ONLINE PERFORMANCE ANALYSIS OF DISTRIBUTED DATAFLOW SYSTEMS

Vasia Kalavri
kalavriv@inf.ethz.ch
ABOUT ME

▸ Postdoc at ETH Zürich
  ▸ Systems Group: https://www.systems.ethz.ch/

▸ PMC member of Apache Flink

▸ Research interests
  ▸ Large-scale graph processing
  ▸ Streaming dataflow engines

▸ Current project
  ▸ Predictive datacenter analytics and management

@vkalavri
STRYMON: ONLINE DATACENTER ANALYTICS AND MANAGEMENT

Datacenter

policy enforcement, what-if scenarios, ...

Strymon

event streams

traces, configuration, topology updates, ...

Datacenter model

queries, complex analytics, simulations, ...

strymon.systems.ethz.ch
STRYMON IS BUILT ON TIMELY

- A steaming framework for data-parallel computations
  - Arbitrary cyclic dataflows
  - Logical timestamps (epochs)
  - Asynchronous execution
  - Low latency

https://github.com/frankmcsherry/timely-dataflow

Strymon

- What-if scenarios
- Real-time datacenter analytics
- Incremental network routing
- Online critical path analysis
- Explaining simulations
Strymon

What-if scenarios
Real-time datacenter analytics
Incremental network routing
Online critical path analysis
Explaining simulations

SnailTrail: Strymon’s component for online performance analysis of distributed dataflows

Apache Flink
Spark
Storm
TensorFlow
DATAFLOW PROGRAMMING

sources

operators

data exchange

sinks
UNDERSTANDING THE PERFORMANCE OF DISTRIBUTED DATAFLOWS

Hard to troubleshoot

- long-running, dynamic workloads
- many tasks, activities, operators, dependencies
- bottleneck causes are usually not isolated but span multiple processes
PERFORMANCE METRICS

![Diagram showing the process flow and metrics for data processing tasks. The diagram includes nodes labeled Source: Custom Source, Flat Map, Keyed Aggregation -> Sink: Unnamed, with parallelism levels and task durations.]
PERFORMANCE METRICS

Dataflow graph

Subtasks | TaskManagers | Metrics | Accumulators | Checkpoints | Back Pressure
--- | --- | --- | --- | --- | ---
| Bytes received | Records received | Bytes sent | Records sent | Parallelism | Tasks | Status
Source: Custom Source | 0 B | 0 | 2.56 GB | 12,942,429 | 4 | Running
Flat Map | 2.54 GB | 12,844,494 | 4.59 GB | 254,056,192 | 2 | Running
Keyed Aggregation -> Sink: Unnamed | 4.59 GB | 254,646,032 | 0 B | 0 | 4 | Running
**PERFORMANCE METRICS**

*Dataflow graph*

```
Source: Custom Source  Parallelism: 4
   REBALANCE

Flat Map  Parallelism: 2
   HASH

Keyed Aggregation -> Sink: Unnamed  Parallelism: 4
```

**Duration**

<table>
<thead>
<tr>
<th>Start Time</th>
<th>End Time</th>
<th>Duration</th>
<th>Name</th>
<th>Bytes received</th>
<th>Records received</th>
<th>Bytes sent</th>
<th>Records sent</th>
<th>Parallelism</th>
<th>Tasks</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-10-19, 17:45:43</td>
<td>2017-10-19, 17:47:42</td>
<td>1m 58s</td>
<td>Source: Custom Source</td>
<td>0 B</td>
<td>0</td>
<td>2.56 GB</td>
<td>12,942,429</td>
<td>4</td>
<td></td>
<td>RUNNING</td>
</tr>
<tr>
<td>2017-10-19, 17:45:43</td>
<td>2017-10-19, 17:47:42</td>
<td>1m 58s</td>
<td>Flat Map</td>
<td>2.54 GB</td>
<td>12,844,494</td>
<td>4.59 GB</td>
<td>254,056,192</td>
<td>2</td>
<td></td>
<td>RUNNING</td>
</tr>
<tr>
<td>2017-10-19, 17:45:43</td>
<td>2017-10-19, 17:47:42</td>
<td>1m 58s</td>
<td>Keyed Aggregation -&gt; Sink: Unnamed</td>
<td>4.59 GB</td>
<td>254,646,032</td>
<td>0 B</td>
<td>0</td>
<td>4</td>
<td></td>
<td>RUNNING</td>
</tr>
</tbody>
</table>
PERFORMANCE METRICS

Dataflow graph

Aggregate data exchange

Duration
PERFORMANCE METRICS

Dataflow graph

Custom aggregate metrics

Aggregate data exchange

Duration
A PARALLEL EXECUTION

W1
- processing
- waiting

message

W2

W3
- serialization
- processing
- deserialization
A PARALLEL EXECUTION

W1

W2

W3

processing

waiting

message

serialization

processing

deserialization
Processing is the most *time-consuming* activity
A PARALLEL EXECUTION

What if we optimize it?

processing

waiting

serialization

processing
deserialization
A PARALLEL EXECUTION

W1

W2

W3

serialization

waiting

deserialization
A PARALLEL EXECUTION

W1

W2

W3

serialization

waiting

deserialization
A PARALLEL EXECUTION

W1

W2

W3

waiting

serialization

deserialization
A PARALLEL EXECUTION

No performance benefit for the parallel execution!
CONVENTIONAL PROFILING CAN BE MISLEADING
CRITICAL PATH ANALYSIS
THE PROGRAM ACTIVITY GRAPH (PAG)
Nodes are *timestamped* events: start or end of a worker activity
THE PROGRAM ACTIVITY GRAPH (PAG)

Nodes are *timestamped* events: start or end of a worker activity

\[
\begin{align*}
u & = \{ \\
& \quad \text{timestamp: } k+1, \\
& \quad \text{worker: } 2 \\
\} 
\end{align*}
\]
THE PROGRAM ACTIVITY GRAPH (PAG)

Edges represent activities annotated with a type and duration
THE PROGRAM ACTIVITY GRAPH (PAG)

Edges represent activities annotated with a type and duration.

\[(u, v) = \{ 
\text{type}: \text{serialization} \\
\text{duration}: 1 
\}\]
The Program Activity Graph captures computational dependencies among parallel workers.

▸ Which activities delay the overall execution?

▸ i.e. which activities lie on the critical path of execution?
The **longest path** in the execution history
(not considering waiting activities)
The **longest path** in the execution history
(not considering waiting activities)
CRITICAL PATH

The **longest path** in the execution history (not considering waiting activities)
The **longest path** in the execution history
(not considering waiting activities)
CRITICAL PATH

The **longest path** in the execution history (not considering waiting activities)
The **longest path** in the execution history
(not considering waiting activities)
The **longest path** in the execution history
(not considering waiting activities)
CRITICAL PATH
CRITICAL PATH
Reduced execution time
POST-MORTEM CRITICAL PATH ANALYSIS IS EASY

1. Collect traces during execution

![Diagram showing job start, profiler, and job end]
POST-MORTEM CRITICAL PATH ANALYSIS IS EASY

1. Collect traces during execution

2. Analyze traces offline

- job start
- job end
- profiler
- database
- analyzer
- performance summaries
How to compute the critical path for continuously running, distributed streaming applications, with potentially unbounded input?
How to compute the critical path for continuously running, distributed streaming applications, with potentially unbounded input?

- There might be no “job end”
- The PAG and critical path are continuously evolving
- Stale profiling information is not useful
ONLINE CRITICAL PATH ANALYSIS
ONLINE ANALYSIS OF TRACE SNAPSHOTS
ONLINE ANALYSIS OF TRACE SNAPSHOTS

input stream
periodic snapshot
output stream
ONLINE ANALYSIS OF TRACE SNAPSHOTS

input stream

periodic snapshot

trace snapshot stream

output stream

performance summaries stream

analyzer

stream
PROGRAM ACTIVITY GRAPH SNAPSHOT
All paths have the same length: $t_e - t_s$
All paths have the same length: $t_e - t_s$
All paths have the same length: $t_e - t_s$
All paths have the same length: $t_e - t_s$

Choosing a random path might miss critical activities
All paths have the same length: $t_e - t_s$

Choosing a random path might miss critical activities
- All paths have the same length: $t_e - t_s$
- Choosing a random path might miss critical activities
- All paths have the same length: \( t_e - t_s \)
- Choosing a random path might miss critical activities
- Enumerating all paths is impractical
All paths are potentially part of the evolving critical path

How to rank activities with regard to criticality?
All paths are potentially part of the evolving critical path

How to rank activities with regard to criticality?

**Intuition:** the more paths an activity appears on the more probable it is that this activity is critical
CRITICAL PARTICIPATION (CP METRIC)

An estimation of the activity’s participation in the critical path

\[ CP_a = \frac{C(a) \cdot \alpha_w}{N(t_e - t_s)} \]

- **centrality**: the number of paths this activity appears on
- **activity duration**: edge weight
- **total number of paths** in the snapshot
**CRITICAL PARTICIPATION (CP METRIC)**

An estimation of the activity’s participation in the critical path

**centrality:** the number of paths this activity appears on

**activity duration:** edge weight

\[ CP_a = \frac{C(a) \cdot a_w}{N(t_e - t_s)} \]

- total number of paths in the snapshot
- Can be computed without path enumeration!
ONLINE PERFORMANCE ANALYSIS WITH SNAILTRAIL
SNAILTRAIL IN ACTION

Reference application

Profiling

Trace generation

SnailTrail

Trace ingestion

PAG construction

CP computation and activity ranking

CP-based performance summaries

Apache Flink, Apache Spark, TensorFlow, Heron, Timely Dataflow, ...
EXAMPLE: TASK SCHEDULING IN APACHE SPARK

SCHEDULING BOTTLENECK IN APACHE SPARK

Apache Spark: Yahoo! Streaming Benchmark, 16 workers, 8s snapshots
Activity Summary

which activity type is a bottleneck?
Apache Flink: Dhalion WordCount Benchmark, 4 workers, 1s snapshots
**Activity Summary**

![Graph](image)

- **Optimize serialization!**

- **Apache Flink: Dhalion WordCount Benchmark, 4 workers, 1s snapshots**
SNAILTRAIL CP-BASED SUMMARIES

- **Activity Summary**
  - *which activity type is a bottleneck?*

- **Straggler Summary**
  - *which worker is a bottleneck?*
Straggler Summary

Apache Flink: Dhalion WordCount Benchmark, 4 workers, 1s snapshots
Apache Flink: Dhalion WordCount Benchmark, 4 workers, 1s snapshots
SNAILTRAIL CP-BASED SUMMARIES

- Activity Summary
  - which activity type is a bottleneck?

- Straggler Summary
  - which worker is a bottleneck?

- Operator Summary
  - which operator is a bottleneck?
Operator Summary

Apache Flink: Dhalion WordCount Benchmark, 10 workers, 1s snapshots
Increase flatmap’s parallelism!

Apache Flink: Dhalion WordCount Benchmark, 10 workers, 1s snapshots
SNAILTRAIL CP-BASED SUMMARIES

- **Activity Summary**
  - *which activity type is a bottleneck?*

- **Straggler Summary**
  - *which worker is a bottleneck?*

- **Operator Summary**
  - *which operator is a bottleneck?*

- **Communication Summary**
  - *which communication channels are bottlenecks?*
Communication Summary

Collocate worker 5 with 2, 9, 10, 11

Communication Criticality

- Purple
- Blue
- Teal
- Green
- Yellow
SNAILTRAIL PERFORMANCE

- Low instrumentation overhead
  - < 10% for all reference systems
- High throughput
  - 1.2 million events per second
- Always online
  - 1s of traces in 6ms, 256s of traces in < 25s

SnailTrail on Intel Xeon E5-4640, 2.40GHz, 32 cores, 512GB RAM
Traces from Apache Flink Sessionization, 48 workers, 1s-256s snapshots
**RECAP**

*Strymon*: online datacenter analytics and management
**RECAP**

**Strymon**: online datacenter analytics and management

*Conventional* profiling is misleading
**RECAP**

**Strymon**: online datacenter analytics and management

**Conventional** profiling is misleading

**CP-metric**: online critical path analysis
RECAP

Strymon: online datacenter analytics and management

Conventional profiling is misleading

CP-metric: online critical path analysis

SnailTrail: online CP-based summaries
THE STRYMON TEAM & FRIENDS

Prof. Timothy Roscoe
John Liagouris
Moritz Hoffmann
Andrea Lattuada
Desislava Dimitrova
Zaheer Chothia
Sebastian Wicki
Frank McSherry
Vasiliki Kalavri
THE STRYMON TEAM & FRIENDS

Prof. Timothy Roscoe

John Liagouris

Moritz Hoffmann

Andrea Lattuada

Frank McSherry

Desislava Dimitrova

Zaheer Chothia

Sebastian Wicki

Vasiliki Kalavri

?
THE STRYMON TEAM & FRIENDS

Prof. Timothy Roscoe
John Liagouris
Moritz Hoffmann
Andrea Lattuada

Vasiliki Kalavri

IT COULD BE YOU!

strymon.systems.ethz.ch
www.systems.ethz.ch/positions

Sebastian Wicki

?
ONLINE PERFORMANCE ANALYSIS OF DISTRIBUTED DATAFLOW SYSTEMS

Vasia Kalavri
kalavriv@inf.ethz.ch

Support: amadeus  FNS SNP  Google