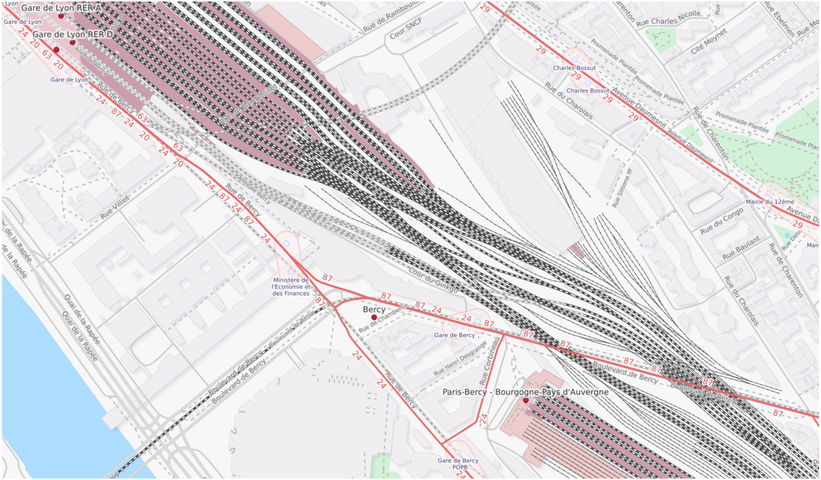
Locality-Aware Routing in Stateful Streaming Applications

Matthieu Caneill, Ahmed El Rheddane, Vincent Leroy, Noël de Palma
Univ. Grenoble Alpes, France

December 16, 2016
Middleware'16, Trento, Italy

Motivation



(C) OpenStreetMaps

Motivation

In datacenters, messages passing through a distributed system often **hop** on many machines, **saturating network links and top-level routers**.

Motivation

In datacenters, messages passing through a distributed system often **hop** on many machines, **saturating network links and top-level routers**.

How can we improve that?

Motivation

In datacenters, messages passing through a distributed system often **hop** on many machines, **saturating network links and top-level routers**.

How can we improve that?

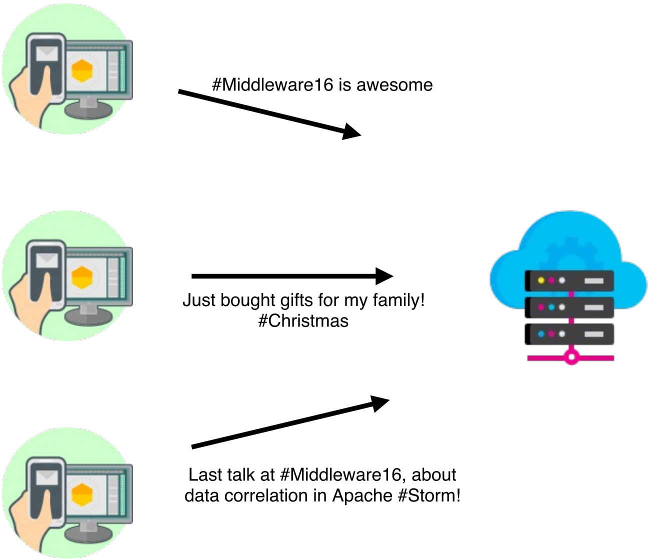
We focus on discovering **correlations** in messages, to **route** them according to their content, in order to **decrease** the machine-to-machine communication.

Table of contents

Table of contents

Distributed streaming engines

Introduction



Trending now

- Middleware16
- Christmas

Goals

- Real-time message handling
- Real-time metric calculations
- Workers scheduling and synchronization
- Fault-tolerance
- Many other features. . .

Distributed streaming engines

Simple topology

An application in Apache Storm is implemented as a topology.

Distributed streaming engines

Simple topology

An application in Apache Storm is implemented as a topology.

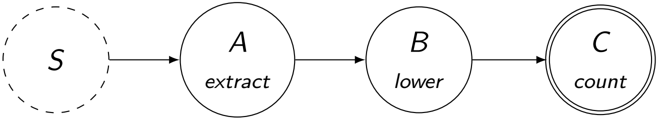


Figure: A simplified *trending hashtag* stream application.

S sends tweets, operator A extract hashtags, B converts them to lowercase, and C counts the frequency of each hashtag.

Distributed streaming engines

Simple topology

An application in Apache Storm is implemented as a topology.

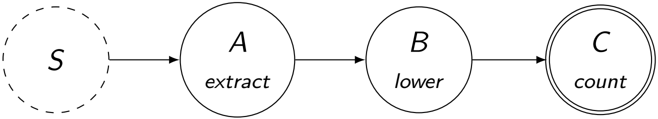


Figure: A simplified *trending hashtag* stream application.

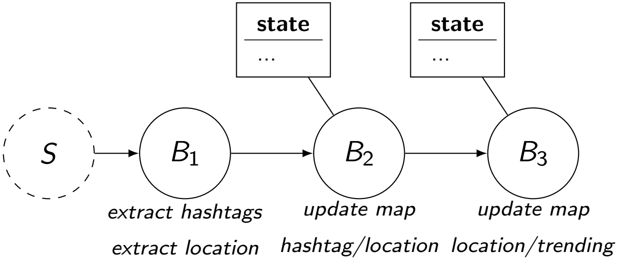
S sends tweets, operator A extract hashtags, B converts them to lowercase, and C counts the frequency of each hashtag.

A streaming application divides actions into different tasks. That makes distribution and parallelization to different nodes easy!

Stateful operators

States are associated to keys

For example, with tweets, we want to keep for each hashtag (key) the list of associated locations (values).



Stateful operators

Parallelization

When a task has many instances, it's harder to keep a consistent state. That's why same keys must be routed to the same instance.

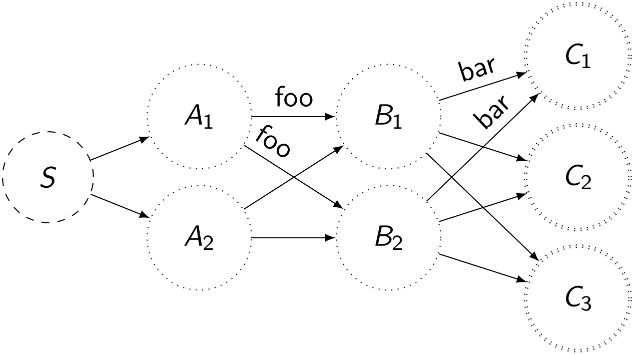


Figure: Tasks A and B are stateless, C is stateful.

Stateful operators

That's great

but only for small parallelism values.

Stateful operators

That's great

but only for small parallelism values.

In the real world

Most of the messages will be routed from one machine to the other.
On average, only $\frac{1}{parallelism}$ messages are treated locally.

Stateful operators

That's great

but only for small parallelism values.

In the real world

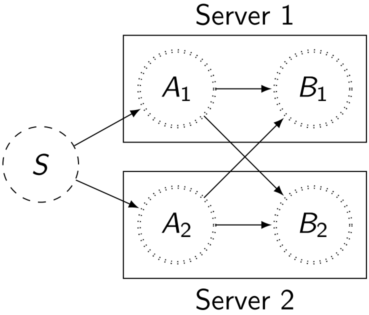
Most of the messages will be routed from one machine to the other.
On average, only $\frac{1}{parallelism}$ messages are treated locally.

$\rightarrow$ High use of network.

Table of contents

Situation

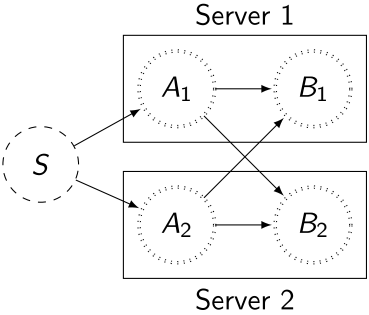
Let's have two stateful operators, each with two instances.



Locality

Situation

Let's have two stateful operators, each with two instances.



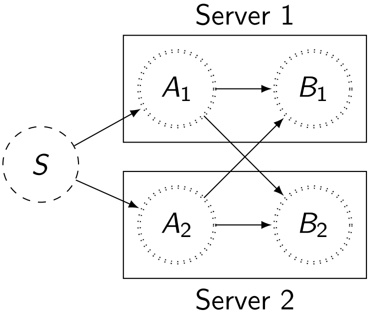
Goal

Minimize the traffic between the machines: $A1 \rightarrow B2$ and $A2 \rightarrow B1$.
By default, $locality = 1/parallelism$

Locality

Situation

Let's have two stateful operators, each with two instances.



Goal

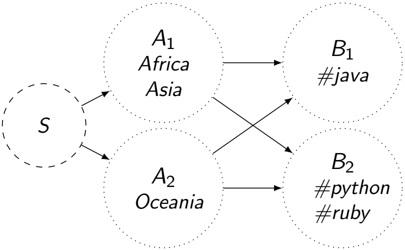
Minimize the traffic between the machines: $A1 \rightarrow B2$ and $A2 \rightarrow B1$. By default, $locality = 1/parallelism$

Constraint

Keep a good load balance between the machines.

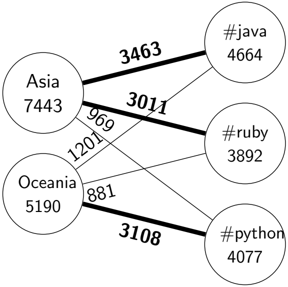
Keys correlation

We propose to dynamically instrument the keys couples and to represent it with a bipartite graph.



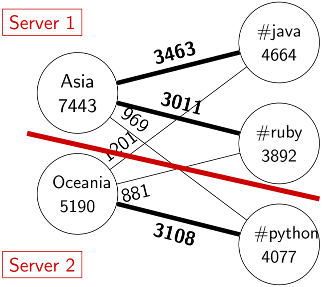
Keys correlation

We propose to dynamically instrument the keys couples and to represent it with a bipartite graph.



Keys correlation

We propose to dynamically instrument the keys couples and to represent it with a bipartite graph.

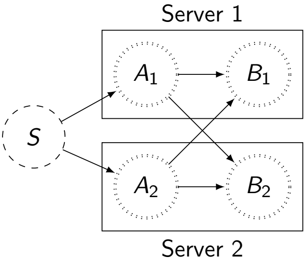


Routing tables

- S: $Asia \rightarrow A_1$
  $Oceania \rightarrow A_2$
- A_1: $\mathbf{\#java \rightarrow B_1}$
  $\mathbf{\#ruby \rightarrow B_1}$
  $\#python \rightarrow B_2$
- A_2: $\mathbf{\#python \rightarrow B_2}$
  $\#java \rightarrow B_1$
  $\#ruby \rightarrow B_1$

We then partition this graph to compute an optimized routing, favorizing local links.

In action



Message:
Posted from:

S

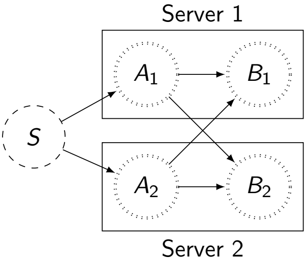
key	**route**

A

key	**route**

In action

Message: #python doesn't have braces
Posted from: Oceania



S

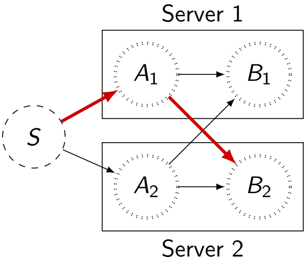
key	**route**

A

key	**route**

In action

Message: #python doesn't have braces
Posted from: Oceania



S

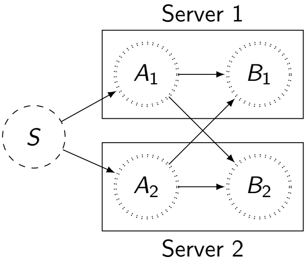
key	**route**
Oceania	A1

A

key	**route**
python	B2

In action

Message: #java is a verbose language
Posted from: Asia



S

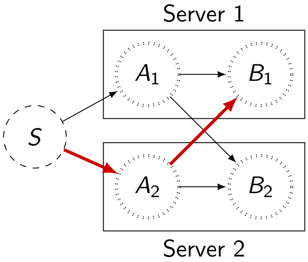
key	**route**
Oceania	A1

A

key	**route**
python	B2

In action

Message: #java is a verbose language
Posted from: Asia



S

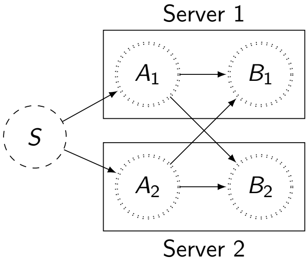
key	**route**
Oceania	A1
Asia	A2

A

key	**route**
python	B2
java	B1

In action

Message:
Posted from:



S

key	**route**
Oceania	A1
Asia	A2

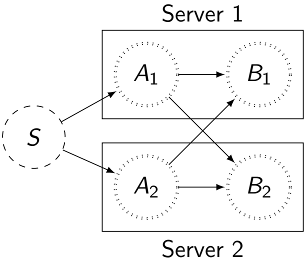
A

key	**route**
python	B2
java	B1

Reconfiguration is computed and applied

In action

Message:
Posted from:



S

key	route
Oceania	A1
Asia	A2

A

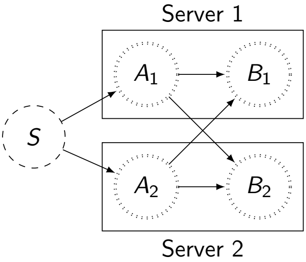
key	route
python	B1
java	B2

Reconfiguration is computed and applied

Correlation between **Oceania**/**python** and **Asia**/**java**

In action

Message: #python is pretty cool!
Posted from: Oceania



S

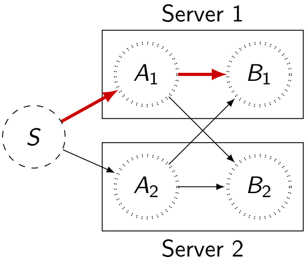
key	**route**
Oceania	A1
Asia	A2

A

key	**route**
python	B1
java	B2

In action

Message: #python is pretty cool!
Posted from: Oceania



S

key	**route**
Oceania	A1
Asia	A2

A

key	**route**
python	B1
java	B2

Trends evolve with time

Data is often skewed; the key frequency distribution often evolves with time.

Trends evolve with time

Data is often skewed; the key frequency distribution often evolves with time.

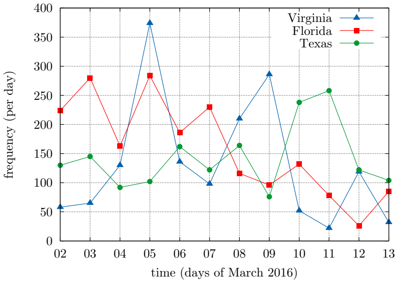


Figure: #nevertrump, in March 2016

Table of contents

Reconfiguration

When do we re-route?

- Key distribution evolves with time

Reconfiguration

When do we re-route?

- Key distribution evolves with time
- Routing tables optimized by examining old data lead to decreased locality.

Reconfiguration

When do we re-route?

- Key distribution evolves with time
- Routing tables optimized by examining old data lead to decreased locality.
- $\rightarrow$ We re-compute them every N minutes

Reconfiguration

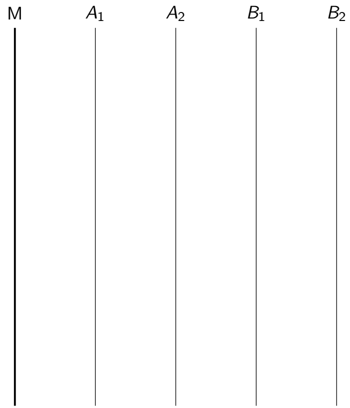
When do we re-route?

- Key distribution evolves with time
- Routing tables optimized by examining old data lead to decreased locality.
- $\rightarrow$ We re-compute them every N minutes
- But when we change the routing tables, we have to move the states to keep them consistent.

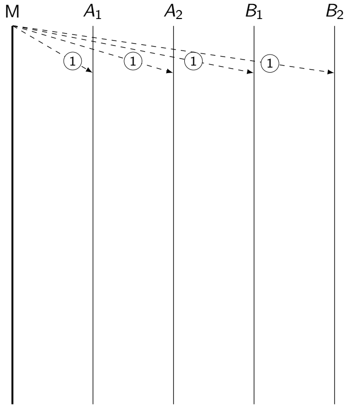
Reconfiguration protocol

We propose an online reconfiguration protocol, to update the routing tables in a live system while not losing any message and state.

Reconfiguration protocol

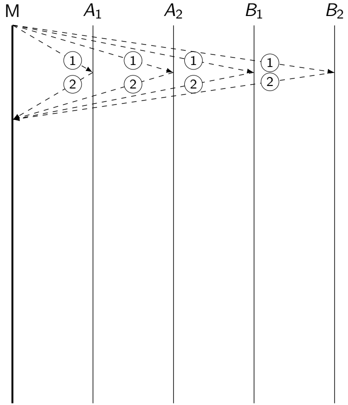


Reconfiguration protocol



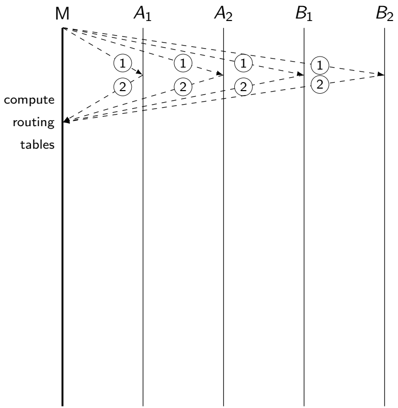
① Get statistics

Reconfiguration protocol



① Get statistics
② Send statistics

Reconfiguration protocol

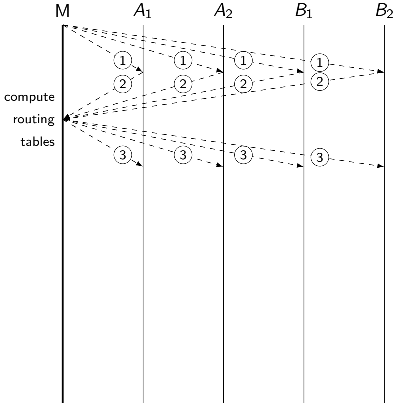


① Get statistics

② Send statistics

Partition graph, compute routing tables

Reconfiguration protocol



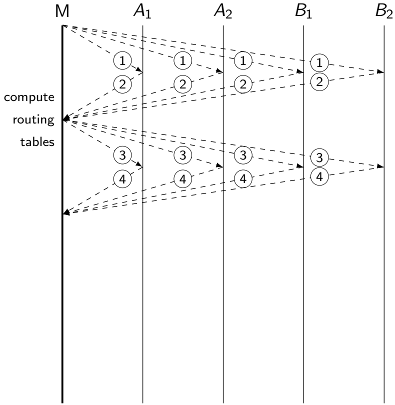
① Get statistics

② Send statistics

Partition graph, compute routing tables

③ Send reconfiguration

Reconfiguration protocol



① Get statistics

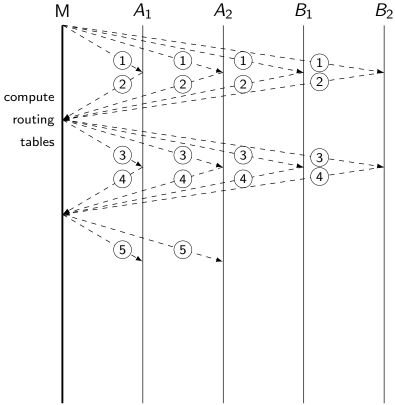
② Send statistics

Partition graph, compute routing tables

③ Send reconfiguration

④ Send ACK

Reconfiguration protocol



① Get statistics

② Send statistics

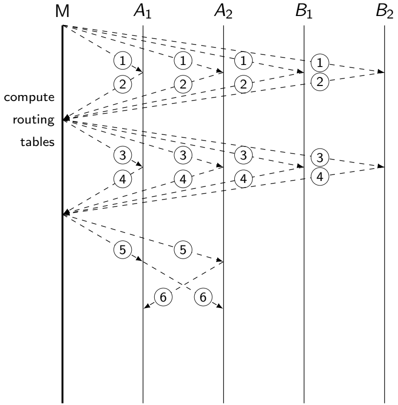
Partition graph, compute routing tables

③ Send reconfiguration

④ Send ACK

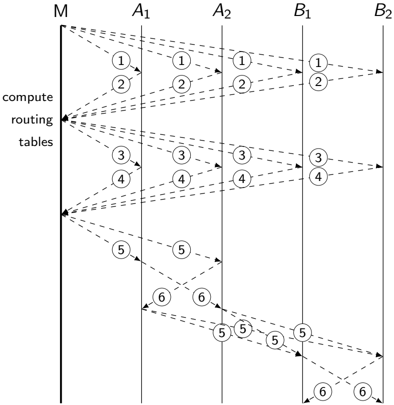
⑤ Propagate

Reconfiguration protocol



① Get statistics
② Send statistics
Partition graph, compute routing tables
③ Send reconfiguration
④ Send ACK
⑤ Propagate
⑥ Exchange key states

Reconfiguration protocol



① Get statistics

② Send statistics

Partition graph, compute routing tables

③ Send reconfiguration

④ Send ACK

⑤ Propagate

⑥ Exchange key states

Propagate to next operator

Table of contents

Datasets

- From Flickr and Twitter
- Fields: location (country or place), hashtag
- Size: 173M records (Flickr), 100M (Twitter)

Experiments

Datasets

- From Flickr and Twitter
- Fields: location (country or place), hashtag
- Size: 173M records (Flickr), 100M (Twitter)

Setup

- 8 HPE Proliant DL380 Gen9 servers (128 GB RAM, 20 cores).
- We simulate the flow by ingesting the records in the topology.
- The stateful workers compute basic aggregated statistics.
- Different parallelisms ranging from 2 to 6, different network speeds, and different message sizes.

Results
Insights

- It works well when network is the bottleneck

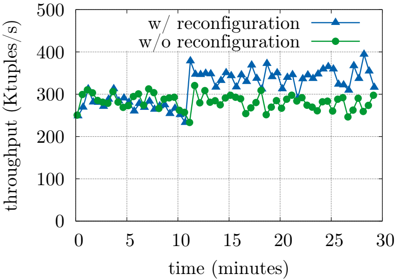
Results

Insights

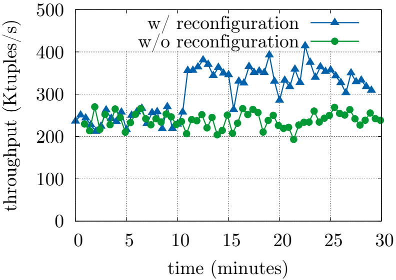
- It works well when network is the bottleneck
- Throughput difference highly depends on message size

Results

Flickr - Throughput with 10Gb/s networking, parallelism 6



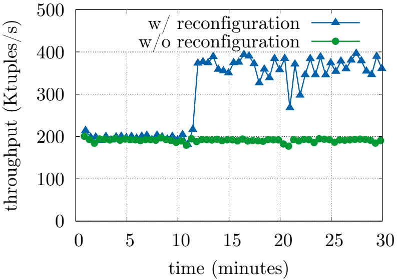
(a) message size=4kB



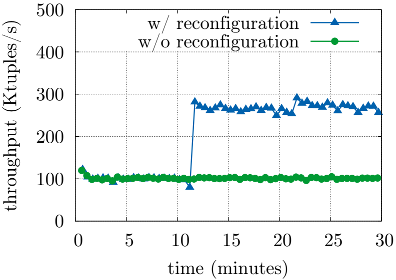
(b) message size=8kB

Results

Flickr - Throughput with 1Gb/s networking, parallelism 6



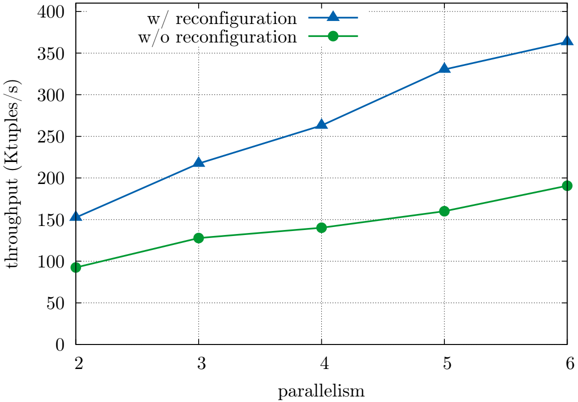
(a) message size=4kB



(b) message size=8kB

Results

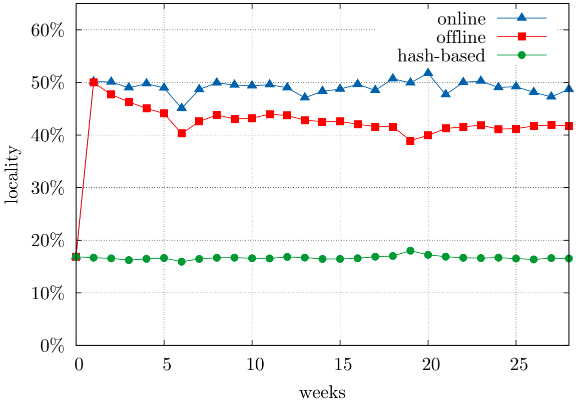
Flickr - Average throughput with 1Gb/s networking, 4kB message size



With reconfiguration, the average is measured after the first reconfiguration.

Results

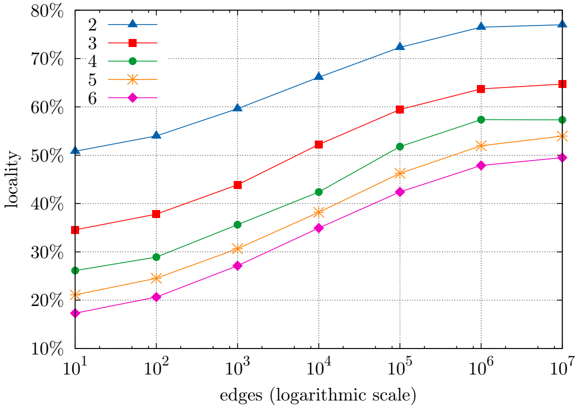
Flickr - Locality, with parallelism 6



Online: regular reconfiguration.
Offline: only one reconfiguration.
Hash-based: no reconfiguration.

Results

Flickr - Locality when changing the number of collected edges



Conclusion

- There are correlations between different fields of streaming messages
- We collect statistics in real-time about these correlations
- We use them to leverage the routing of the next messages, so that they are treated by co-located workers
- We use an orchestration algorithm to reconfigure the routing tables and not lose state

Table of contents

Future work

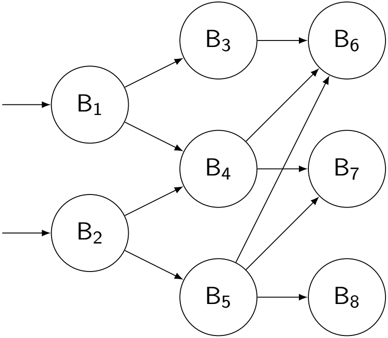
What's next?

- Replace binary locality/non-locality with distance
- Smarter way to determine when to reschedule
- Extend to more complex topologies

Future work

What's next?

- Replace binary locality/non-locality with distance
- Smarter way to determine when to reschedule
- Extend to more complex topologies



Thanks! Questions?

Matthieu Caneill
Univ. Grenoble Alpes, France
caneill@imag.fr