Skua: Extending Distributed Tracing Vertically into the Linux Kernel

Harshal Sheth and Andrew Sun
Distributed Systems

• Complex applications are no longer monolithic
  • Modular/agile development
  • Continuous deployment
  • Independent scaling
Distributed Systems

• Complex applications are no longer monolithic
  • Modular/agile development
  • Continuous deployment
  • Independent scaling
• Increasingly seen in large companies
• Hard to debug

Twitter, 2013
Example Distributed System

Legend
- request
- application within a distributed system
- request pathway
Example Distributed System

Legend
- request
- application within a distributed system
- request pathway
Example Distributed System

Legend

- request
- application within a distributed system
- request pathway
Example Distributed System

Legend

- request
- application within a distributed system
- request pathway
Example Distributed System

Legend
- request
- application within a distributed system
- request pathway
Example Distributed System

Legend
- request
- application within a distributed system
- request pathway
Example Distributed System

Legend
- Green arrow: request
- Blue circle: application within a distributed system
- Orange arrow: request pathway

Frontend → Web Results → PageRank
Frontend ← Images ← VisualRank
Example Distributed System

Legend
- request
- application within a distributed system
- request pathway

Frontend -> Web Results -> PageRank
Frontend -> Images
Frontend -> VisualRank

(request pathway)
Example Distributed System

Legend
- request
- application within a distributed system
- request pathway
Example Distributed System

Legend
- request
- application within a distributed system
- request pathway
Example Distributed System

Legend
- green triangle: request
- blue circle: application within a distributed system
- orange arrow: request pathway
Example Distributed System

Legend
- request
- application within a distributed system
- request pathway
Example Distributed System

Legend
- request
- application within a distributed system
- request pathway
Distributed Tracing

- Monitoring and troubleshooting distributed systems
Distributed Tracing

• Monitoring and troubleshooting distributed systems
  • Discovering latency issues
Distributed Tracing

• Monitoring and troubleshooting distributed systems
  • Discovering latency issues
  • Graphing service dependencies
Distributed Tracing

• Monitoring and troubleshooting distributed systems
  • Discovering latency issues
  • Graphing service dependencies
  • Root-cause analysis of backend issues
Distributed Tracing

- Monitoring and troubleshooting distributed systems
  - Discovering latency issues
  - Graphing service dependencies
  - Root-cause analysis of backend issues
  - Tracing a specific request through the entire system
What does distributed tracing miss?
What does distributed tracing miss?

• There’s more to performance than meets the eye of existing distributed tracing tools
What does distributed tracing miss?

- There’s more to performance than meets the eye of existing distributed tracing tools
  - Contention between applications
  - Kernel bugs
  - Security patches (e.g. Meltdown/Spectre)
What does distributed tracing miss?

• There’s more to performance than meets the eye of existing distributed tracing tools
  • Contention between applications
  • Kernel bugs
  • Security patches (e.g. Meltdown/Spectre)
• Can we gain visibility regarding these issues via the kernel?
Our goal: extend distributed tracing into the kernel
Our Approach

Jaeger
distributed tracing
framework from Uber
Our Approach

Jaeger
distributed tracing framework from Uber

LTTng
Linux kernel trace toolkit
Our Approach

Jaeger
distributed tracing
framework from Uber

+ LTTng
Linux kernel
trace toolkit

= Skua
Traces and Spans

Frontend Request
Trace ID: 1337
Parent ID: (none)
Span ID: 7893

Time

Frontend
Traces and Spans

Frontend Request
Trace ID: 1337
Parent ID: (none)
Span ID: 7893

Frontend
Traces and Spans

Frontend

Request
Trace ID: 1337
Parent ID: (none)
Span ID: 7893
Traces and Spans

Frontend Request
Trace ID: 1337
Parent ID: (none)
Span ID: 7893

Web Results
Trace ID: 1337
Parent ID: 7893
Span ID: 3460
Traces and Spans

Context: trace ID parent ID span ID sampled

Frontend Request
- Trace ID: 1337
- Parent ID: 7893
- Span ID: 3460

Frontend

Web Results

PageRank

Web Results
- Trace ID: 1337
- Parent ID: 3460
- Span ID: 1231

PageRank
Traces and Spans

Context:
- Trace ID
- Parent ID
- Span ID
- Sampled

Frontend Request
- Trace ID: 1337
- Parent ID: 7893
- Span ID: 3460

Web Results
- Trace ID: 1337
- Parent ID: 3460
- Span ID: 1231

PageRank
- Trace ID: 1337
- Parent ID: 3460
- Span ID: 1231

Images
- Trace ID: 1337
- Parent ID: 7893
- Span ID: 8652

Web Results
- Trace ID: 1337
- Parent ID: 7893
- Span ID: 8652
Traces and Spans

- Frontend
- Web Results
- PageRank
- Images
- VisualRank

**Context:**
- trace ID
- parent ID
- span ID
- sampled

**Time:**
- Frontend Request
  - Trace ID: 1337
  - Parent ID: 7893
  - Span ID: 3460
- Web Results
  - Trace ID: 1337
  - Parent ID: 7893
  - Span ID: 3460
- PageRank
  - Trace ID: 1337
  - Parent ID: 3460
  - Span ID: 1231
- Images
  - Trace ID: 1337
  - Parent ID: 7893
  - Span ID: 8652
- VisualRank
  - Trace ID: 1337
  - Parent ID: 8652
  - Span ID: 3460
Traces and Spans

Frontend Request
Trace ID: 1337
Parent ID: (none)
Span ID: 7893

Web Results
Trace ID: 1337
Parent ID: 7893
Span ID: 3460

PageRank
Trace ID: 1337
Parent ID: 3460
Span ID: 1231

Images
Trace ID: 1337
Parent ID: 7893
Span ID: 8652

VisualRank
Trace ID: 1337
Parent ID: 8652
Span ID: 3460

Context:
trace ID
parent ID
span ID
sampled
Traces and Spans

Frontend

- Web Results
  - Trace ID: 1337
  - Parent ID: 8652
  - Span ID: 3460

- Images
  - Trace ID: 1337
  - Parent ID: 7893
  - Span ID: 8652

- VisualRank
  - Trace ID: 1337
  - Parent ID: 3460
  - Span ID: 1231

Span Aggregator

Time

Frontend Request
- Trace ID: 1337
- Parent ID: 7893
- Span ID: 3460

Web Results
- Trace ID: 1337
- Parent ID: 7893
- Span ID: 3460

PageRank
- Trace ID: 1337
- Parent ID: 3460
- Span ID: 1231

Images
- Trace ID: 1337
- Parent ID: 7893
- Span ID: 8652

VisualRank
- Trace ID: 1337
- Parent ID: 8652
- Span ID: 3460

Context: trace ID, parent ID, span ID sampled
Jaeger

https://eng.uber.com/distributed-tracing/
Skua Design

Jaeger Client  User Application
Skua Design

Jaeger Client

User Application

Jaeger Framework
Skua Design

Jaeger Client

User Application

Jaeger Framework

syscalls

Kernel Events

Linux Kernel
Skua Design

Jaeger Client

User Application

Jaeger Framework

syscalls

Kernel Events

LTTng Kernel Modules

Linux Kernel
Skua Design

Jaeger Client

User Application

Kernel Module using procfs

Kernel Events

LTTng Kernel Modules

Linux Kernel

Jaeger Framework

syscalls
Skua Design

Jaeger Client

User Application

Kernel Module using procfs

Kernel Events

task_struct

Jaeger Framework

LTTng Kernel Modules

Linux Kernel
Skua Design
Skua Design

Jaeger Client

User Application

Kernel Module using procfs

task_struct

Linux Kernel

Jaeger Framework

LTTng Adapter

LTTng Kernel Modules

Kernel Events

syscalls
Skua Design

1. Jaeger Client
2. Kernel Module using procfs
3. Kernel Events
4. task_struct
5. LTTng Kernel Modules
6. LTTng Adapter

Jaeger Application

Kernel Framework

Linux Kernel
Skua Details

• Jaeger C++ client sends its context into the kernel 25 LOC
Skua Details

• Jaeger C++ client sends its context into the kernel

• Treats the Linux kernel as the next level of the span hierarchy
  • Each syscall is considered a span
  • Tracepoint events become span logs
Skua Details

- Jaeger C++ client sends its context into the kernel
- Treats the Linux kernel as the next level of the span hierarchy
  - Each syscall is considered a span
  - Tracepoint events become span logs
- LTTng kernel modules tag each span and log with the context information
Skua Details

- Jaeger C++ client sends its context into the kernel
- Treats the Linux kernel as the next level of the span hierarchy
  - Each syscall is considered a span
  - Tracepoint events become span logs
- LTTng kernel modules tag each span and log with the context information
- Custom adapter sends kernel data into the Jaeger
Evaluation
Correctness

Setup

• Small C++ program
  • Spawns a few threads
  • Makes 10 different syscalls
• Verifies that Skua is correctly recording syscalls
Correctness

Setup

• Small C++ program
• Spawns a few threads
• Makes 10 different syscalls
• Verifies that Skua is correctly recording syscalls

Results

• Syscalls recorded in Jaeger as spans
• Misses a few syscalls
  • vDSO — gettimeofday
  • LTTng instrumentation
• Tracepoint events recorded properly as logs
Performance Overhead

- Created a small C++ Web server using uWebSockets
Performance Overhead

- Created a small C++ Web server using uWebSockets
- Used benchmarking tool autocannon
Performance Overhead

- Created a small C++ Web server using uWebSockets
- Used benchmarking tool autocannon
- Evaluated throughput and latency under different tracing scenarios
Performance Overhead

- Created a small C++ Web server using uWebSockets
- Used benchmarking tool autocannon
- Evaluated throughput and latency under different tracing scenarios
- 1,000,000 requests, 10 connections
Performance Overhead

• Created a small C++ Web server using uWebSockets
• Used benchmarking tool autocannon
• Evaluated throughput and latency under different tracing scenarios
• 1,000,000 requests, 10 connections
• Traced 0.1% of requests
Web Server Throughput

Requests/Sec

Benchmark Scenario

No Tracing | Jaeger | LTTng | Skua

Web Server Latency

Latency (ms)

Benchmark Scenario

No Tracing | Jaeger | LTTng | Skua
### Web Server Throughput

<table>
<thead>
<tr>
<th>Benchmark Scenario</th>
<th>No Tracing</th>
<th>Jaeger</th>
<th>LTTng</th>
<th>Skua</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests/Sec</td>
<td>11628</td>
<td>11111</td>
<td>11111</td>
<td>11111</td>
</tr>
</tbody>
</table>

### Web Server Latency

<table>
<thead>
<tr>
<th>Benchmark Scenario</th>
<th>No Tracing</th>
<th>Jaeger</th>
<th>LTTng</th>
<th>Skua</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency (ms)</td>
<td>0.33</td>
<td>0.45</td>
<td>36.3%</td>
<td>4.0%</td>
</tr>
</tbody>
</table>
### Web Server Throughput

- **No Tracing**: 11628 Requests/Sec
- **Jaeger**: 11111 Requests/Sec (4.0% slower)
- **LTTng**: 10310 Requests/Sec (11.3% slower)
- **Skua**: 11628 Requests/Sec

### Web Server Latency

- **No Tracing**: 0.33 ms (0.4% slower)
- **Jaeger**: 0.45 ms (24.2% slower)
- **LTTng**: 0.45 ms (24.2% slower)
- **Skua**: 0.33 ms
### Web Server Throughput

<table>
<thead>
<tr>
<th>Benchmark Scenario</th>
<th>No Tracing</th>
<th>Jaeger</th>
<th>LTTng</th>
<th>Skua</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests/Sec</td>
<td>11628</td>
<td>11111</td>
<td>10310</td>
<td>10205</td>
</tr>
<tr>
<td>4.0%</td>
<td>11.3%</td>
<td>12.2%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Web Server Latency

<table>
<thead>
<tr>
<th>Benchmark Scenario</th>
<th>No Tracing</th>
<th>Jaeger</th>
<th>LTTng</th>
<th>Skua</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency (ms)</td>
<td>0.33</td>
<td>0.45</td>
<td>0.41</td>
<td>0.53</td>
</tr>
<tr>
<td>36.3%</td>
<td>24.2%</td>
<td>60.6%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusions

- Can use distributed tracing to monitor and debug complex distributed systems
Conclusions

• Can use distributed tracing to monitor and debug complex distributed systems

• Current open source distributed tracing frameworks miss kernel information
Conclusions

• Can use distributed tracing to monitor and debug complex distributed systems

• Current open source distributed tracing frameworks miss kernel information

• Skua integrates LTTng kernel data with Jaeger tracing
Conclusions

• Can use distributed tracing to monitor and debug complex distributed systems

• Current open source distributed tracing frameworks miss kernel information

• Skua integrates LTTng kernel data with Jaeger tracing

• Skua has some impact on throughput and latency
Conclusions

• Can use distributed tracing to monitor and debug complex distributed systems

• Current open source distributed tracing frameworks miss kernel information

• Skua integrates LTTng kernel data with Jaeger tracing

• Skua has some impact on throughput and latency

• “[12.2%] ought to be [good] enough for anybody” —Bill Gates, paraphrased
Conclusions

• Can use distributed tracing to monitor and debug complex distributed systems

• Current open source distributed tracing frameworks miss kernel information

• Skua integrates LTTng kernel data with Jaeger tracing

• Skua has some impact on throughput and latency

• “[12.2%] ought to be [good] enough for anybody”
  —Bill Gates, paraphrased
Acknowledgements

- Raja Sambasivan - Mentor

MirPRimes
Backup Slides
Syscalls made by correctness tester

- getpid
- getppid
- gettid
- gettimeofday
- nanosleep
- open
- close
- write
- fstat
- futex
Performance Benchmark Details

- Tests run on KVM virtual machine assigned 24 vCPUs (2 × Intel Xeon X5670), 16 GB RAM, with latest Arch Linux updates

<table>
<thead>
<tr>
<th>Benchmark Scenario</th>
<th>Program Instrumentation</th>
<th>Kernel Tracing via LTTng</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Tracing</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>Unmodified Jaeger</td>
<td>Original Jaeger Client</td>
<td>No</td>
</tr>
<tr>
<td>Modified Jaeger</td>
<td>Modified Jaeger Client</td>
<td>No</td>
</tr>
<tr>
<td>LTTng</td>
<td>N/A</td>
<td>Yes, but output ignored</td>
</tr>
<tr>
<td>Skua</td>
<td>Modified Jaeger Client</td>
<td>Yes, output sent to adapter</td>
</tr>
</tbody>
</table>
### Web Server Throughput

<table>
<thead>
<tr>
<th>Benchmark Scenario</th>
<th>Requests/Sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Tracing</td>
<td>11628</td>
</tr>
<tr>
<td>Unmodified Jaeger</td>
<td>11629</td>
</tr>
<tr>
<td>Modified Jaeger</td>
<td>11111</td>
</tr>
<tr>
<td>LTTng</td>
<td>10310</td>
</tr>
<tr>
<td>Skua</td>
<td>10205</td>
</tr>
</tbody>
</table>

### Web Server Latency

<table>
<thead>
<tr>
<th>Benchmark Scenario</th>
<th>Latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Tracing</td>
<td>0.33</td>
</tr>
<tr>
<td>Unmodified Jaeger</td>
<td>0.36</td>
</tr>
<tr>
<td>Modified Jaeger</td>
<td>0.45</td>
</tr>
<tr>
<td>LTTng</td>
<td>0.41</td>
</tr>
<tr>
<td>Skua</td>
<td>0.53</td>
</tr>
</tbody>
</table>
Skua Implementation Details

User Application

Modified Jaeger Client Library

Instrumentation using the OpenTracing API

Linux Kernel with modified LTTng Kernel Modules

jaeger-agent

LTTng-adapter running jaeger-client-go

jaeger-collector

backend database

babeltrace

jaeger-ui jaeger-query
Existing Tracing Frameworks

- Dapper (2010) — Google
- Zipkin (2012) — Twitter
- Canopy (2017) — Facebook
- Jaeger (2017) — open sourced by Uber
Distributed Systems

AWS (2008)  
Netflix (2012)  
Twitter (2013)

AWS: https://twitter.com/werner/status/741673514567143424  
Netflix: https://www.slideshare.net/BruceWong3/the-case-for-chaos  
Twitter: https://blog.twitter.com/engineering/en_us/a/2013/observability-at-twitter.html