An eventful journey to real-time joins with Storm and Kafka

Nicolas Dalsass - Yann Schwartz
6 DATA CENTERS, 4 CONTINENTS.
120 BILLION REQUESTS/DAY*. 

54 OPEN POSITIONS IN PARIS’ R&D
http://criteolabs.com/jobs

* EVERY DAY CRITEO IS CALLED MORE THAN 100 BILLION TIMES BY ADVERTISERS AND PUBLISHERS
1. Our Use Case

2. What we used to run

3. What we want to run

4. How we got there
Our use case

SELECT aid, b.data, d.data

FROM bids b, displays d

WHERE b.aid = d.aid
Our Use Case

Log 1
{id1, data}
{id2, data}
{id3, data}

Log 2
{id1, data2}
{id3, data2}

Log 3
{id1, data, data2}
{id3, data, data2}

+=+=+
Our Use Case

Log 1
{id1, data}
{id2, data}
{id3, data}
{id4, data}
{id5, data}
{id6, data}
{id7, data}
{id8, data}
10 Billion lines / day

Log 2
{id1, data2}
{id3, data2}

Log 3
{id1, data, data2}
{id3, data, data2}

2.5 Billion lines / day

10 Billion lines / day

+ = +

Copyright © 2012. Confidential
What we used to have

Hadoop
What we used to have

Hadoop
Data import

Bleeding edge

• Syslog
• gzip
• bash
• curl
The pain... it hurts
The pain…it hurts
What we want to do

Hadoop
Message Broker

Apache Kafka

Why?

• Seen as a message broker by clients
• Implemented as a logger
• Distributed without SPOF
• Good scaling performances
Stream Processing

Storm

• "wide" adoption
• Linearly scalable
• Simple enough development model
Orchestration

ZooKeeper

• Just works, you can forget it *
• Enables distributed consistency
The naïve plan

Kafka: Log 1
Spout 1
Join bolt
Spout 2
Kafka: Log 2
Storm
Kafka: Join Log
First wall: streams desynchronization

Kafka Topic 1 → Spout 1 → Spout 2 → Kafka Topic 2

t= Now  
t= Now
First wall: streams desynchronization

Kafka Topic 1 → Spout 1 → Spout 2 → Kafka Topic 2

\[ t = -2H \]

\[ t = -2H \]
First wall: streams desynchronization

Kafka Topic 1 → Spout 1 → Spout 2 → Kafka Topic 2

$t = -118 \text{ min}$

$t = -112 \text{ min}$
First wall: streams desynchronization

Kafka Topic 1 ➔ Spout 1 ➔ Spout 2 ➔ Kafka Topic 2

t = -118 min

t = -112 min

Let's add a feedback loop for synchronization and see what happens...
First wall: streams desynchronization

Let’s add a feedback loop for synchronization and see what happens…
First wall: streams desynchronization

Kafka Topic 1 → Spout 1 → Spout 2 → Kafka Topic 2

t = -118 min

t = -112 min

WTF?
The catch #1

Time’s arrow
Kafka: stock producer model

- Producer
  - Message 1
    - Message 100
  - Message 101
  - Message 200
  - Message 201
  - Message 300
  - Message 301
  - Message 400
Kafka loose consuming order

Spout read order:

Message 1, Message 101, Message 102, Message 201, Message 2, Message 3,…
First solution: regenerate timeline

Message 1 (t)
  
Message 100 (t+60s)
Message 301

Message 101 (t+61s)
  
Message 200 (t+120s)

Message 201 (t+121s)
  
Message 300 (t+180s)

Priority Queue Bolt

Join bolt message order: 1, 2, 3, …
Second solution: smart partitions

Log 1, partition 1
{id1, data}
{id11, data}
{id21, data}

Log 2, partition 1
{id1, data2}
{id11, data2}
{id21, data2}

Join bolt

Log 1, partition 2
{id2, data}
{id12, data}
{id22, data}

Log 2, partition 2
{id2, data2}
{id12, data2}
{id22, data2}

Join bolt
Second wall: the internal cache

Join bolt

Internal cache: HashMap

Java Heap Size needed: 4G

+ 

All Storm workers have the same huge heap size

= 

Limit on the number of workers per server
How we solve it

Join Bolt

External cache: Couchbase (server local)

Small heap size = Lot of workers

The performance overhead is negligible

We now have for free all monitoring tools associated to couchbase!
Feeling better

We no longer have logical problems

CPU usage on servers is still quite low

Let’s Profile

.....

We’re network bound!
Storm scheduling: 1\textsuperscript{st} problem
Storm scheduling: 1st problem

Server 1
- Spout 1
- Kafka Writer Bolt
- Join bolt

Server 2
- Spout 2
- Kafka Writer Bolt
- Join bolt

Join
bolt

Spout
1

Kafka
Writer
Bolt
Storm scheduling: 2nd problem

Server 1

Spout 1

Join bolt

Kafka Writer Bolt

Server 2

Spout 2

Join bolt

Kafka Writer Bolt
Storm scheduling : 2\textsuperscript{nd} problem
Storm scheduling: 2\textsuperscript{nd} problem

**Server 1**

- Spout 1
- Join bolt
- Kafka Writer Bolt

**Server 2**

- Spout 2
- Join bolt
- Kafka Writer Bolt
Let’s write our own scheduler

Storm allows plugging of custom schedulers

+ 

Storm allows adding any configuration to components, accessible by scheduler

= 

It’s easy to write a custom one!
Time for preprod

• Hardening Kafka
  • Replication
  • Acks
  • Batch sending
  • Compression

• Stock producers are not fit for our use case
  • YAQ (Yet Another Queue)
Lessons learned

Be aware of Kafka topic/partition model

Storm linear scalability: network bound

Forget Storm default scheduler: enforce data locality

In a distributed system, time is not linear
Next steps

Remove Rsyslog, let Kafka handle all logging

Let Kafka do the inter-DC mirroring
QUESTIONS ?
PSA

Did we mention *we’re hiring*?

We’re hiring lots of engineers in 2014.

http://criteolabs.com/jobs