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
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

Background

Locality-aware routing

Reconfiguration protocol

Evaluation

Future work
Table of contents

Background

Locality-aware routing

Reconfiguration protocol

Evaluation

Future work
Distributed streaming engines

Introduction

#Middleware16 is awesome

Just bought gifts for my family! #Christmas

Last talk at #Middleware16, about data correlation in Apache #Storm!
Distributed streaming engines

Introduction

Trending now

- Middleware16
- Christmas
Distributed streaming engines

Introduction

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.
An application in Apache Storm is implemented as a topology.

**Figure:** A simplified *trending hashtag* stream application.

\[ S \rightarrow A \text{ extract} \rightarrow B \text{ lower} \rightarrow C \text{ count} \]

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

In the real world, most of the messages will be routed from one machine to the other. On average, only 1 parallelism message is treated locally. → High use of network.
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 $\frac{1}{\text{parallelism}}$ 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 $\frac{1}{\text{parallelism}}$ messages are treated locally.

→ High use of network.
Table of contents

Background

Locality-aware routing

Reconfiguration protocol

Evaluation

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

**Situation**

Goal

- Minimize the traffic between the machines: $A_1 \rightarrow B_2$ and $A_2 \rightarrow B_1$.

By default, locality $= \frac{1}{\text{parallelism}}$

Constraint

- Keep a good load balance between the machines.
Situation
Let’s have two stateful operators, each with two instances.

Goal
Minimize the traffic between the machines: $A_1 \rightarrow B_2$ and $A_2 \rightarrow B_1$. By default, $locality = 1/parallelism$.
Situation
Let’s have two stateful operators, each with two instances.

Goal
Minimize the traffic between the machines: $A_1 \rightarrow B_2$ and $A_2 \rightarrow B_1$. By default, $\textit{locality} = \frac{1}{\textit{parallelism}}$

Constraint
Keep a good load balance between the machines.
We propose to dynamically instrument the keys couples and to represent it with a bipartite graph.
We propose to dynamically instrument the keys couples and to represent it with a bipartite graph.
We propose to dynamically instrument the keys couples and to represent it with a bipartite graph.

Routing tables

- **S**: $\text{Asia} \rightarrow A_1$
  $\text{Oceania} \rightarrow A_2$

- **$A_1$**: $\text{#java} \rightarrow B_1$
  $\text{#ruby} \rightarrow B_1$
  $\text{#python} \rightarrow B_2$

- **$A_2$**: $\text{#python} \rightarrow B_2$
  $\text{#java} \rightarrow B_1$
  $\text{#ruby} \rightarrow B_1$

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

Message:
Posted from:

Server 1

$S$

$A_1$ $ightarrow$ $B_1$

$A_2$ $ightarrow$ $B_2$

Server 2

<table>
<thead>
<tr>
<th>key</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Message: #python doesn’t have braces
Posted from: Oceania
In action

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

<table>
<thead>
<tr>
<th>S</th>
<th>key</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Oceania</td>
<td>A1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>key</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>python</td>
<td>B2</td>
</tr>
</tbody>
</table>
Message: #java is a verbose language
Posted from: Asia
Message: #java is a verbose language
Posted from: Asia
In action

Message:
Posted from:

<table>
<thead>
<tr>
<th>key</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oceania</td>
<td>A1</td>
</tr>
<tr>
<td>Asia</td>
<td>A2</td>
</tr>
</tbody>
</table>

Reconfiguration is computed and applied
In action

Message:
Posted from:

<table>
<thead>
<tr>
<th>S</th>
<th>key</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oceania</td>
<td>A1</td>
<td></td>
</tr>
<tr>
<td>Asia</td>
<td>A2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>key</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td>python</td>
<td>B1</td>
<td></td>
</tr>
<tr>
<td>java</td>
<td>B2</td>
<td></td>
</tr>
</tbody>
</table>

Reconfiguration is computed and applied

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

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

<table>
<thead>
<tr>
<th>S</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oceania</td>
<td>A1</td>
</tr>
<tr>
<td>Asia</td>
<td>A2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td>python</td>
<td>B1</td>
</tr>
<tr>
<td>java</td>
<td>B2</td>
</tr>
</tbody>
</table>
Message: #python is pretty cool!
Posted from: Oceania
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
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.
When do we re-route?

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

<table>
<thead>
<tr>
<th>M</th>
<th>A₁</th>
<th>A₂</th>
<th>B₁</th>
<th>B₂</th>
</tr>
</thead>
</table>

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

Propagate to next operator
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

\( \begin{align*}
\text{compute routing tables} \\
\text{Partition graph, compute routing tables} \\
\text{Send reconfiguration}
\end{align*} \)
Reconfiguration protocol

1. Get statistics
2. Send statistics
   *Partition graph, compute routing tables*
3. Send reconfiguration
4. Send ACK
Reconfiguration protocol

Partition graph, compute routing tables

1. Get statistics
2. Send statistics
3. Send reconfiguration
4. Send ACK
5. Propagate
Reconfiguration protocol

Partition graph, compute routing tables

1. Get statistics
2. Send statistics
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*
Experiments

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
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.
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

Background

Locality-aware routing

Reconfiguration protocol

Evaluation

Future work
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