Counting events reliably with storm & riak

Frank Schröder - eBay Classifieds Group
Amsterdam

marktplaats.nl

Classifieds

Admarkt

Pay-per-click ads for professional sellers

Seller places ad, chooses a budget and cost per click

We show the ad
if it is relevant and
budget is available

We show the ad
if it is **relevant** and
budget is available

Count
clicks & impressions

Update
budget & ranking

We chose
Storm & riak for
ranking calculation

Constraints

135M events/day
@ 3.2K/sec peak

accurate

real-time

scale horizontally

handle events out-of-order

ebay classifieds group

accurate

real-time

scale horizontally

handle events out-of-order

Storm

Real-time computation framework from Twitter

Stream based producer-consumer topologies

Nice properties for concurrent processing

Storm

You write:

a) code that handles a **single event**
in a **single threaded context**

b) **configuration** how the events are
produced and flow through the topology

Then **Storm sets up the queues** and
manages the Java VMs which run your code

Storm

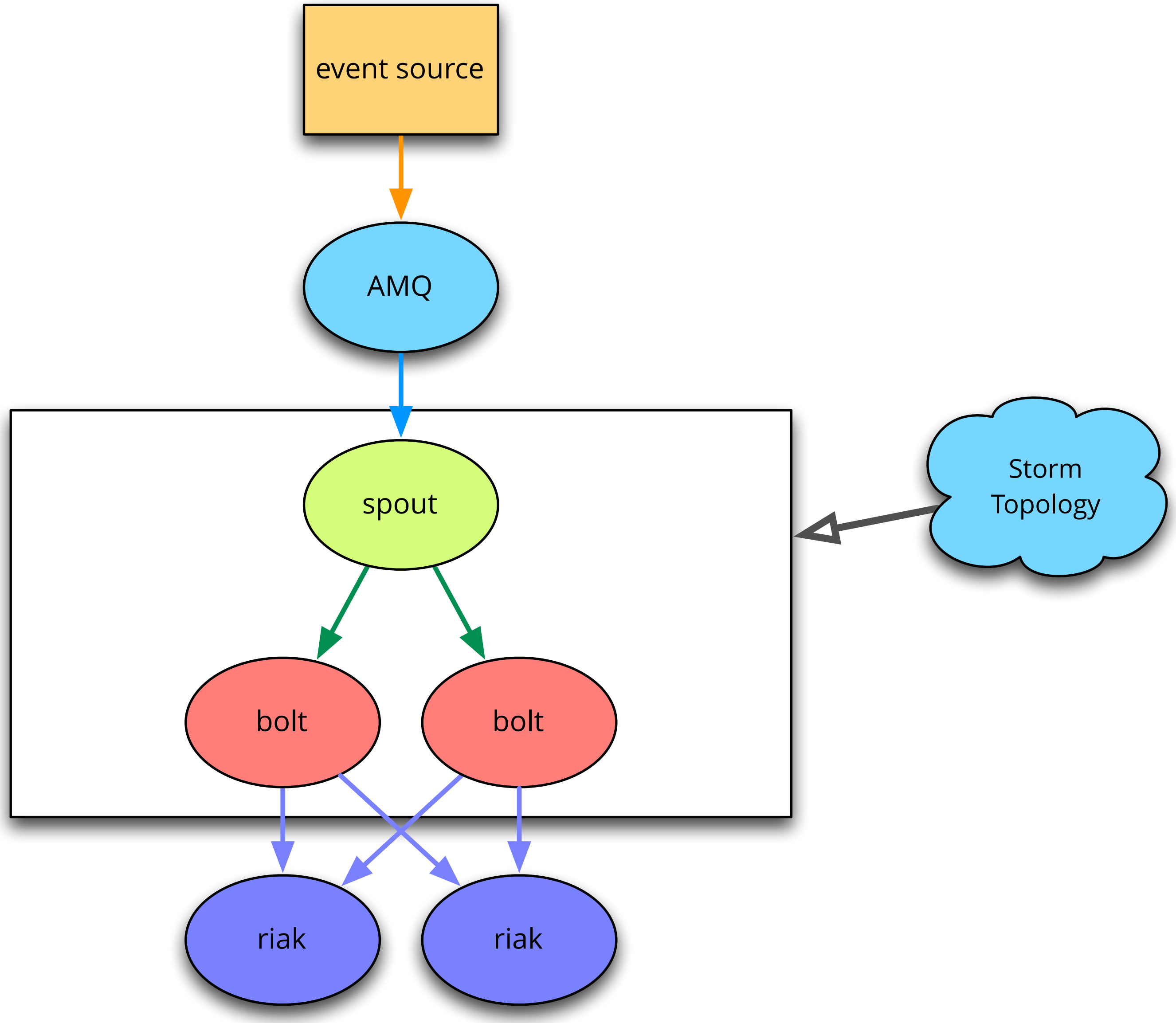
Spouts emit **tuples** (Producer)

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

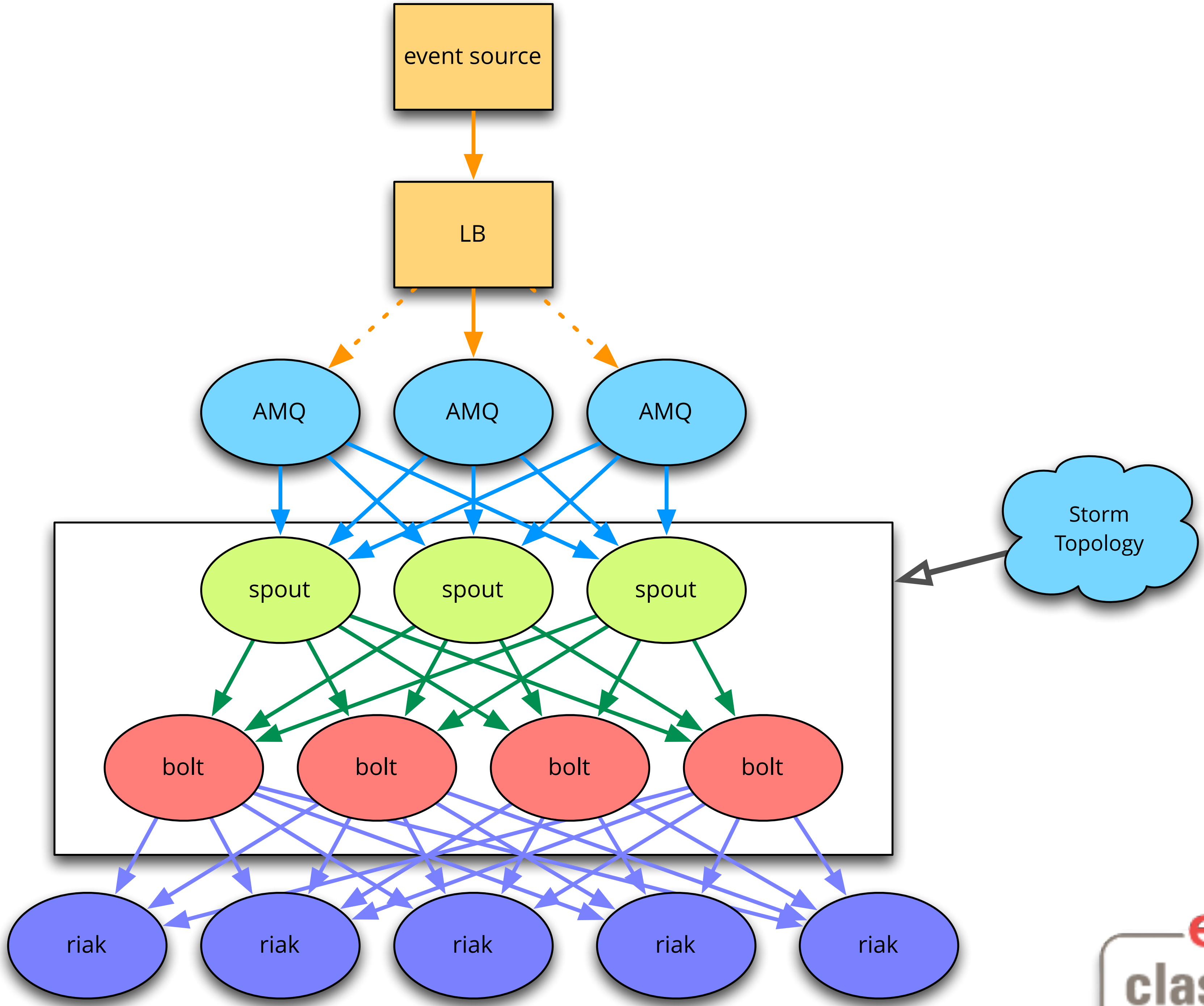
Storm worker = Java VM,
Each spout & bolt = 1 thread in a worker

Concurrency is configurable
and independent of your code

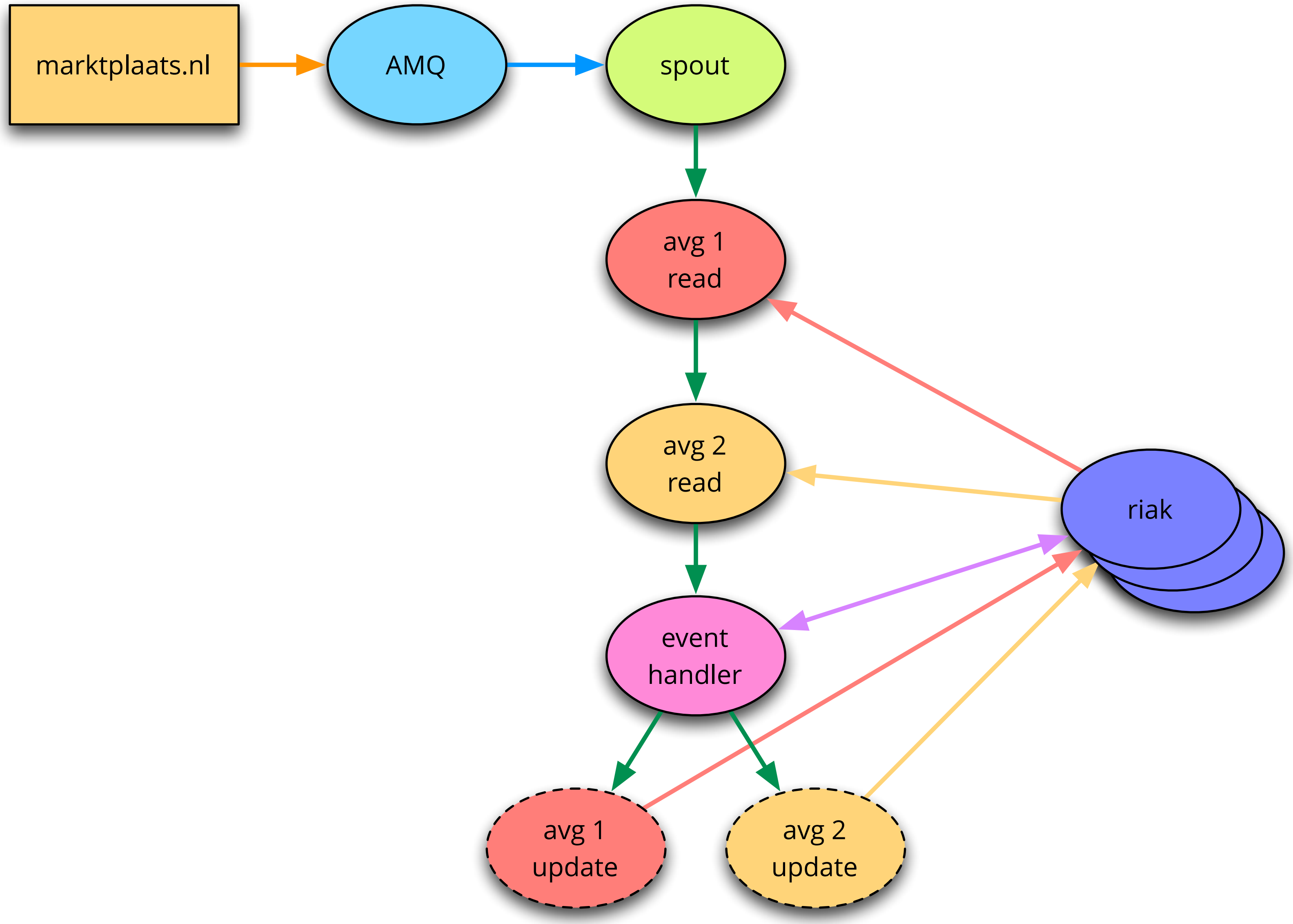
Storm
simple topology



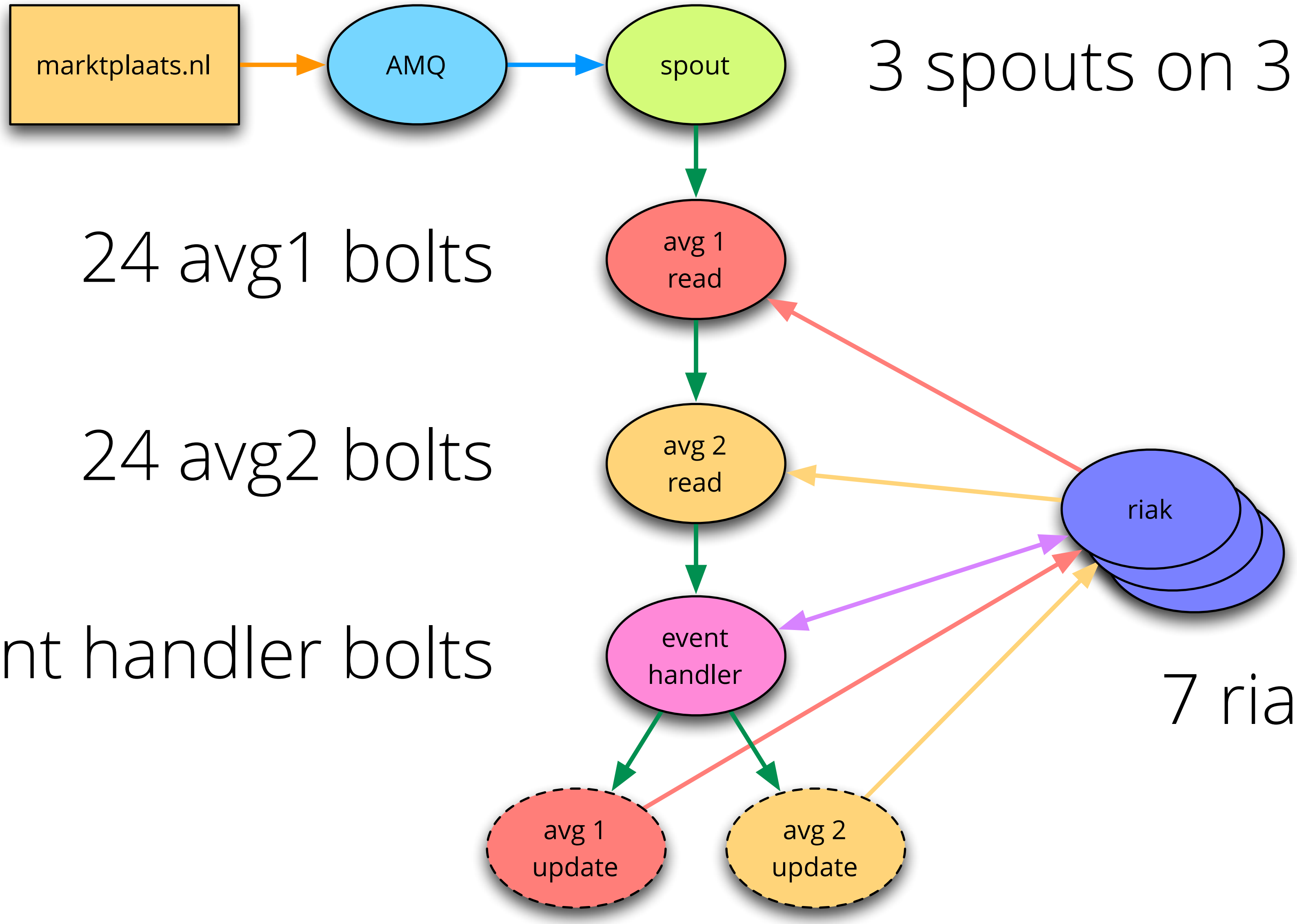
Storm
complex topology



Storm
our topology



Storm
our topology



marktplaats.nl

AMQ

spout

3 spouts on 3 servers

24 avg1 bolts

avg 1
read

24 avg2 bolts

avg 2
read

riak

96 event handler bolts

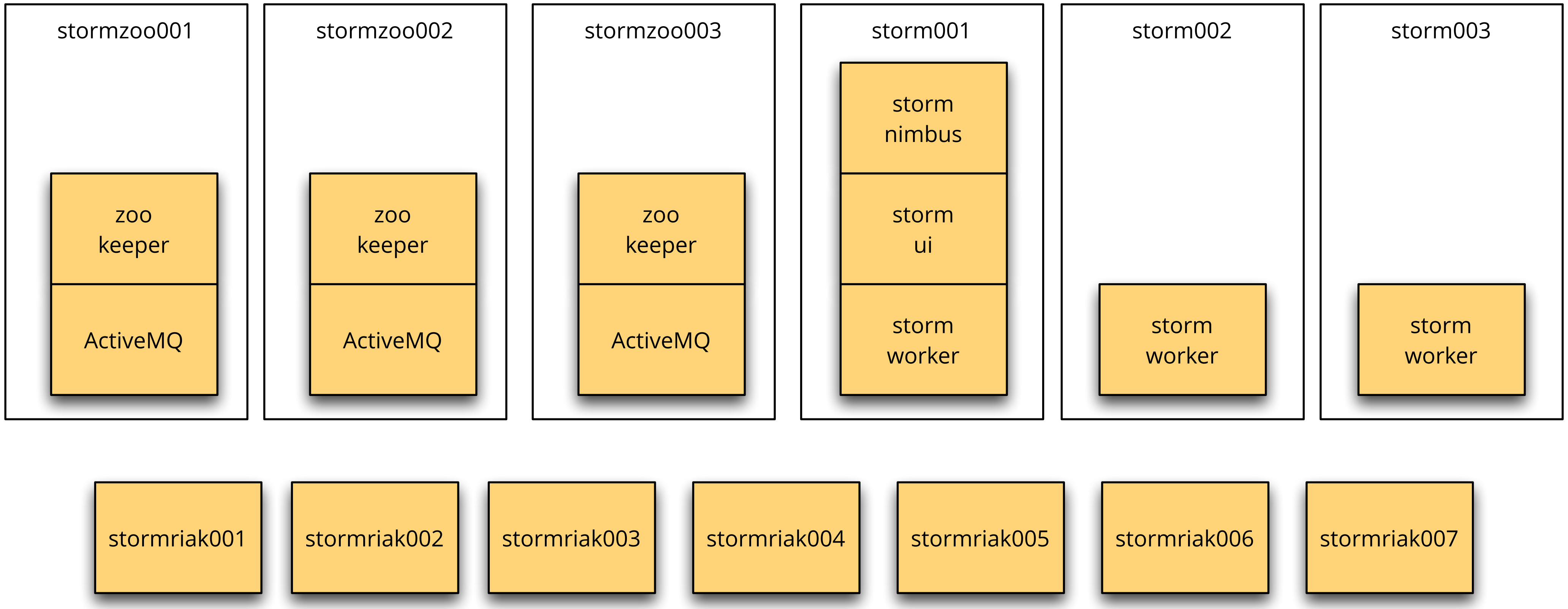
event
handler

7 riak nodes

avg 1
update

avg 2
update

Storm
Hardware Setup



Admarkt click-counter

1. Service writes JSON event to file and sends it to ActiveMQ. Use same format for logs and Storm.

2. Spouts read JSON events from ActiveMQ and emit them into the topology

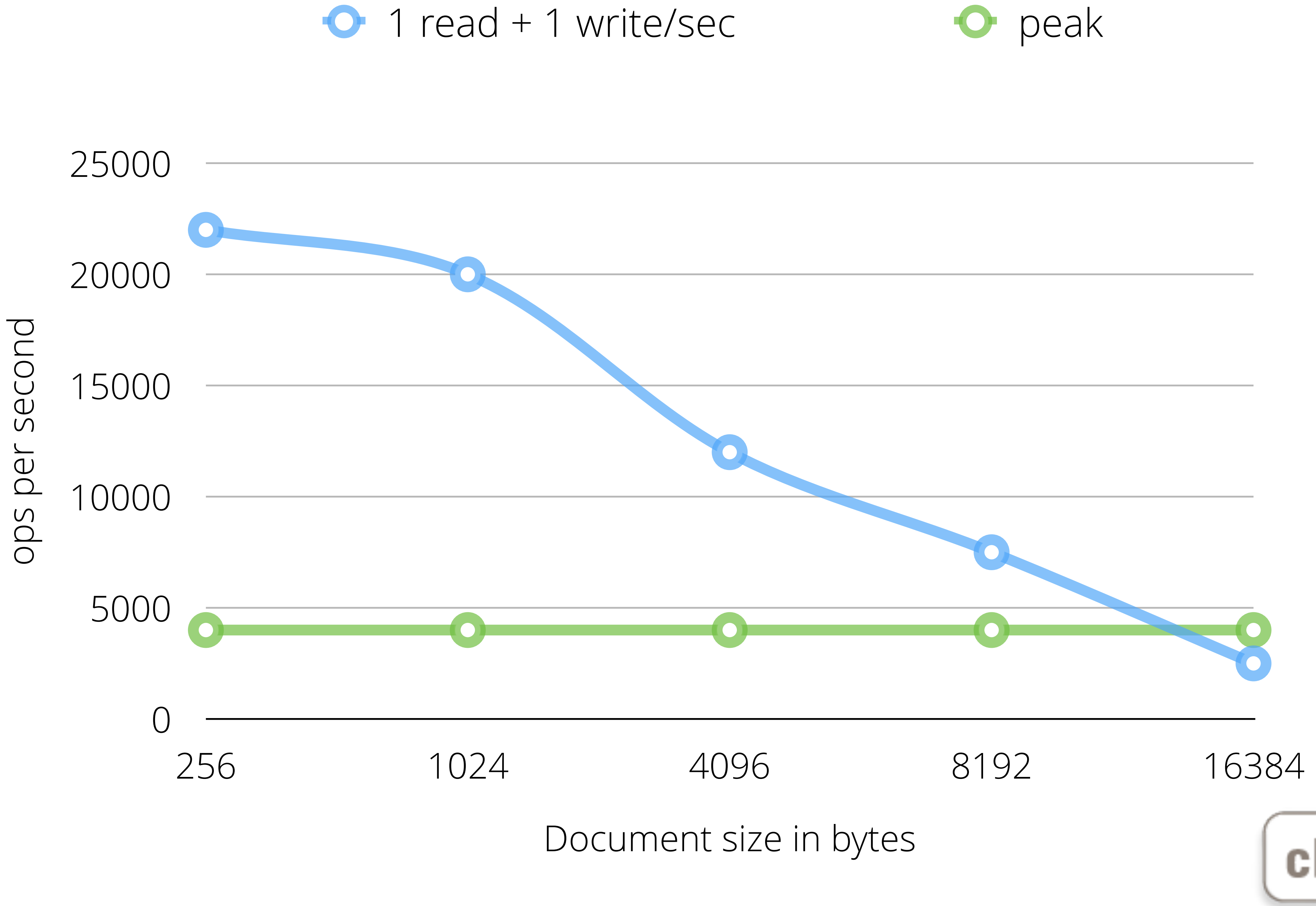
3. Bolts process events and update state in riak

If something goes wrong we replay events by putting the logs on the queue again riak for persistence

How fast can we write?

Riak Write Performance

riak 1.2.1, 5 node cluster



Conclusion: Document size is important

How can we be accurate?

How can we be accurate?

Handle each event exactly once

But events can arrive out-of-order…

But events can arrive out-of-order…

How can we know whether we have seen an event before?

Idea 1:
Comparing 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

Idea 2: Sequential 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

Idea 3:
Keep track of hashes

Event hash in current document: we have seen it already

Bootstrapping and replaying just works

Over-counting cannot happen

On failure just replay the logs

but ...

How many hashes do we have?

Keeping track of events

135M events per day -> 135M hashes

650K live ads -> 210 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?

Hash sizes

Remember that document size is important

sha1: 210*40 = 8.4KB

md5: 210*32 = 6.7KB

crc32: 210*8 = 1.7KB

Keeping documents small

Usually events are played forward in chronological order

Only during replay and failure we need older hashes

Keeping documents small

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 hh:00 move the older hashes into a separate document

Keep documents with older hashes for as long we want to be able to replay (1-2 weeks)

But with riak we don’t have TX …

Moving hashes from one doc to another without TX

1. Write archive doc with older hashes but keep them in the main document

2. Remove older events from the current document and then write it

Replaying events without TX

1. Load older hashes from riak and merge them with main document

2. Write archive doc with older hashes but keep them in the main document

3. Remove older events from the current document and then write i

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

Serialization

Maintainability is important, too

You can look at JSON (helpful)

Schema evolution via
Content-Type headers

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)

Persistence

Simple back pressure rule (deferred writes) saves us

Small doc -> write immediately

Larger doc -> wait up to 5 sec

Volatile docs receive lots of events during defer period. Saves writes

8 months in

Lessons learned

Riak

Cleaning up riak is hard since you ~~can’t~~ shouldn’t list buckets or keys. Easier with 2.0

Can’t query riak for “how many docs have value x > 5” without a program. Easier with 2.0

MapRed with gzipped JSON requires Erlang code. JS can’t handle it. Not in 2.0

Riak

Deferred writes only help so much. Maybe use constant write rate to make system more predictable.

Riak scales nicely with more nodes.

Storm

Mostly stable and fast (v0.8.2)

Must understand internal queues and their sizing. Otherwise, topology just stops

Need external tools for verifying that topology is working correctly

Hashes

Nice idea but creates unbounded number of documents. Disks fill up and cleaning up is hard.

Replay logic kills performance.

Replaying is too slow if we need to replay a full day or more.

rethink

We don’t want to know what we have seen

We want to know what we have not seen

This would solve some problems:

doc size constant

number of docs constant

riak cleanup not necessary

But how do we know what we haven’t seen if we don’t know what is coming?

Idea 2: Sequential 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

Idea 2:
Sequential Counters

event id < last id: we have seen it already

How do you build a distributed, reliable, sequential 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

Why just one counter?

Lets have multiple

Lets have multiple

e.g.

one per service instance

eventId =
counterId + counterValue

e.g.

hostA-20131030_152543:15

Create **unique counter id** at service start and **start counting from 0**

Increment atomically (AtomicLong) and send counter id + value to storm

Storm keeps track of counter value per counter id

Keep gap lists of missed events

Now we can predict what is coming

Questions?

ebay classifieds group

Thank you

frschroeder@ebay.com