Tracking Causal Order in AWS Lambda Applications

Wei-Tsung Lin, Chandra Krintz, Rich Wolski, and Michael Zhang
Dept. of Computer Science, UC Santa Barbara
Xiaogang Cai, Tongjun Li, and Weijin Xu
Huawei Technologies Co. Inc.

IC2E 2018
AWS Lambda

• Serverless computing platform
• No resource provisioning needed, hence simplifies cloud applications deployment
• Stateless functions interacting with other cloud services
• Billed by runtime duration and memory use, enables scalable distributed applications at low cost
Challenges

• Difficult to debug, analyze, reason about

• Tooling for serverless applications is nascent with only simple logging services available
  o CloudWatch
  o X-Ray
Tools and limitations

- **CloudWatch**
  - Runtime duration, memory usage, customized information
  - No causality information
  - Difficult to distinguish concurrent invocations

- **X-Ray**
  - Presents dependency trees as service graph
  - Doesn’t track implicit relationship
  - Doesn’t track dependency across regions
  - Statistical sampling leads to record loss
West Region X-Ray Management Console

East Region X-Ray Management Console
(Logs and service graphs are separate and disconnected from those above in another AWS region)
Tools and limitations

• **CloudWatch**
  + Runtime duration, memory usage, customized information
    - No causality information
    - Difficult to distinguish concurrent invocations

• **X-Ray**
  + Presents dependency trees as service graph
    - **Doesn’t track implicit relationship**
    - Doesn’t track dependency across regions
    - Statistical sampling leads to record loss
X-Ray service graph

West Region X-Ray Management Console

East Region X-Ray Management Console
(Logs and service graphs are separate and disconnected from those above in another AWS region)
Tools and limitations

- **CloudWatch**
  - Runtime duration, memory usage, customized information
  - No causality information
  - Difficult to distinguish concurrent invocations

- **X-Ray**
  - Presents dependency trees as service graph
  - Doesn’t track implicit relationship
  - **Doesn’t track dependency across regions**
  - Statistical sampling leads to record loss
X-Ray service graph

West Region X-Ray Management Console

East Region X-Ray Management Console
(Logs and service graphs are separate and disconnected from those above in another AWS region)
Tools and limitations

- **CloudWatch**
  - Runtime duration, memory usage, customized information
  - No causality information
  - Difficult to distinguish concurrent invocations

- **X-Ray**
  - Presents dependency trees as service graph
  - Doesn’t track implicit relationship
  - Doesn’t track dependency across regions
  - Statistical sampling leads to record loss
Alternative: GammaRay

- Tracking causal order across all services and regions
- Automatically instrument Lambda functions and AWS SDK
- Compute performance statistics and construct service graph offline
- No record loss
GammaRay components

• **Lambda Deployment tool**
  - Injects GammaRay instrumentation to capture and report events
  - Packs source codes, needed libraries, and runtime support as a zip file

• **Runtime support**
  - Replace the function entry point with a function wrapper
  - Assume control when Lambda handler or AWS SDK is invoked
  - Assign an unique ID to root event and pass it to all downstream events
  - Capture events and send them to shared DynamoDB table synchronously

• **Event processing engine**
  - Construct a service graph using DynamoDB stream offline
GammaRay components

• Lambda Deployment tool
  o Injects GammaRay instrumentation to capture and report events
  o Packs source codes, needed libraries, and runtime support as a zip file

• Runtime support
  o Replace the function entry point with a function wrapper
  o Assume control when Lambda function or AWS SDK is invoked
  o Assign an unique ID to root event and pass it to all downstream events
  o Capture events and send them to shared DynamoDB table synchronously

• Event processing engine
  o Construct a service graph using DynamoDB stream offline
How it works

GammaRay Wrapper

Entry

Lambda Handler

Before

SDK Calls

After

Optional

Dynamo DB Table

Exit

Optional
Instrumentation injection

- **Dynamic**
  - “Monkey patches” AWS SDK calls made by the function to invoke the GammaRay runtime before and after the call
  - [https://github.com/racker/fleece](https://github.com/racker/fleece)

- **Static**
  - Replacing SDK to be imported with modified version
  - Increase memory footprint

- **Hybrid**
  - Lighter version of dynamic patching
  - Only SDK calls that can trigger other events are captured
  - Relies on X-Ray for performance data gathering
GammaRay components

• Lambda Deployment tool
  o Injects GammaRay instrumentation to capture and report events
  o Packs source codes, needed libraries, and runtime support as a zip file

• Runtime support
  o Replace the function entry point with a function wrapper
  o Assume control when Lambda handler or AWS SDK is invoked
  o Assign an unique ID to root event and pass it to all downstream events
  o Capture events and send them to shared DynamoDB table synchronously

• Event processing engine
  o Construct a service graph using DynamoDB stream offline
GammaRay service graph
Evaluation

• Applications
  o Map-Reduce
  o ImgProc

• Micro-benchmarks
  o Empty function
  o DynamoDB read/write
  o S3 read/write
  o SNS posting

• Compared to X-Ray with Python SDK logging turned on*

*Fleece: https://github.com/racker/fleece
Application: Map-Reduce

- Dynamic & static: 840 records
- Hybrid: 125 records
Baseline performance (X-Ray)
- Total runtime duration: 114 seconds
- Total memory use: 1231 MB
Application: ImgProc

- Dynamic & static: 18 records
- Hybrid: 5 records
Application: ImgProc

Baseline performance (X-Ray)

- Total runtime duration: 3.1 seconds
- Total memory use: 114 MB
### Micro-benchmarks

<table>
<thead>
<tr>
<th>Overhead (ms)</th>
<th>Startup</th>
<th>DDB Read</th>
<th>DDB Write</th>
<th>S3 Read</th>
<th>S3 Write</th>
<th>SNS</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-Ray</td>
<td>6.0</td>
<td>47.3</td>
<td>47.4</td>
<td>52.0</td>
<td>87.1</td>
<td>64.2</td>
<td>59.6</td>
</tr>
<tr>
<td>G-Ray-H</td>
<td>418.9</td>
<td>1.5</td>
<td>29.5</td>
<td>2.7</td>
<td>19.3</td>
<td>33.8</td>
<td>17.3</td>
</tr>
</tbody>
</table>

- Average of 200 runs, each run contains 100 operations
- Row X-Ray shows the overheads over clean application deployment
- Row G-Ray-H shows the overhead of Hybrid GammaRay over X-Ray
- Obtaining DynamoDB handler takes 126ms in average
Summary

- A tool for debugging and reasoning about AWS Lambda application
- Captured causality across regions and services
- No instrumentation needed for developers
Future works

- Optimizing wrapper startup overhead
- Porting to other clouds
- Asynchronous event reporting
Acknowledgements

• National Science Foundation
• Huawei Technologies Co.
Thank you!

- [https://github.com/MAYHEM-Lab/UCSBFaaS-Wrappers](https://github.com/MAYHEM-Lab/UCSBFaaS-Wrappers)
- weitsung@cs.ucsb.edu