JetStream: Cluster-scale Parallelization of Information Flow Queries

Andrew Quinn, David Devecsey, Peter Chen and Jason Flinn
Dynamic Information Flow Tracking

• DIFT instruments execution to track causality
• Also known as “Taint-Tracking”
DIFT for Debugging
DIFT for Debugging
DIFT for Debugging

Inputs

Server

Outputs
DIFT for Debugging

Inputs

Server

Outputs
DIFT for Debugging

Inputs

Server

Outputs
DIFT – limitations

Arnold ‘14
Overheads ~100x

X-Ray ‘12
Long queries

TaintDroid ‘10
No native code

Backtracker ‘03
Coarse-grained causality
Parallelize DIFT
Parallelizing DIFT is HARD

A = read()
B = read()
C = A + B
D = X + Y
E = C
B = 0
Z = A[D]
F = E
write(F)

Sequential Dependencies
Parallelizing DIFT is HARD

Speck (ASPLOS ’08)

the parallel taint tracker outperforms the sequential version. With 8 cores, the parallel version achieves a 2x speedup compared to its sequential counterpart.

Parallel Lifeguards (ASPLOS ‘08)

As shown in Figure 10, we achieve 1.2X–3.4X speedup with 8 workers.
Parallelizing DIFT is HARD

Speck (ASPLOS ‘08)

“Embarrassingly Sequential”
- Ruwase et al.

As shown in Figure 10, we achieve 1.2X–3.4X speedup with 8 workers.
JetStream

Local DIFT – epoch parallelism

Faster than original execution!

2x $\rightarrow$ 21x
Outline

- Motivation and Introduction
- Design of JetStream
  - Local DIFT
  - Aggregation
- Evaluation
Debugging Query
Local DIFT

Time slice execution into Epochs
Local DIFT

Leverage Record and Replay to calculate DIFT in parallel
Local DIFT

Track mapping between all intermediate locations
Local DIFT

Mapping is too expensive to calculate:
- log operations
- defer calculating relationships until aggregation
DIFT

- Fast Forward: replay execution until start of epoch
- Analysis: log operations using a graph

A = read()
B = read()
C = A + B
D = B + C
C = A[D]
D = A
write(D)

Fast Forward

Analysis
DIFT

- Fast Forward: replay execution until start of epoch
- Analysis: log operations using a graph

\[
\begin{align*}
A &= \text{read()} \\
B &= \text{read()} \\
C &= A + B \\
D &= B + C \\
C &= A[D] \\
D &= A \\
\text{write}(D)
\end{align*}
\]
DIFT

- Fast Forward: replay execution until start of epoch
- Analysis: log operations using a graph

A = read()
B = read()
C = A + B
\[
D = B + C
\]
C = A[D]
D = A
write(D)
Local DIFT output
JetStream

Local DIFT – epoch parallelism

Aggregation – pipeline parallelism
Aggregation

Calculate paths between source and sinks
- Many nodes are not on path between source and sink
- Use sequential information to prune work
Forward Pass

Pass locations which are derived from a source
Forward Pass

Pass locations which are derived from a source.
Backward Pass

Pass locations which are used by a sink
Backward Pass

Pass locations which are used by a sink
In the paper:
- Insights about why naïve approaches fail
- Partitioning – challenging to predict the local DIFT time
- Pre-pruning – garbage collection of the graph
Outline

• Motivation and Introduction
• Design of JetStream
  • Local DIFT
  • Aggregation
• Evaluation
Experimental Setup

- CloudLab cluster of 32 machines, 1–128 cores

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Sequential DIFT Time (Minutes)</th>
<th>Sources (millions)</th>
<th>Sinks (millions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ghostscript</td>
<td>1.3</td>
<td>3</td>
<td>0.2</td>
</tr>
<tr>
<td>Gzip</td>
<td>1.8</td>
<td>64</td>
<td>488</td>
</tr>
<tr>
<td>Evince</td>
<td>3.9</td>
<td>10</td>
<td>104</td>
</tr>
<tr>
<td>Nginx</td>
<td>3.3</td>
<td>10</td>
<td>35</td>
</tr>
<tr>
<td>Mongodb</td>
<td>5.2</td>
<td>9</td>
<td>117</td>
</tr>
<tr>
<td>OpenOffice</td>
<td>7.0</td>
<td>10</td>
<td>32</td>
</tr>
<tr>
<td>Firefox</td>
<td>30.6</td>
<td>0.9</td>
<td>2</td>
</tr>
</tbody>
</table>
## Experimental Setup

- CloudLab cluster of 32 machines, 1–128 cores

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Sequential DIFT Time (Minutes)</th>
<th>Sources (millions)</th>
<th>Sinks (millions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ghostscript</td>
<td>1.3</td>
<td>3</td>
<td>0.2</td>
</tr>
<tr>
<td>Gzip</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evince</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nginx</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mongodb</td>
<td>5.2</td>
<td>9</td>
<td>117</td>
</tr>
<tr>
<td>OpenOffice</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Firefox</td>
<td>30.6</td>
<td>0.9</td>
<td>2</td>
</tr>
</tbody>
</table>

sources: home directory
sinks: all

sources: Cookies
sinks: suspicious connections
Two Different Scenarios

- Unexpected analysis:
  - prioritize low record overhead

- Expected analysis
  - periodic checkpoint
  - gather partitioning stats
Scalability of Unexpected Analysis

Normalized Speedup

Number of Cores

Gzip

Ghostscript

Evince

Mongodb

Ideal

mean: 13x
Scalability of Expected Analysis

![Graph showing normalized speedup across different number of cores for various applications, with an average speedup of 21x.](image)

- **Gzip**
- **Ghostscript**
- **Evince**
- **Mongodb**
- **Nginx**
- **OpenOffice**
- **Firefox**
- **Ideal**

**Normalized Speedup**

**Number of Cores**

Mean: 21x
JetStream

Local DIFT – epoch parallelism

Faster than original execution!

2x → 21x
Questions
Related Work

Epoch Parallelism

Local DIFT
• Ruwase et al. “Parallelizing Dynamic Information Flow Tracking”
• Nightengale et al. “Parallelizing security checks on commodity hardware”
# Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Replay Time (seconds)</th>
<th>JetStream Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ghostscript</td>
<td>1.0</td>
<td>5.6</td>
</tr>
<tr>
<td>Gzip</td>
<td>3.0</td>
<td>2.3</td>
</tr>
<tr>
<td>Evince</td>
<td>13.5</td>
<td>19.5</td>
</tr>
<tr>
<td>Nginx</td>
<td>4.8</td>
<td>5.6</td>
</tr>
<tr>
<td>Mongodb</td>
<td>22.8</td>
<td>13.8</td>
</tr>
<tr>
<td>OpenOffice</td>
<td>7.6</td>
<td>25.0</td>
</tr>
<tr>
<td>Firefox</td>
<td>67.4</td>
<td>94.4</td>
</tr>
</tbody>
</table>
Aggregation Results

![Bar chart showing time comparisons between different applications for Backwards Pass and Both Passes Time.](image-url)
Gzip

Gzip First Query

Gzip Second Query

- Fast Forward
- Instrumentation
- Analysis
- Pre-prune
- Forward Pass
- Prune
- Backward Pass
OpenOffice

OpenOffice First Query

OpenOffice Second Query

- Fast Forward
- Instrumentation
- Analysis
- Forward Pass
- Prune
- Backward Pass
- Pre-prune