Serverless Application Analytics Framework

Robert Cordingly, Hanfei Yu, Varik Hoang, Zohreh Sadeghi, David Foster, David Perez, Rashad Hatchett, Wes Lloyd

School of Engineering and Technology
University of Washington Tacoma
Sixth International Workshop on Serverless Computing (WoSC6) 2020
Motivation

- Serverless platforms offer many benefits:
  - Simple deployment
  - Automatic scaling
  - Automatic infrastructure management
  - Only billed for actual runtime

- The unpredictable cost of FaaS:
  - (Function Runtime) x (Memory Setting) x (Price)
  - What impacts the runtime of an application?
What is SAAF? - The Inspector

Using SAAF in a Function:

Using SAAF in a function is as simple importing the framework of code. Attributes collected by SAAF will be appended to asynchronous functions, this data could be stored into database for retrieval after the function is finished.

Example Function:

```python
from Inspector import *

def myFunction(request):
    # Initialize the Inspector and collect data
    inspector = Inspector()
    inspector.inspectAll()

    # Add a "Hello World!" message
    inspector.addAttribute("message", "Hello World!")

    # Return attributes collected.
    return inspector.inspectAll()
```

Example Output JSON:

```
{
    "version": 0.2,
    "lang": "python",
    "cpuType": "Intel(R) Xeon(R) Processor @ 2.500GHz",
    "cpuModel": 63,
    "vmuptime": 1551727835,
    "uuid": "d241c618-78d8-48e2-9736-997dc1a931d4",
    "vmID": "tUCmA",
    "platform": "AWS Lambda",
    "newcontainer": 1,
    "cpuUsrDelta": "904",
    "cpuNiceDelta": "0",
    "cpuKrnDelta": "585",
    "cpuIdleDelta": "82428",
    "cpuIowaitDelta": "226",
    "cpuIrqDelta": "0",
    "cpuSoftIrqDelta": "7",
    "vmpostableDelta": "1594",
    "frameworkRuntime": 35.72,
    "message": "Hello Fred Smith!",
    "runtime": 38.94
}
```

Attributes Collected by Each Function:

The amount of data collected is determined by which functions are called. If some attributes are not needed, then some functions may not need to be called. If you would like to collect every attribute, the inspectAll method will run all methods.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>The version of the SAAF Framework.</td>
</tr>
<tr>
<td>lang</td>
<td>The language of the function.</td>
</tr>
<tr>
<td>runtime</td>
<td>The server-side runtime from when the function is initialized until inspector.finish() is called.</td>
</tr>
<tr>
<td>startTime</td>
<td>The Unix Epoch that the Inspector was initialized in.</td>
</tr>
</tbody>
</table>

```python
inspectContainer()

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uuid</td>
<td>A unique identifier assigned to a container if one does not already exist.</td>
</tr>
<tr>
<td>newcontainer</td>
<td>Whether a container is new (no assigned uuid) or if it has been used before.</td>
</tr>
<tr>
<td>vmuptime</td>
<td>Time when the host booted in seconds since January 1, 1970 (Unix epoch).</td>
</tr>
</tbody>
</table>

inspectCPU()

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpuType</td>
<td>The model name of the CPU.</td>
</tr>
<tr>
<td>cpuModel</td>
<td>The model number of the CPU.</td>
</tr>
<tr>
<td>cpuUser</td>
<td>Time spent normally executing CPU instructions.</td>
</tr>
<tr>
<td>cpuNice</td>
<td>Time spent executing CPU instructions that are nice.</td>
</tr>
<tr>
<td>cpuIowait</td>
<td>Time spent executing CPU instructions waiting for I/O.</td>
</tr>
</tbody>
</table>

```
Supported Platforms and Languages

- AWS
- Google Cloud
- Azure
- IBM Cloud
- JavaScript
- Go
SAAF Tools: Publish Script

Example Hello World Function:

```python
from Inspector import *

def myFunction(request):
    # Import the module and collect data
    inspector = Inspector()
    inspector.inspectAll()
    # Add custom message and finish the function
    inspector.addAttribute("message", "Hello ", request['name'] + "!")
    inspector.inspectAllDeletions()
    return inspector.finish()
```

`./publish.sh`
Data collection is directed by calling functions

CPU and Memory metrics are collected from the Linux procfs

Cold/Warm infrastructure state is observed by stamping function instances

Tenancy is determined by introspecting the environment

With another tool we can do more...
SAAF Tools: FaaS Runner

- Client for running experiments
- Executes reproducible tests defined by files or command line arguments
  - Automatically change memory settings or redeploy functions
  - Run functions sequentially or concurrently with many threads
  - Run functions synchronously or asynchronously
  - Define payload distribution and creation with inheritance
  - Execute complex pipelines with multiple functions
  - Run multiple iterations of an experiment
- Automatically compile results into a report
SAAF + FaaS Runner

- Observations made by FaaS Runner:
  - Network latency
  - Round trip time
  - Runtime concurrency
  - Run/thread IDs to trace pipelines
  - Sum/average/lists of attributes returned by functions

- Combining SAAF and FaaS Runner collects a total of 48 metrics
Research with SAAF: Languages Comparison
Languages Comparison Conclusions

- **Best Performance: Load**
- **Low High-Tenancy Impact**
- **Good Memory Scaling**

- **Best Performance: Transform**
- **Low Cold Latency**
- **Low High-Tenancy Impact**

- **Best Performance: Query**
- **Low Cold Latency**
- **Low High-Tenancy Impact**

- **Low High-Tenancy Impact**
- **Low Cold Latency**
- **Good Memory Scaling**
Research with SAAF: Predicting Performance

![Graph showing the cost estimate for different memory sizes on AWS Lambda Function with different CPU types. The graph compares AWS and IBM CPUs across various memory sizes, illustrating the cost differences between NO VPC, a1 CPU, a2 CPU, a3 CPU, and Combined (VPC). The data suggests that using a3 CPU with VPC is the most cost-effective option for larger memory sizes.]
Predicting Performance Scenarios

Runtime = \frac{(\text{cpuUsr} + \text{cpuKrn} + \text{cpuidle} + \text{cpuOWait} + \text{cpuIntSrcv} + \text{cpuSftIntSrcv})}{\text{(\# of cores)}}

- CPU:
  - 256 MBs \(a_1 \rightarrow a_2\)
  - 256 MBs \(a_1 \rightarrow a_3\)
  - 256 MBs \(a_2 \rightarrow a_3\)
  - 512 MBs \(a_1 \rightarrow a_2\)
  - 512 MBs \(a_1 \rightarrow a_3\)
  - 512 MBs \(a_2 \rightarrow a_3\)
  - 1024 MBs \(a_1 \rightarrow a_2\)
  - 1024 MBs \(a_1 \rightarrow a_3\)
  - 1024 MBs \(a_2 \rightarrow a_3\)
  - 2048 MBs \(a_1 \rightarrow a_2\)
  - 2048 MBs \(a_1 \rightarrow a_3\)
  - 2048 MBs \(a_2 \rightarrow a_3\)

- Memory:
  - a1 \(256\text{MBs} \rightarrow 512\text{MBs}\)
  - a1 \(256\text{MBs} \rightarrow 1024\text{MBs}\)
  - a1 \(256\text{MBs} \rightarrow 2048\text{MBs}\)
  - a2 \(256\text{MBs} \rightarrow 512\text{MBs}\)
  - a2 \(256\text{MBs} \rightarrow 1024\text{MBs}\)
  - a2 \(256\text{MBs} \rightarrow 2048\text{MBs}\)
  - a3 \(256\text{MBs} \rightarrow 512\text{MBs}\)
  - a3 \(256\text{MBs} \rightarrow 1024\text{MBs}\)
  - a3 \(256\text{MBs} \rightarrow 2048\text{MBs}\)

- Platform:
  - 256 MBs \(a_1 \rightarrow i_1\)
  - 256 MBs \(a_1 \rightarrow i_2\)
  - 256 MBs \(a_1 \rightarrow i_3\)
  - 256 MBs \(a_1 \rightarrow i_4\)
  - 256 MBs \(a_1 \rightarrow i_1\)
  - 256 MBs \(a_1 \rightarrow i_2\)
  - 256 MBs \(a_1 \rightarrow i_3\)
  - 256 MBs \(a_1 \rightarrow i_4\)
  - 1024 MBs \(a_1 \rightarrow i_1\)
  - 1024 MBs \(a_1 \rightarrow i_2\)
  - 1024 MBs \(a_1 \rightarrow i_3\)
  - 1024 MBs \(a_1 \rightarrow i_4\)
  - 2048 MBs \(a_1 \rightarrow i_1\)
  - 2048 MBs \(a_1 \rightarrow i_2\)
  - 2048 MBs \(a_1 \rightarrow i_3\)
  - 2048 MBs \(a_1 \rightarrow i_4\)
Predicting Performance Conclusions

![Graph showing performance conclusions with data points for different scenarios and configurations. The graph includes bars representing the mean absolute percent error for different conditions, such as 256 MBs, 512 MBs, 1024 MBs, and 2048 MBs, across various CPU to CPU prediction models such as SCNMT2, SCMT2, SCSMT2, and AWS→IBM.]
SAAF’s goal is to enable developers and researchers to make educated observations into the factors that impact performance on FaaS platforms

- **Design goals:**
  - Easy to implement and deploy
  - Low overhead and minimal dependencies
  - Cross platform/language support
  - A complete development workflow with SAAF + FaaS Runner:
    - Development -> Deployment -> Testing -> Data Analysis
  - Available for anyone
Questions or comments?
Please email:
rcording@uw.edu or wlloyd@uw.edu

Download the Serverless Application Analytics Framework:
github.com/wlloyduw/saaf

Paper Link:
https://www.serverlesscomputing.org/wosc6/#p12

This research is supported by NSF Advanced Cyberinfrastructure Research Program (OAC-1849970), NIH grant R01GM126019, and the AWS Cloud Credits for Research program.