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

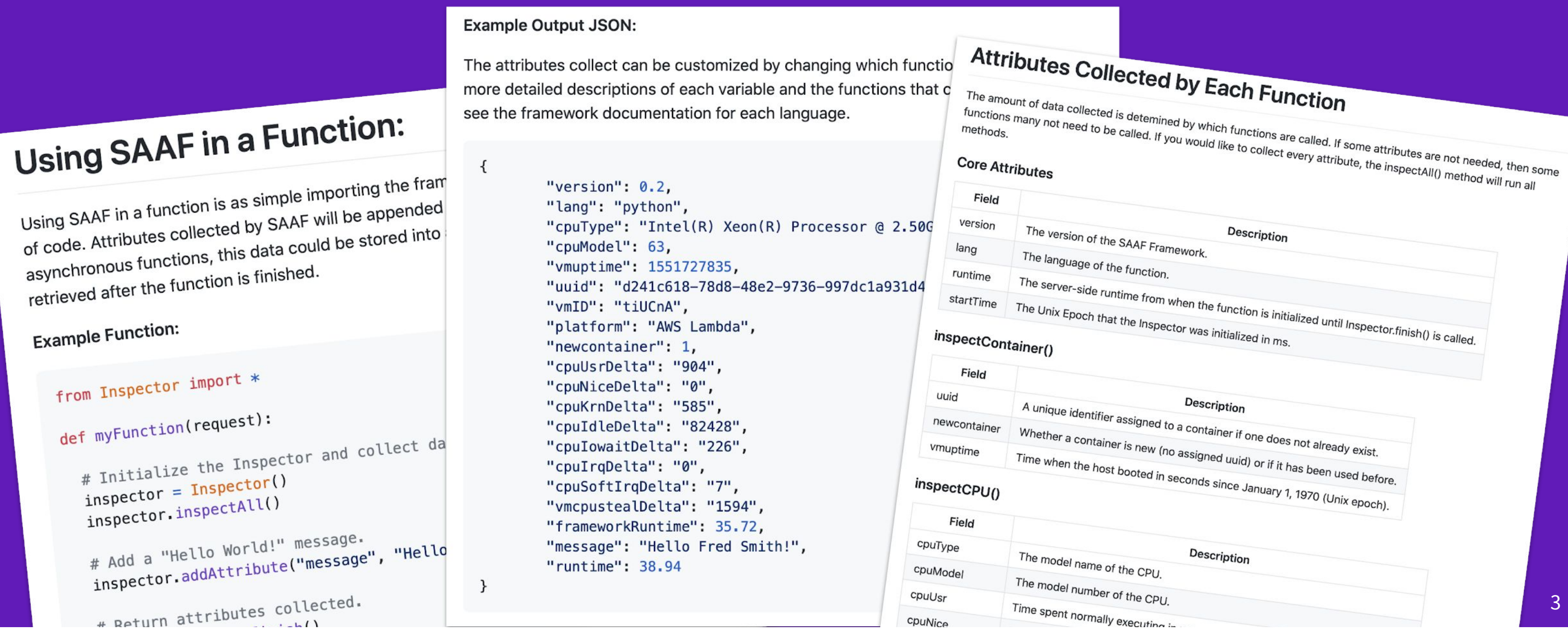
Motivation

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

λ

- 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 fram
of code. Attributes collected by SAAF will be appended
asynchronous functions, this data could be stored into
retrieved after the function is finished.

Example Function:

```
from Inspector import *

def myFunction(request):

    # Initialize the Inspector and collect da
    inspector = Inspector()
    inspector.inspectAll()

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

    # Return attributes collected.
```

Example Output JSON:

The attributes collect can be customized by changing which functio
more detailed descriptions of each variable and the functions that c
see the framework documentation for each language.

```
{
        "version": 0.2,
        "lang": "python",
        "cpuType": "Intel(R) Xeon(R) Processor @ 2.50G
        "cpuModel": 63,
        "vmuptime": 1551727835,
        "uuid": "d241c618-78d8-48e2-9736-997dc1a931d4
        "vmID": "tiUCnA",
        "platform": "AWS Lambda",
        "newcontainer": 1,
        "cpuUsrDelta": "904",
        "cpuNiceDelta": "0",
        "cpuKrnDelta": "585",
        "cpuIdleDelta": "82428",
        "cpuIowaitDelta": "226",
        "cpuIrqDelta": "0",
        "cpuSoftIrqDelta": "7",
        "vmcpustealDelta": "1594",
        "frameworkRuntime": 35.72,
        "message": "Hello Fred Smith!",
        "runtime": 38.94
}
```

Attributes Collected by Each Function

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

Core Attributes

Field	Description
version	The version of the SAAF Framework.
lang	The language of the function.
runtime	The server-side runtime from when the function is initialized until Inspector.finish() is called.
startTime	The Unix Epoch that the Inspector was initialized in ms.

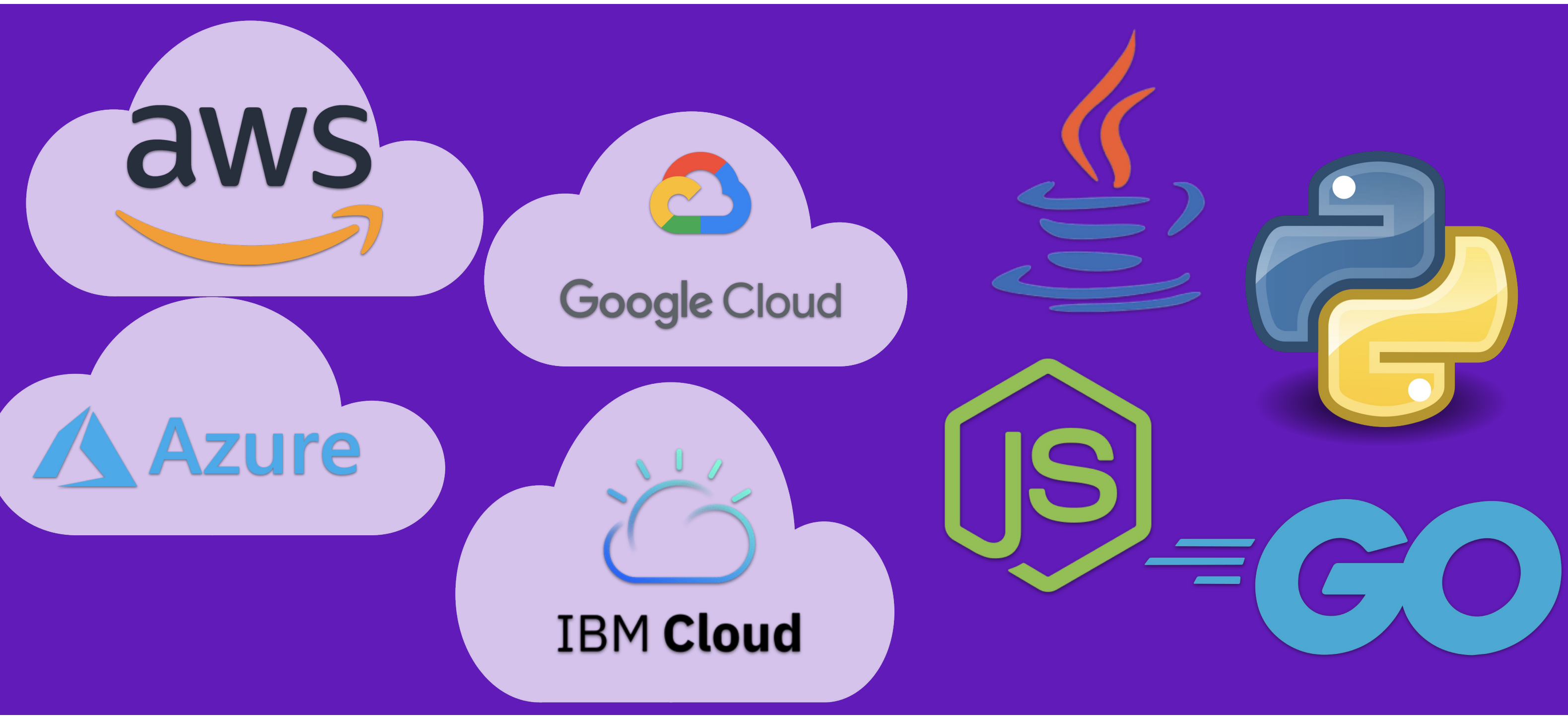
inspectContainer()

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

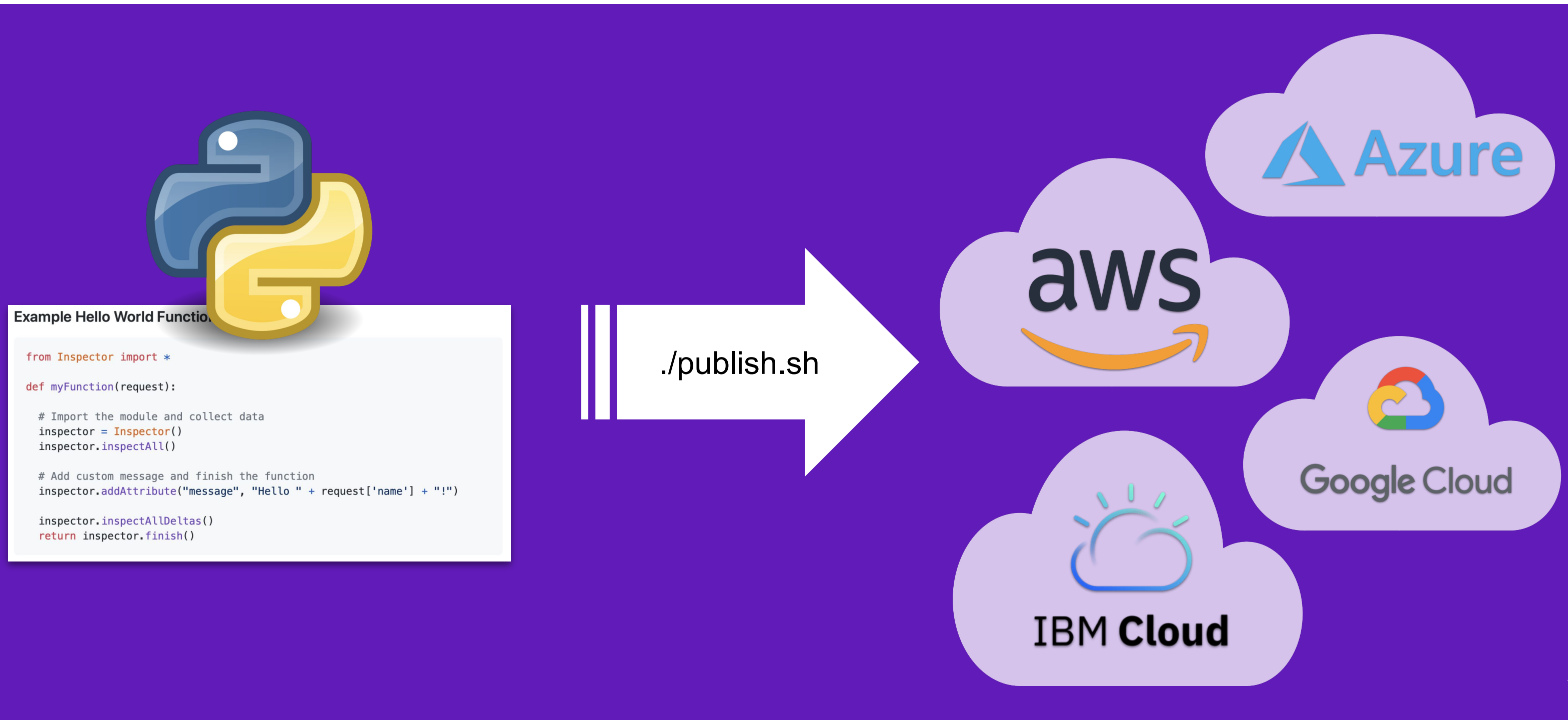
inspectCPU()

Field	Description
cpuType	The model name of the CPU.
cpuModel	The model number of the CPU.
cpuUsr	Time spent normally executing in
cpuNice	

Supported Platforms and Languages



SAAF Tools: Publish Script

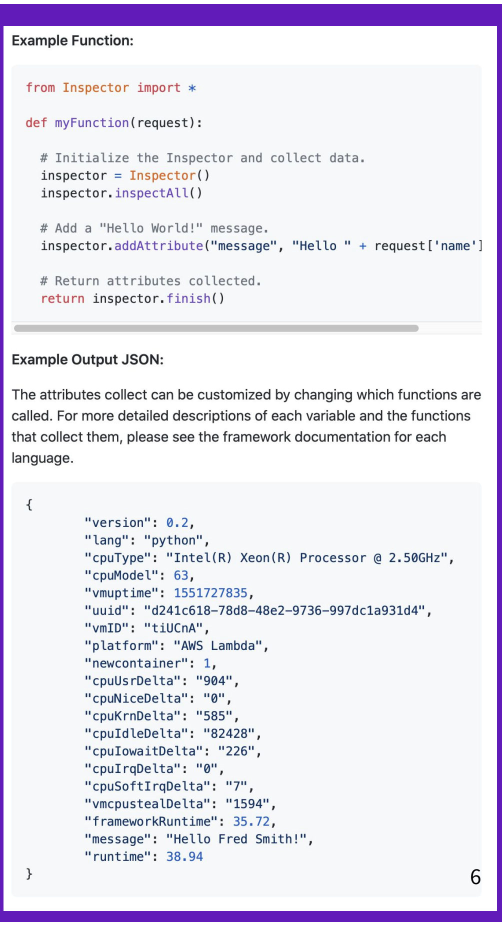


Example Hello World Functio

```
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.inspectAllDeltas()
  return inspector.finish()
```

SAAF Metrics and Design

- 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…



Example Function:

```
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 " + request['name']

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

Example Output JSON:

The attributes collect can be customized by changing which functions are called. For more detailed descriptions of each variable and the functions that collect them, please see the framework documentation for each language.

```
{
        "version": 0.2,
        "lang": "python",
        "cpuType": "Intel(R) Xeon(R) Processor @ 2.50GHz",
        "cpuModel": 63,
        "vmuptime": 1551727835,
        "uuid": "d241c618-78d8-48e2-9736-997dc1a931d4",
        "vmID": "tiUCnA",
        "platform": "AWS Lambda",
        "newcontainer": 1,
        "cpuUsrDelta": "904",
        "cpuNiceDelta": "0",
        "cpuKrnDelta": "585",
        "cpuIdleDelta": "82428",
        "cpuIowaitDelta": "226",
        "cpuIrqDelta": "0",
        "cpuSoftIrqDelta": "7",
        "vmcpustealDelta": "1594",
        "frameworkRuntime": 35.72,
        "message": "Hello Fred Smith!",
        "runtime": 38.94
}
```

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

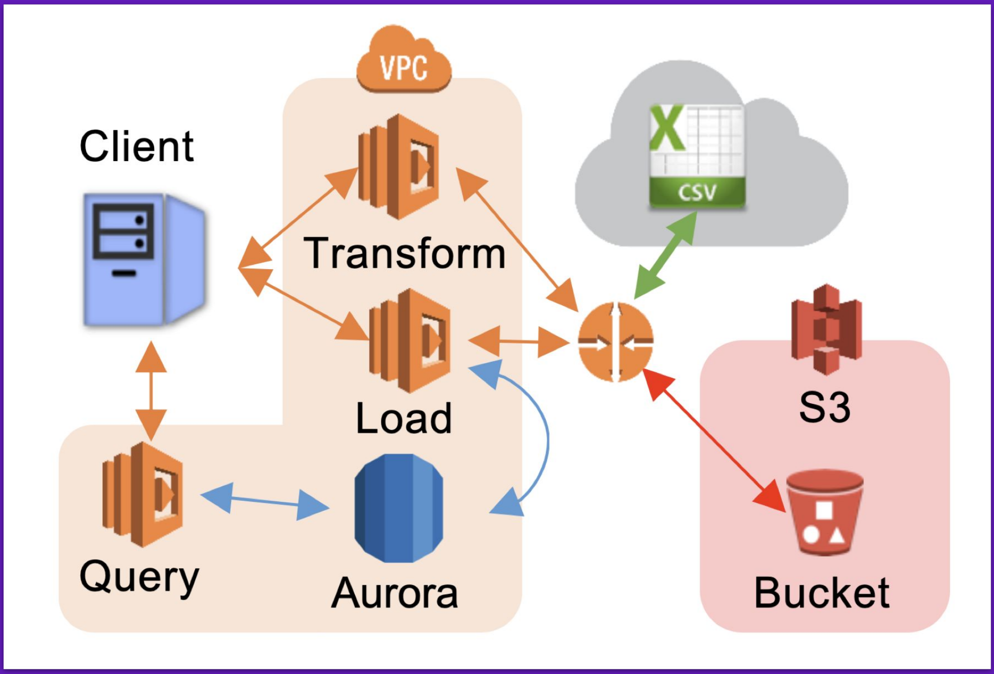
- Automatically compile results into a report

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

Best Performance: Query

Low Cold Latency

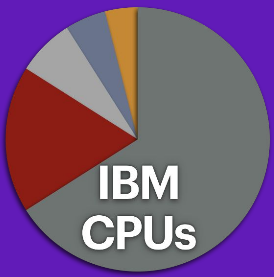
Low High-Tenancy Impact

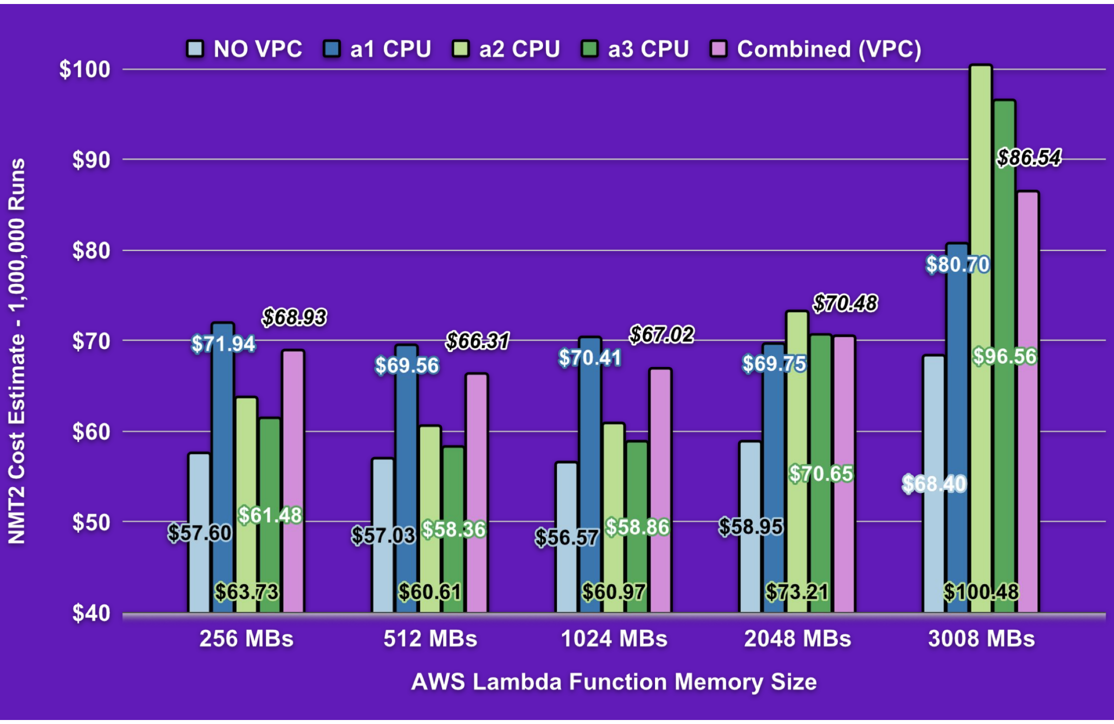
Low High-Tenancy Impact

Low Cold Latency

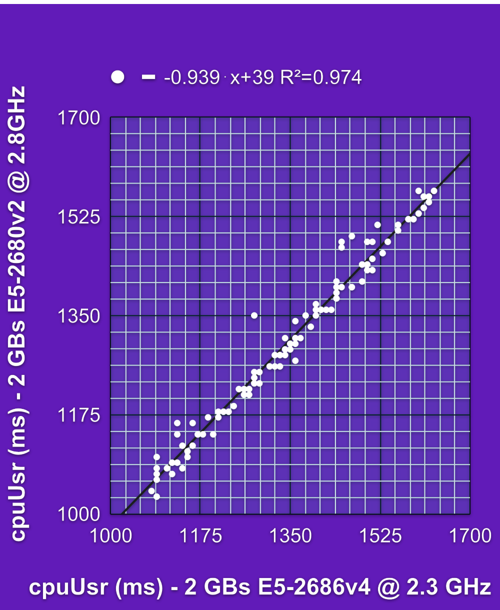
Good Memory Scaling

Research with SAAF: Predicting Performance





Predicting Performance Scenarios



CPU:

256 MBs **a1 → a2**
256 MBs **a1 → a3**
256 MBs **a2 → a3**
512 MBs **a1 → a2**
512 MBs **a1 → a3**
512 MBs **a2 → a3**
1024 MBs **a1 → a2**
1024 MBs **a1 → a3**
1024 MBs **a2 → a3**
2048 MBs **a1 → a2**
2048 MBs **a1 → a3**
2048 MBs **a2 → a3**

Memory:

a1 **256MBs → 512MBs**
a1 **256MBs → 1024MBs**
a1 **256MBs → 2048MBs**
a2 **256MBs → 512MBs**
a2 **256MBs → 1024MBs**
a2 **256MBs → 2048MBs**
a3 **256MBs → 512MBs**
a3 **256MBs → 1024MBs**
a3 **256MBs → 2048MBs**

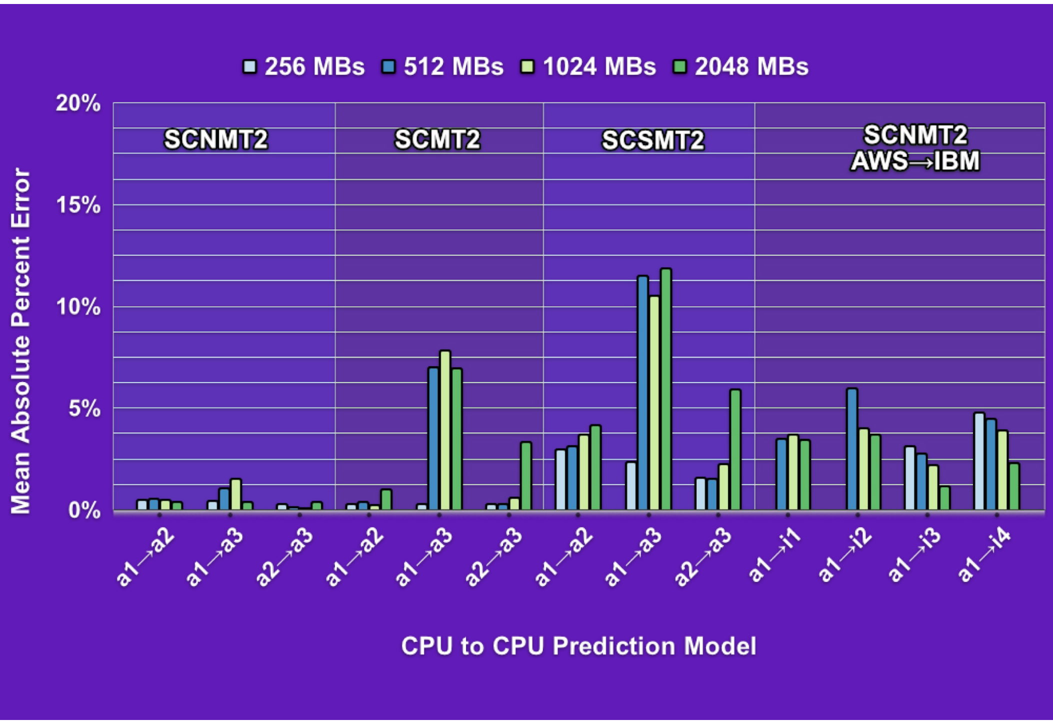
Platform:

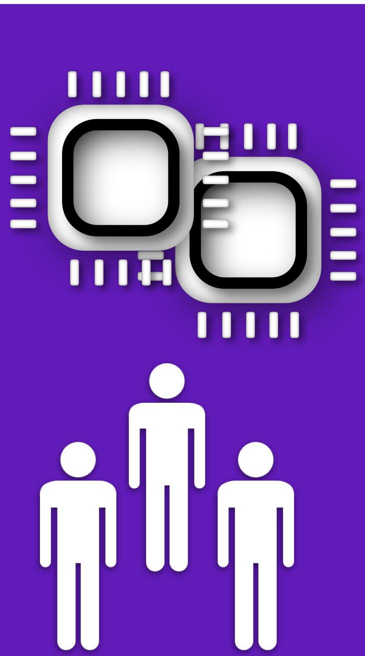
256MBs **a1 → i1**
256MBs **a1 → i2**
256MBs **a1 → i3**
256MBs **a1 → i4**
512MBs **a1 → i1**
512MBs **a1 → i2**
512MBs **a1 → i3**
512MBs **a1 → i4**
1024MBs **a1 → i1**
1024MBs **a1 → i2**
1024MBs **a1 → i3**
1024MBs **a1 → i4**
2048MBs **a1 → i1**
2048MBs **a1 → i2**
2048MBs **a1 → i3**
2048MBs **a1 → i4**

Prediction Scenarios

$$\text{Runtime} = \frac{(\text{cpuUsr} + \text{cpuKrn} + \text{cpuIdle} + \text{cpuIOWait} + \text{cpuIntSrvc} + \text{cpuSftIntSrvc})}{(\text{\# of cores})}$$

Predicting Performance Conclusions





Overall Conclusions

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

Thank You!

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.