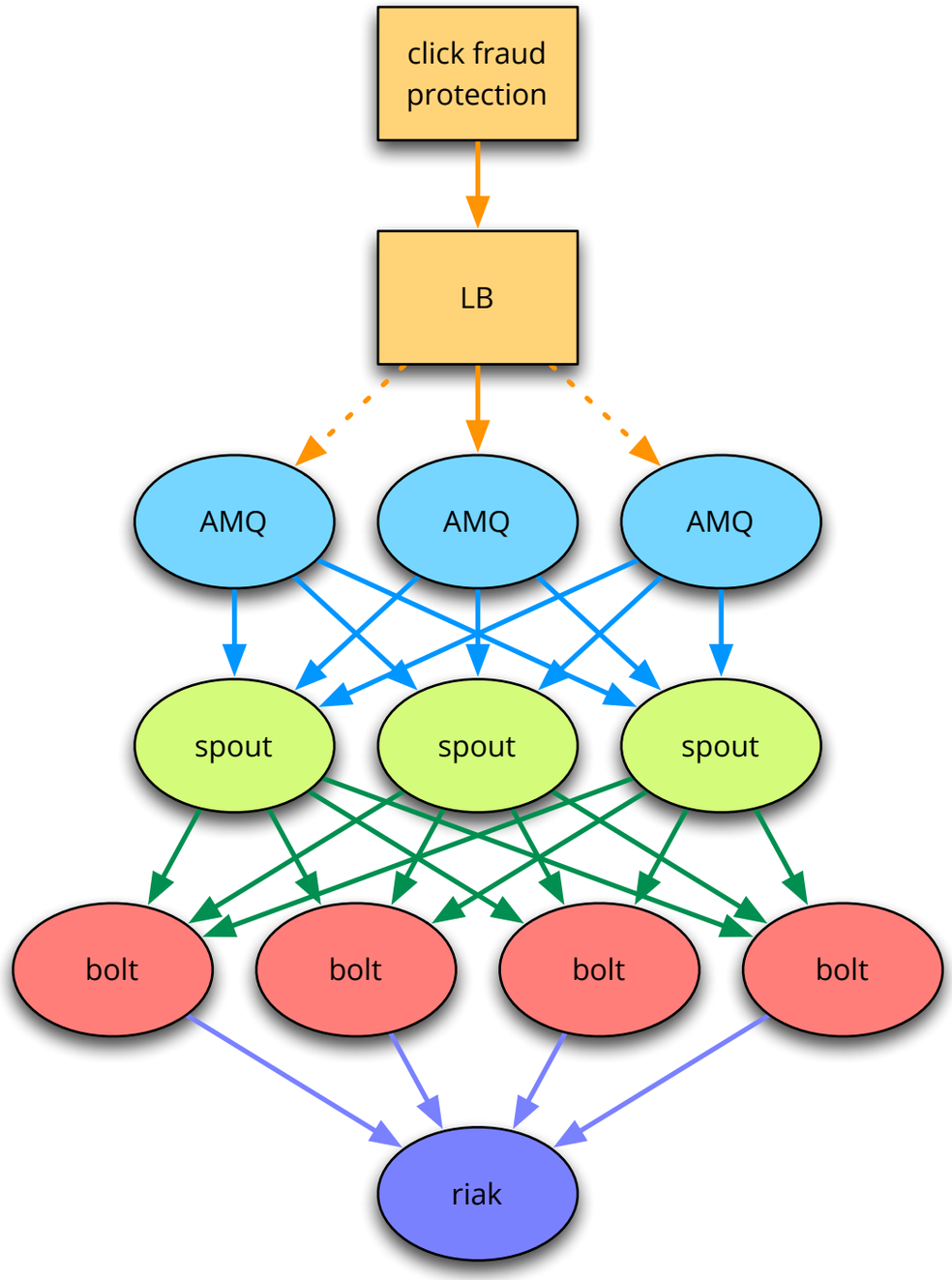
Real-time computation framework from Twitter

Orchestrated by Zookeeper

Stream based producer-consumer topologies

Nice properties for concurrent processing

Storm
Software Architecture



ebay classifieds group

Storm

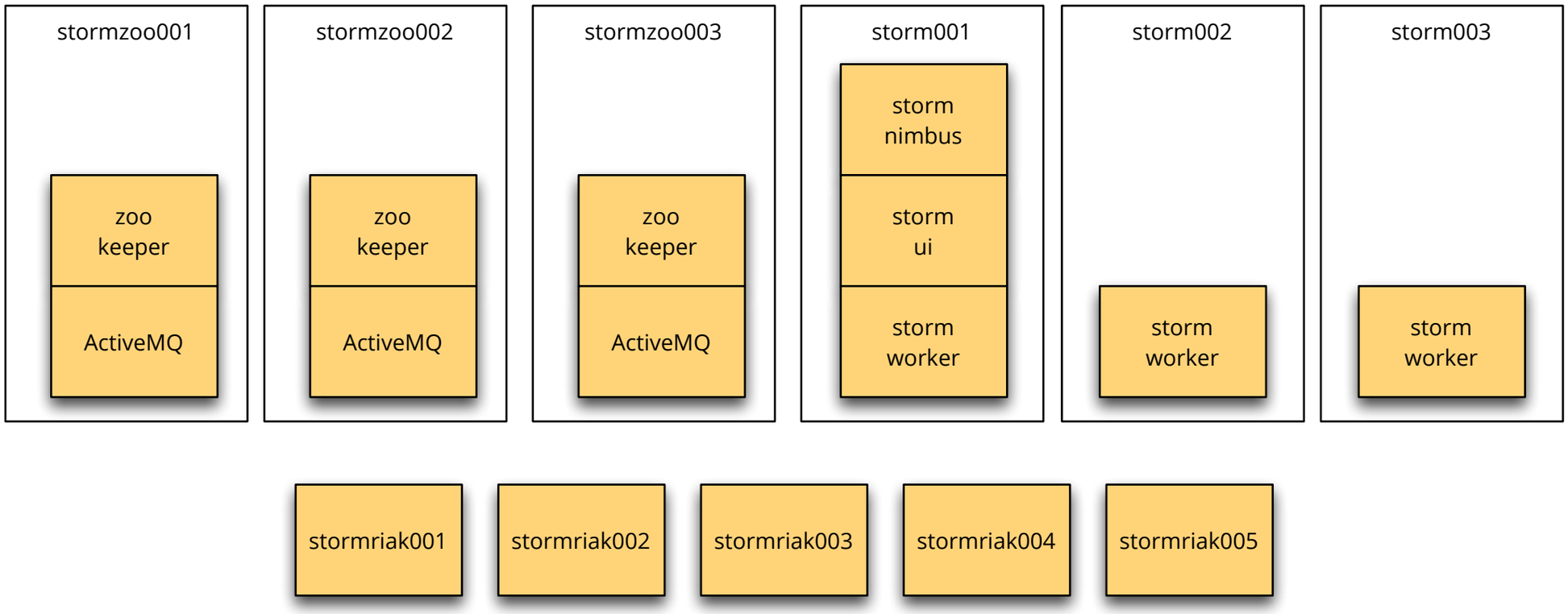
Spouts emit tuples (Producer)

Bolts consume tuples and can emit them, too (Consumer & Producer)

Storm worker = Java VM,
Bolt = 1 thread in a worker

Concurrency of spouts and bolts is configurable

Storm
Hardware Architecture



ebay classifieds group

Admarkt click-counter

Input are log lines (same as archive)

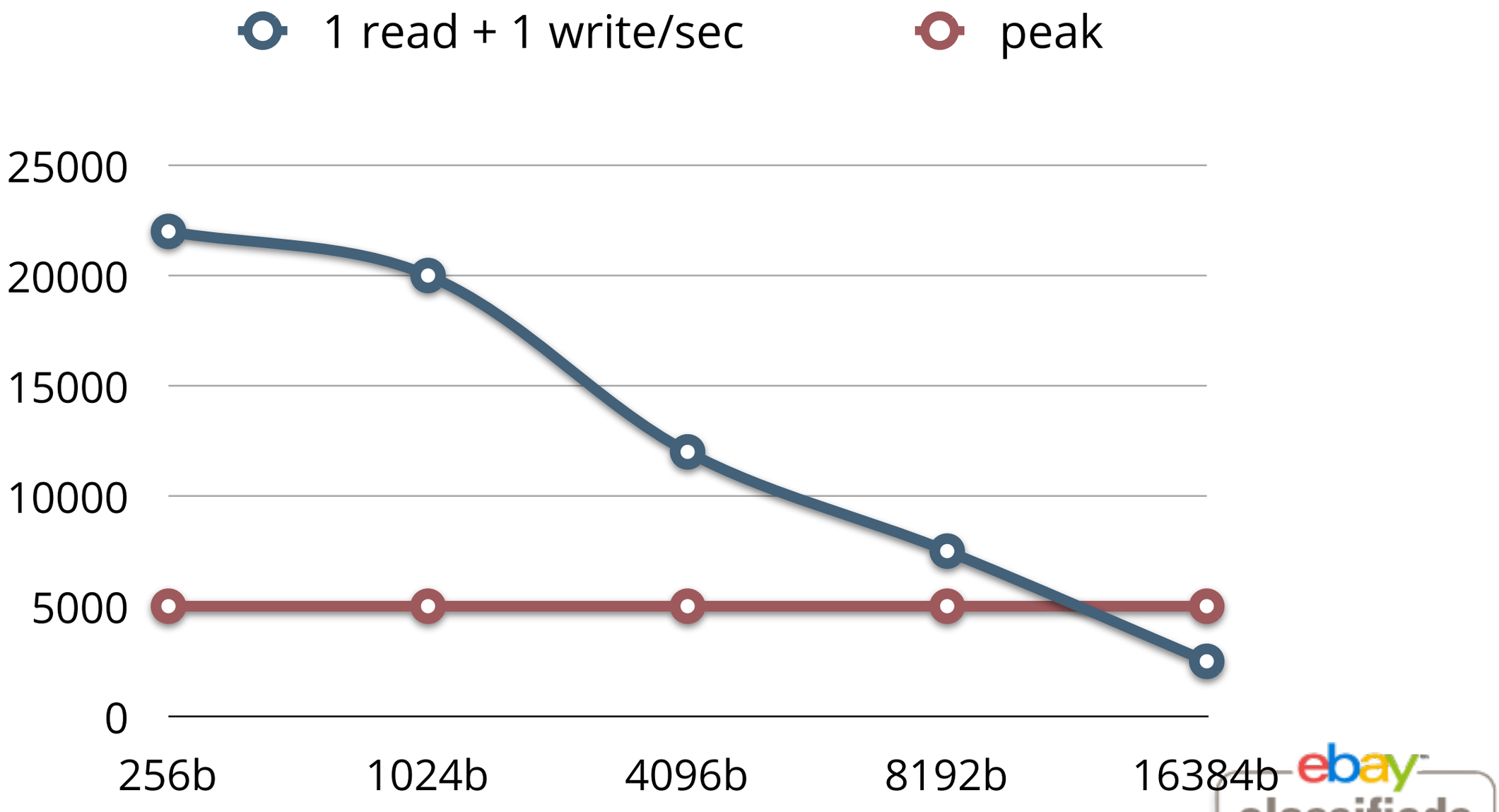
Service sends log lines via LB to one of the ActiveMQs

ActiveMQs don’t do HA, the app does

For replays just put the logs on the queue

How fast can we write to riak?

Riak Performance



ebay classifieds group

Conclusion:

Document size is important

Pre-Computing Values

Pre-Computing How?

A) Logs become the sole truth

B) Remember which log lines we have already seen

Pre-Computing How?

A) Logs become the sole truth ✔

B) Remember which log lines we have already seen **? ? ?**

Pre-Computing
1. Timestamps

event timestamp < last timestamp:
we have seen it already

Milliseconds are not accurate enough

NTP clock skew

Replaying and bootstrapping does not work since you can't tell an old from a replayed event

Pre-Computing
2. Counters

event id < last id: we have seen it already

How do you build a distributed, reliable, sorted counter? How do you handle service restarts? How can this not be the SPOF of the service? No idea …

Replaying and bootstrapping does not work for the same reasons as before

Pre-Computing
3. Hashes

Event hash in current document:
we have seen it already

Bootstrapping and replaying just works

Overcounting cannot happen

On failure just replay the logs

but ...

Pre-Computing
3. Hashes

… how many hashes do we have?

Pre-Computing
3. Hashes

70M events per day -> 70M hashes

500K live ads -> 1400 events per day/ad

But a handful of outliers get
40.000 events / hour - each

sha1: 40 chars, md5: 32 chars, crc32: 8 chars

Collisions?

ebay classifieds group

Pre-Computing
3. Hashes

sha1: 1400*40 = 56KB

md5: 1400*32 = 45KB

crc32: 1400*8 = 11KB

One day is too much - and this is average

Pre-Computing
3. Hashes

Why do we store the hashes?

Usually events are played forward in chronological order

Only during replay and failure we need older hashes

Keep only “last couple” of hashes in the document -> hot set

Pre-Computing
3. Hashes

Keep only the current hour in the main document (hot set)

=> Hash must be unique per ad per hour -> Should take care of collisions. Should ...

At nn:00 archive the hashes into a separate document

Keep documents with older event hashes for as long we want to be able to replay

Pre-Computing
3. Hashes

But with riak we don’t have TX …

Pre-Computing TX with Riak

First write archive doc and keep events of last hour in hot set

Then on next event prune events from previous hour from hot set

What if there is no next event? Storm has tick tuples

Pre-Computing Serialization

Document size is important -> Serialization makes a difference

Kryo isn’t as fast as you might think

JSON isn’t as bad as you might think

Custom beats everything by a wide margin

Maintainability is important, too

Pre-Computing Serialization

Maintainability is important, too

You can look at JSON but not at protobuf

Schema evolution via Content-Type headers

Pre-Computing Persistence

Average ad has average number of hashes

Can be written in real-time

Outliers have orders of magnitude more hashes

More hashes -> bigger docs & more writes
-> kills riak (even a handful of them)

Pre-Computing Persistence

Simple back pressure rule saves us

Small doc -> write immediately

Doc above threshold -> defer write a bit
(the bigger the doc the longer, up to 30 sec)

-> Volatile docs receive lots of events during defer period
(~ 5K events, saves 4,999 writes)

ebay classifieds group

What's next?

Next steps

More counters -> more hashes

gzip+protobuf saves another 30% and (probably) produces less garbage -> less GC

Constantly recompute static data from logs -> less hashes and errors get flushed out quickly. (Hadoop or Storm)

Next steps

Partition data at the source with Kafka

Memcached for saving even more writes

Span storage cluster across N>2 data centers

Questions?

Thank you

frschroeder@ebay.com