SnailTrail
Generalizing Critical Paths for Online Analysis of Distributed Dataflows

Moritz Hoffmann, Andrea Lattuada, John Liagouris, Vasiliki Kalavri, Desislava Dimitrova, Sebastian Wicki, Zaheer Chothia, and Timothy Roscoe

Supported by

amADEUS

Google
SnailTrail: Diagnosing latency issues in dataflows
“Where is the latency bottleneck in my computation?”

Reference Application

Profiling data

trace streams

Trace ingestion
Program activity graph construction
Critical participation computation and activity ranking

Performance summaries

SnailTrail

Flink, Spark, TensorFlow, Heron, Timely dataflow, ...

2
SnailTrail works online with minimal instrumentation

- SnailTrail
- Coz: Causal Profiling
- Stitch
- Pivot Tracing
- Vscoope

Online

Offline

Less Instrumentation complexity
- More

OSDI'16
SOSP'15
Middleware '12
Example 1: Metrics in Flink’s dashboard

<table>
<thead>
<tr>
<th>Duration</th>
<th>Bytes received</th>
<th>Records received</th>
<th>Bytes sent</th>
<th>Records sent</th>
<th>Attempt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1m 28s</td>
<td>1.26 GB</td>
<td>2,871,683</td>
<td>3.20 MB</td>
<td>91,159</td>
<td>1</td>
</tr>
<tr>
<td>1m 28s</td>
<td>1.22 GB</td>
<td>2,764,835</td>
<td>3.52 MB</td>
<td>99,696</td>
<td>1</td>
</tr>
<tr>
<td>1m 28s</td>
<td>1.22 GB</td>
<td>2,777,058</td>
<td>3.15 MB</td>
<td>88,610</td>
<td>1</td>
</tr>
</tbody>
</table>
Example 2: Task Scheduling in Spark

Driver

W1

W2

W3

Conventional profiling

SnailTrail, critical participation

% time

Window

Processing
Scheduling
The real-world is more complex

Many tasks, activities, operators, dependencies

Long-running, dynamic workloads

Bottlenecks not isolated

Credits: Frank McSherry, “Tracking progress in timely dataflow”
Conventional profiling can indicate wrong bottleneck

W1

W2

W3

processing

waiting

serialization

processing

deserialization
Conventional profiling can indicate wrong bottleneck
A quick review of critical path analysis
The program activity graph

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

\[ u = \{ \text{timestamp: } t, \text{ worker: } 2 \} \]
The program activity graph

(u, v) = {
    type: serialization,
    operator: map,
}

Edges represent typed activities
The program activity graph

Activities express dependencies

Waiting activities are never on a critical path

All workers terminate
The program activity graph

Which activities delay the overall execution?
Classical critical path analysis
What is the equivalent of a critical path for continuously running, distributed **streaming** applications, with potentially **unbounded** input?

There might be no “job end”

The program activity graph and critical paths change continuously

Profiling information can quickly become stale
Online critical path analysis
SnailTrail: Online analysis of trace windows

Input stream

Output stream

Trace stream

SnailTrail

Periodic windows

Windowed performance summaries
Program activity graph window

All critical paths have equal length
Cannot enumerate all critical paths

Impractical!
Spark trace: $10^{21}$ critical paths in 10 second window
Sampling critical paths misses critical activities
We rank activities across all critical paths to capture their relative importance.

Intuition: The more critical paths go through an activity, the more critical it might be.
Counting over enumerating
The Critical Participation metric

Fraction of an edge’s time contribution across all critical paths

\[ CP(e) = \frac{\#p(e)}{N} \cdot \frac{w(e)}{t_e - t_s} \]

Critical paths through edge

Edge weight

Total number of critical paths

Can be computed without critical path enumeration!
SnailTrail in action

Reference Application

Profiling data

trace streams

SnailTrail

Trace ingestion

Program activity graph construction

Critical participation computation and activity ranking

Performance summaries

Flink, Spark, TensorFlow, Heron, Timely dataflow, ...

✓

✓
Interpreting critical participation-based summaries

Stream of tuples:
(Activity type, Operator, Worker, ..., Critical participation)

Examples:
Activity type bottleneck analysis
Operator bottleneck analysis

(More in the paper!)
Activity type bottleneck analysis (Spark)

Apache Spark: Yahoo! Streaming Benchmark, 16 workers, 8s windows

Conventional profiling vs SnailTrail, critical participation
Operator bottleneck analysis (Flink)

Apache Flink: Dhalion WordCount Benchmark, 10 workers, 1s windows

Read sentences → Flatmap: Split words → Count words

![Graph showing conventional profiling vs. SnailTrail, critical participation with emphasis on increasing flatmap's parallelism.](chart.png)
SnailTrail performance

**Low instrumentation overhead**

Spark, TensorFlow
No observed overhead
Flink, Timely
~10% overhead compared to logging disabled

**High throughput**

1.2 million events/s
8 workers

**Always online**

1s of traces in 6ms (100x)
256s of traces in < 25s (10x)

SnailTrail on Intel Xeon E5-4640, 2.40GHz, 32 cores, 512GiB RAM
Trace: Apache Flink Sessionization, 48 workers, 1s-256s windows
Summary

Conventional profiling is misleading

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

**SnailTrail**: online CP-based summaries

[Link to Strymon System GitHub repository](https://github.com/strymon-system/snailtrail)