Sifter: Scalable Sampling for Distributed Traces, without Feature Engineering

Pedro Las-Casas

with Giorgi Papakerashvili, Vaastav Anand and Jonathan Mace
Sifter: a sampler for distributed traces
Part of distributed tracing backends
Problem: too many traces

Biased trace sampling
Which traces should we keep?
Which traces should we discard?
What constitutes an “interesting” trace?
Distributed Tracing

Uber
Jaeger

Google Dapper

Twitter

OpenTracing

Zipkin

Facebook Canopy
Distributed Trace

An end-to-end recording of one request
Distributed Trace
An end-to-end recording of one request
Each request generates a new trace
Distributed Trace
An end-to-end recording of one request
Each request generates a new trace

Traces with different execution paths == Traces with different structure
Distributed Trace

An end-to-end recording of one request

Each request generates a new trace
- Diagnosing latency problems
- Investigating bugs
Sampling

Trace sampling
Individual traces can be very detailed
Tracing every request = too much data

Uniform random sampling

![Graph showing frequency vs. request latency with trace sampling and uniform random sampling illustrated]
Biased Sampling

Adjust sampling probability based on how “interesting” trace is

- **Uncommon cases**
  - Infrequently seen
  - Interesting
  - High probability

- **Common-cases**
  - Frequently seen
  - Not very interesting
  - Low probability
Biased Sampling

Adjust sampling probability based on how “interesting” trace is

Sample traces across latency distribution
Sifter: a sampler for distributed traces

Part of distributed tracing backends
Biased trace sampling
Use traces to model the system’s behaviors
Low-dimensional probabilistic model forces approximation
Challenges

Operational requirements
Continuous operation over a stream of traces
Low overhead per sampling decision
Large volume of traces

What is an interesting trace?
Lack of standard techniques or metrics
Feature engineering is undesirable
Differences manifest structurally

If two traces are conceptually different then they will also differ in their events, spans, timing, and ordering.
Differences manifest structurally

If two traces are conceptually different then they will also differ in their events, spans, timing, and ordering

Sifter’s approach:
Unsupervised sampling decisions
Directly on trace data
No pre-defined high-level features
Sifter: Trace Representation
Sifter: Trace Representation

We rely on the system’s source code information for the events.
Sifter: Trace Representation

We represent our traces as a directed acyclic graph (DAG), instead of a span.
Sifter: Trace Representation

We represent our traces as a directed acyclic graph (DAG), instead of a span.

- Labels aren’t unique
- Same line of code can execute multiple times
Sifter: Probabilistic Modeling

Traces are examples
Each trace executes some code paths
The stream of traces tell us path frequencies
Use traces to build a probabilistic model

Unbiased model
Sifter sees all traces, regardless of sampling decision
Unbiased model can identify outliers to sample
Sifter Workflow

(1) Receive trace

(2) Convert it into a DAG
Sifter Workflow

1. Receive trace
2. Convert it into a DAG
3. Extract all N-length paths
Sifter Workflow

1. Use paths as input to Sifter’s model.

2. Sifter’s internal model.

3. Model outputs a prediction of the middle event in the path.
Sifter Workflow

(7) **Loss**

*Error between predictions labels and actual labels*

(8) **Backpropagation**

*updates model weights incorporates new trace*
Sifter Workflow

(1) Sampling probability

(2) loss

(3) Backpropagation

(4) previous K traces

(5) this trace

(6) highest sampling probability

(7) lowest sampling probability
Sifter Workflow

(1) Sifter sees all traces, regardless of sampling decision
(2) Every trace updates the model
(3) Unbiased model can identify outliers to sample
(4) No pretraining necessary

Unbiased model

loss

Backpropagation
Evaluation

Operational requirements
Is Sifter fast?
Does Sifter scale?

What is an interesting trace?
Do we detect uncommon and outlier traces?
Can we manage imbalanced classes?
Evaluation

Sifter’s implementation using Tensorflow

DeathStarBench social network benchmark

Hadoop Distributed File System

Production traces
Operational requirements

Is Sifter fast?
Does Sifter scale?

Sifter’s internal state is explicitly constrained

Computational cost depends only on:
(1) number of paths in the trace
(2) number of unique labels in the trace

Sampling latencies range from 3 and 20 milliseconds

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Avg. Labels</th>
<th>Avg. Walks</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFS</td>
<td>38</td>
<td>2547</td>
</tr>
<tr>
<td>DeathStar</td>
<td>82</td>
<td>155</td>
</tr>
<tr>
<td>Production</td>
<td>56</td>
<td>130</td>
</tr>
</tbody>
</table>
Does Sifter detect uncommon and outlier traces?

Replay a stream of traces
Inject traces from unrepresented / underrepresented classes

Known features:
(1) different API types
(2) parameters to API calls
(3) known errors / exceptions
Does Sifter detect uncommon and outlier traces?

995 HDFS read API calls  5 HDFS write API calls

1% sampling rate
How does Sifter manage imbalanced classes?

Production traces - 10,000 traces in 5 different classes

- Random
- Sifter
- Hierarchical Clustering

API-1
API-2
API-3
API-4
API-5

Ideal
How does Sifter manage imbalanced classes?

Production traces - 10,000 traces in 5 different classes

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean-squared error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sifter</td>
<td>35.95</td>
</tr>
<tr>
<td>Hierarchical Clustering</td>
<td>193.12</td>
</tr>
<tr>
<td>Random</td>
<td>283.60</td>
</tr>
</tbody>
</table>

API-1 | API-2 | API-3 | API-4 | API-5

0 10 20 30 40 50
How does Sifter manage imbalanced classes?

Production traces - 10,000 traces in 5 different classes

Sifter’s error is more than 5 times smaller than the hierarchical clustering approach

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean-squared error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sifter</td>
<td>35.95</td>
</tr>
<tr>
<td>Hierarchical Clustering</td>
<td>193.12</td>
</tr>
<tr>
<td>Random</td>
<td>283.60</td>
</tr>
</tbody>
</table>
Side effect: clustering traces
Some other results obtained by Sifter

Adapts over time
Some other results obtained by Sifter

Adapts over time

Structure discriminates!
Biased trace sampling
What constitutes an “interesting” trace?
Efficient + Scalable

Sifter: a sampler for distributed traces
Use traces to model the system’s behaviors
Low-dimensional probabilistic model forces approximation
Sifter: Scalable Sampling for Distributed Traces, without Feature Engineering

ACM Symposium on Cloud Computing (SoCC), 2019
Pedro Las-Casas, Giorgi Papakerashvili, Vaastav Anand, Jonathan Mace

Thank you!

Questions?