Tutorial: Enhancing Observability of Serverless Computing with the Serverless Application Analytics Framework (SAAF)

Robert Cordingly, Navid Heydari, Hanfei Yu,
Varik Hoang, Zohreh Sadeghi, Wes Lloyd

School of Engineering and Technology
University of Washington Tacoma
12th ACM/SPEC International Conference on Performance Engineering **(ICPE 2021)**

SAAF Outline

- Introduction to Serverless Computing
    - Motivation
    - Delivery models and platforms
    - Advantages and challenges
- Serverless Application Analytics Framework (SAAF)
    - Design of SAAF, Supported Languages, Metrics
    - Tools: FaaS Runner, Publish Script
- Analysis Examples with SAAF
    - Programming language comparison, performance modeling
    - Scalability testing, Resource utilization profiling
    - Tracking infrastructure reuse
- Conclusions
- SAAF Demo

How should my app withstand a server **failing**?

How can I tell if a server has been **compromised**?

How can I increase **utilization** of my servers?

Which **OS** should my servers run?

How much remaining **capacity** do my servers have?

How should I implement dynamic **configuration changes** on my servers

When should I decide to **scale up** my servers?

What size servers are right for my budget?

How will I keep my server OS **patched**?

How can I control **access from** my servers?

Servers

(AAHHHHHHHHH!!)

Which packages should be baked into my **server images**?

How will new code be **deployed** to my servers?

How will the application handle server **hardware failure**?

How many users create **too much load** for my servers?

What size server is right for my **performance**?

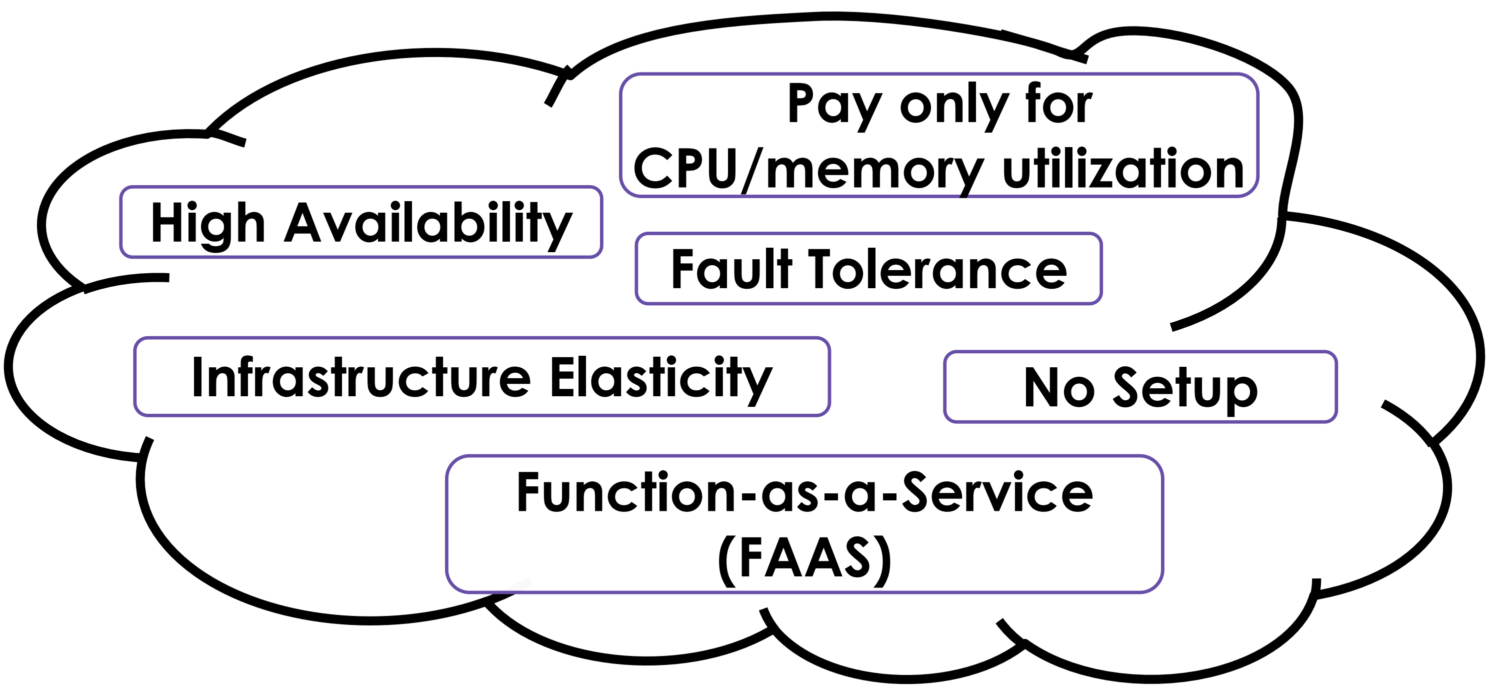
Which users should have **access to** my servers?

Should I **tune OS settings** to optimize my application?

When should I decide to **scale out** my servers?

How many servers should I budget for?

Serverless Computing



Serverless Computing

Why Serverless Computing?

Many features of distributed systems, that are challenging to deliver, are provided automatically

…they are built into the platform

SAAF Outline

- Introduction to Serverless Computing
  - Motivation
  - Delivery models and platforms
  - Advantages and challenges
- Serverless Application Analytics Framework (SAAF)
  - Design of SAAF, Supported Languages, Metrics
  - Tools: FaaS Runner, Publish Script
- Analysis Examples with SAAF
  - Programming language comparison, performance modeling
  - Scalability testing, Resource utilization profiling
  - Tracking infrastructure reuse
- Conclusions
- SAAF Demo

Serverless Computing Delivery Models

- Function-as-a-Service (FaaS)
- Container-as-a-Service (CaaS)
- Database-as-a-Service (DBaaS)

Commercial FaaS Platforms

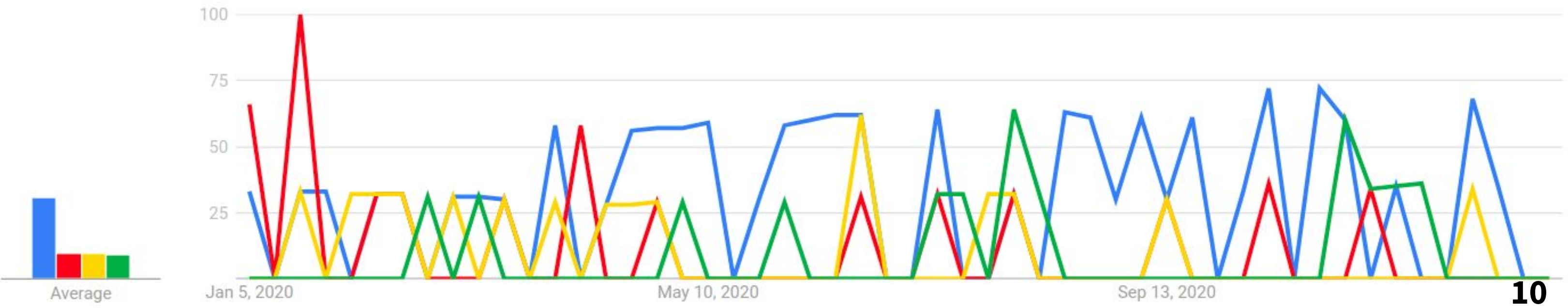
- AWS Lambda
- Azure Functions
- IBM Cloud Functions
- Google Cloud Functions

Open Source FaaS Platforms

- Apache OpenWhisk
- OpenFaaS
- Fn Project
- Kubeless

Google Search Trends: Open Source Faas Platform 2020

OpenFaaS: 31	Apache OpenWhisk: 10
Kubeless: 10	Fn Project: 9



FaaS Platform Example: AWS Lambda

Bring your own code

- Languages: Java, Python, Node.js, Go, C#, Ruby, PowerShell, Bash
- Bring you own libraries

Flexible use

- Synchronous or asynchronous
- Integration w/ other AWS services

Simple Resource Model

- Function memory 128 MB to 10 GB
- CPU timeshare and network bandwidth scaled proportional to memory

Flexible Authorization

- Grant access to resources and VPCs
- Fine-grained control for invoking functions

SAAF Outline

- Introduction to Serverless Computing
  - Motivation
  - Delivery models and platforms
  - Advantages and challenges
- Serverless Application Analytics Framework (SAAF)
  - Design of SAAF, Supported Languages, Metrics
  - Tools: FaaS Runner, Publish Script
- Analysis Examples with SAAF
  - Programming language comparison, performance modeling
  - Scalability testing, Resource utilization profiling
  - Tracking infrastructure reuse
- Conclusions
- SAAF Demo

Function-as-a-Service

Advantages

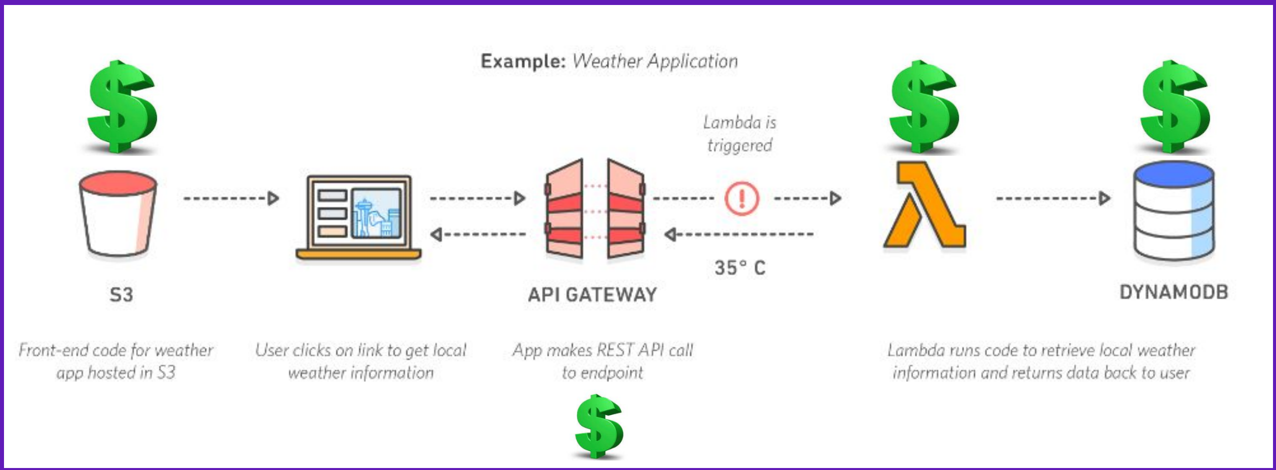
- No management of servers
- Pay only for actual compute time
- High availability (24/7)
- Scalability (elastic resources)
- Fault tolerance
- Rapid deployment/updates
- Supports vendor workload consolidation

Challenges

- Limited observability of servers
- Multi-dimensional pricing policies
- Vendor lock-In
- Heterogeneous infrastructure
- Performance variation
- Memory reservation size
- Infrastructure freeze/thaw
- Function composition
- Pricing obfuscation

Vendor Architectural Lock-In

Cloud native (FaaS) software architecture requires external services/components



Increased dependencies → increased hosting costs

Pricing Obfuscation

<u>VM pricing:</u> hourly pricing policy, billed to the nearest second is intuitive to understand…

<u>FaaS pricing:</u>

<u>AWS Lambda Pricing</u>

<u>FREE TIER:</u> first 1,000,000 function calls/month □ FREE
first 400,000 GB-sec/month □ FREE

Afterwards: $0.0000002 per request
$0.000000208 to rent 128MB / 100-ms *-or-*
$0.00001667 to rent 1GB / 1-sec

IaaS Cloud Pricing Policies

Virtual machines as-a-service at ¢ per hour

No premium to scale:

```
      1000 computers   @     1 hour
  =      1 computer    @  1000 hours
```

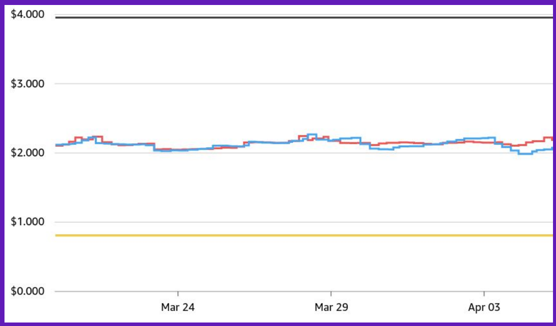
Illusion of infinite scalability to cloud user

As many computers as you can afford

Pricing policies are becoming increasingly granular

- By the minute, second

Preemptive/Spot reduced price instances □



Price Comparison: IaaS vs. FaaS

Assume 1 month = 30.41667 days (365d / 12)

Workload

Continuous 1-sec service calls: 100% of 2 CPU-cores
100% of 4GB of memory

Workload: 2 continuous client threads

Duration: 1 month (30.41667 days)

ON AWS EC2: Amazon EC2 c5.large 2-vCPU VM x 4GB

c5.large: 8.5¢/hour, 24 hrs/day x 30.41667 days

Hosting cost: $62.05/month

Price Comparison: IaaS vs. FaaS - II

Assume 1 month = 30.41667 days (365d / 12)

Workload

Continuous 1-sec service calls: 100% of 2 CPU-cores
100% of 4GB of memory

Workload: 2 continuous client threads

Duration: 1 month (30.41667 days)

ON AWS Lambda: 2,628,000 function calls, 1-sec @ 4GB

function calls: 2.628 million x 20¢/million

runtime: 2,628,000 sec x 4 GB

Hosting cost: ???

Price Comparison: IaaS vs FaaS - III

Workload:	(4GB)	10,512,000 GB-sec	
FREE:	-	400,000 GB-sec	
Runtime/Memory:		10,112,000 GB-sec	
Charge:		x .00001667 GB-sec	**$168.57**
Invocations:		2,628,000 calls	
FREE:	-	1,000,000 calls	
Charge:		1,628,000 calls	**$.33**
Total:			**$168.90** **(272.2%)**

BREAK-EVEN POINT = $62.05 (VM) - $0.33 (calls) = $61.72

$61.72 / .00001667 GB-sec = ~3,702,459 GB-sec-month / 4GB/call = ~925,614 sec
~10.71 days

Point at which using FaaS costs the same as IaaS

Price Comparison: IaaS vs FaaS - III

Workload: (4GB) 10,512,000 GB-sec

FREE: - 400,000 GB-sec

Worst-case scenario= ~2.72x !

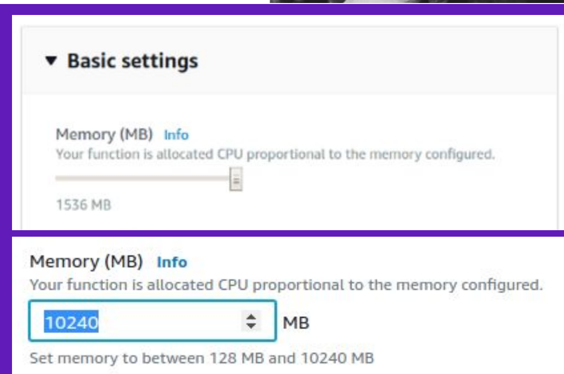
AWS EC2:	**$62.05**
AWS Lambda:	**$168.90**

~10.71 days

Point at which using FaaS costs the same as IaaS

Memory Reservation

Lambda memory reserved for functions



UI provides textbox to set function's memory *(previously a slidebar)*

Resource capacity (CPU, disk, network) scaled relative to memory

*"every **doubling** of memory, **doubles** CPU…"*

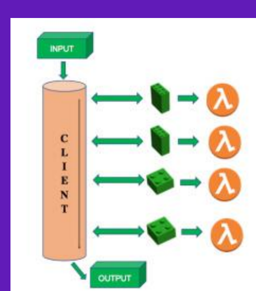
But how much memory do functions require?

Service Composition

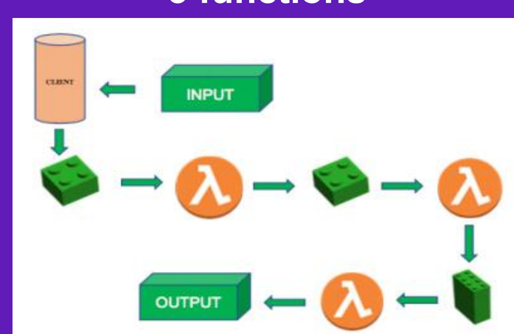
How should applications be decomposed into serverless functions for deployment?

Monolithic Deployment

Client flow control, 4 functions



Server-side flow control, 3 functions



<u>Recommended practice:</u>

Decompose into many microservices

Platform limits: code + libraries ~250MB (uncompressed)

How does composition impact the number of function invocations, and memory utilization?

Infrastructure Freeze/Thaw Cycle

Unused infrastructure is deprecated

- ***But after how long?***

FaaS Infrastructure known as "function instances"

Implemented using VMs/microVMs, containers, or software-based isolates

STATES:

Physical Host-COLD

- Function code not yet transferred to any server

Container-COLD

- Function code transferred to server, but infrastructure not created

Container-WARM

- Active container running

SAAF Outline

λ

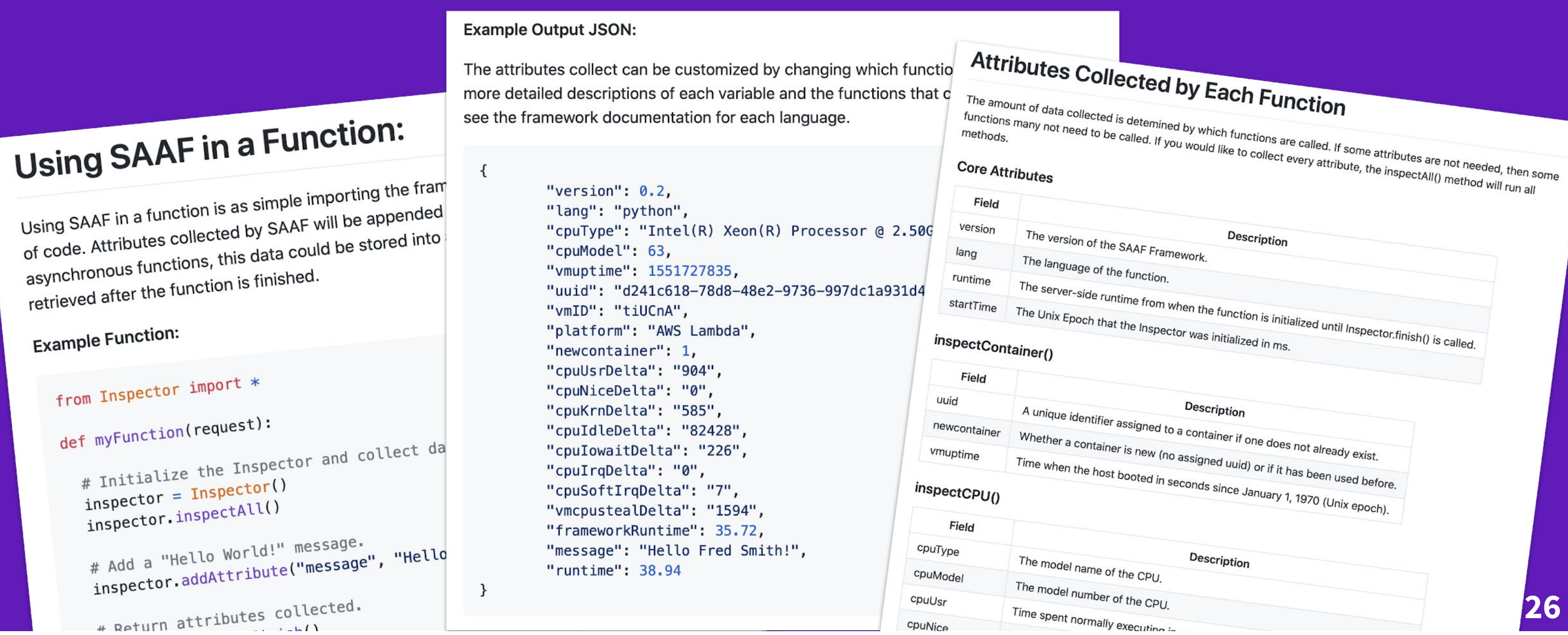
- Introduction to Serverless Computing
  - Motivation
  - Delivery models and platforms
  - Advantages and challenges
- Serverless Application Analytics Framework (SAAF)
  - Design of SAAF, Supported Languages, Metrics
  - Tools: FaaS Runner, Publish Script
- Analysis Examples with SAAF
  - Programming language comparison, performance modeling
  - Scalability testing, Resource utilization profiling
  - Tracking infrastructure reuse
- Conclusions
- SAAF Demo

SAAF for FAAS

To address these challenges better FaaS profiling & measurement tools are required:

- Limited observability of servers
- Multi-dimensional pricing policies
- Vendor lock-In
- Heterogeneous infrastructure
- Performance variation
- Memory reservation size
- Infrastructure freeze/thaw
- Function composition
- Pricing obfuscation

SAAF: The Serverless Application Analytics Framework



SAAF Outline

- Introduction to Serverless Computing
  - Motivation
  - Delivery models and platforms
  - Advantages and challenges
- Serverless Application Analytics Framework (SAAF)
  - Design of SAAF, Supported Languages, Metrics
  - Tools: FaaS Runner, Publish Script
- Analysis Examples with SAAF
  - Programming language comparison, performance modeling
  - Scalability testing, Resource utilization profiling
  - Tracking infrastructure reuse
- Conclusions
- SAAF Demo

Supported Platforms and Languages

SAAF Metrics and Design

- Profiles **48** distinct metrics (CPU, memory, I/O utilization), monitors infrastructure state, and observes platform scalability
- Data collection is directed by calling profiling 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

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:

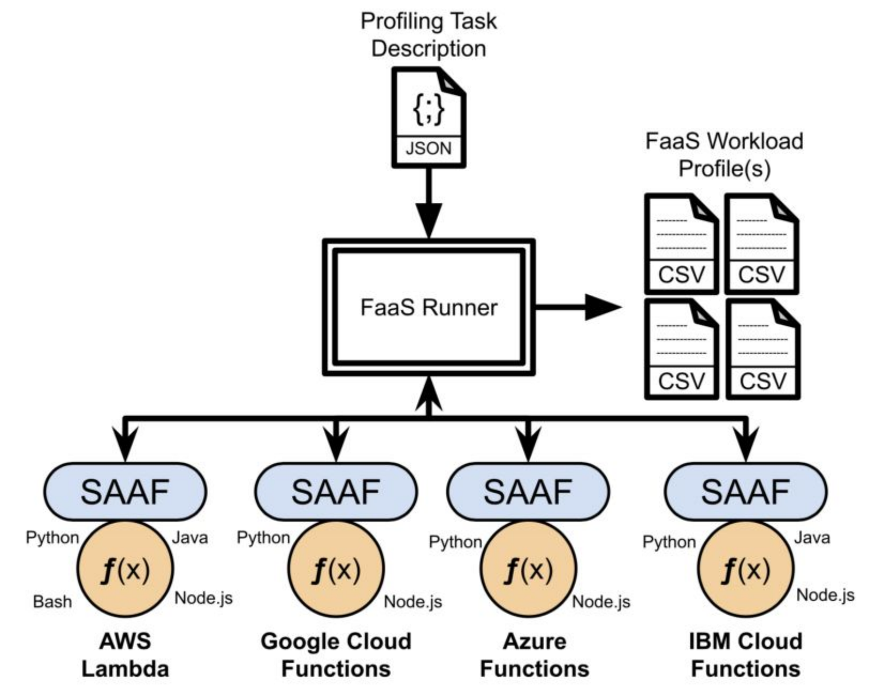
The attributes collect can be customized by changing which functions are called. For more detailed descriptions of each variable and the functions that collect them, please see the framework documentation for each language.


```
{
        "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 Outline

- Introduction to Serverless Computing
  - Motivation
  - Delivery models and platforms
  - Advantages and challenges
- Serverless Application Analytics Framework (SAAF)
  - Design of SAAF, Supported Languages, Metrics
  - Tools: FaaS Runner, Publish Script
- Analysis Examples with SAAF
  - Programming language comparison, performance modeling
  - Scalability testing, Resource utilization profiling
  - Tracking infrastructure reuse
- Conclusions
- SAAF Demo

Workload Profiling with SAAF and FaaS Runner



SAAF Tools: FaaS Runner

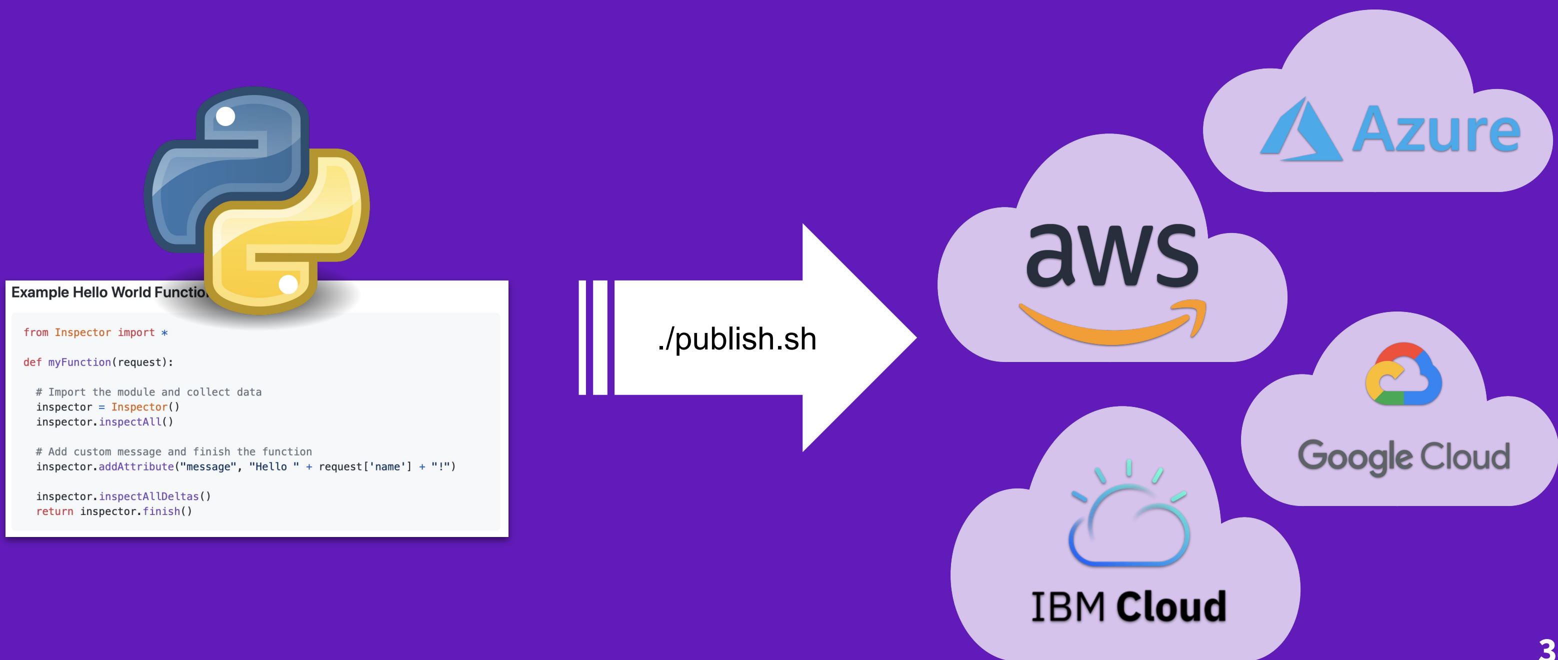
- Client for running experiments
- Executes reproducible tests defined by files or command line arguments
  - Automatically change memory settings or redeploy functions
  - Run functions sequentially or concurrently with many threads
  - Run functions synchronously or asynchronously
  - Define payload 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

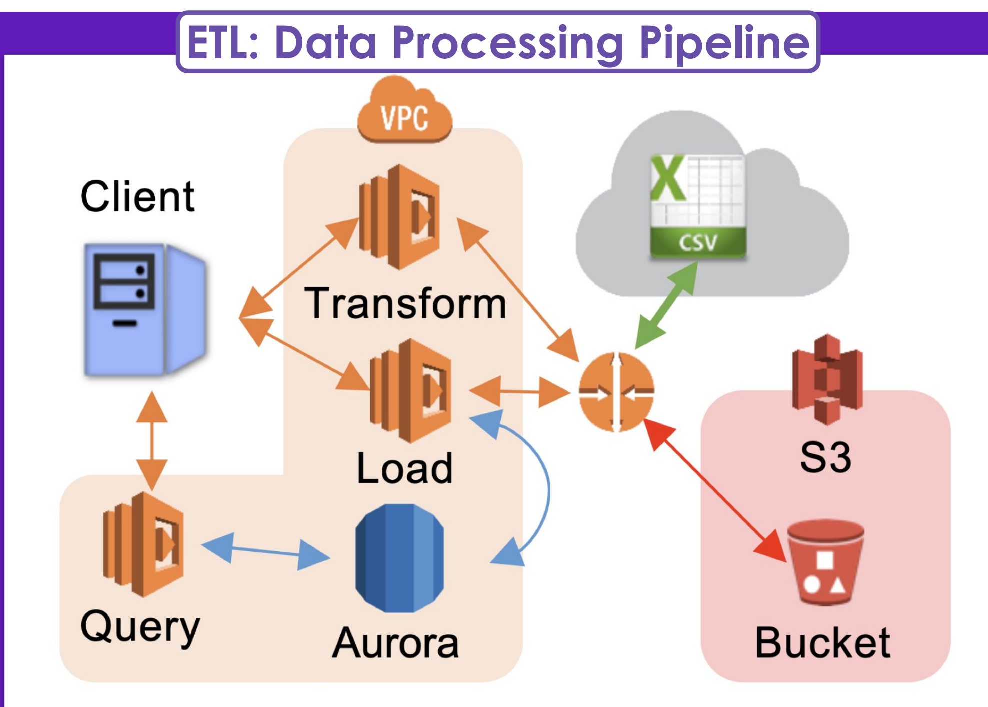
SAAF Tools: Publish Script



SAAF Outline

- Introduction to Serverless Computing
  - Motivation
  - Delivery models and platforms
  - Advantages and challenges
- Serverless Application Analytics Framework (SAAF)
  - Design of SAAF, Supported Languages, Metrics
  - Tools: FaaS Runner, Publish Script
- Analysis Examples with SAAF
  - Programming language comparison, performance modeling
  - Scalability testing, Resource utilization profiling
  - Tracking infrastructure reuse
- Conclusions
- SAAF Demo

SAAF: FaaS Programming Languages Comparison



FaaS Programming Languages Comparison - II

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

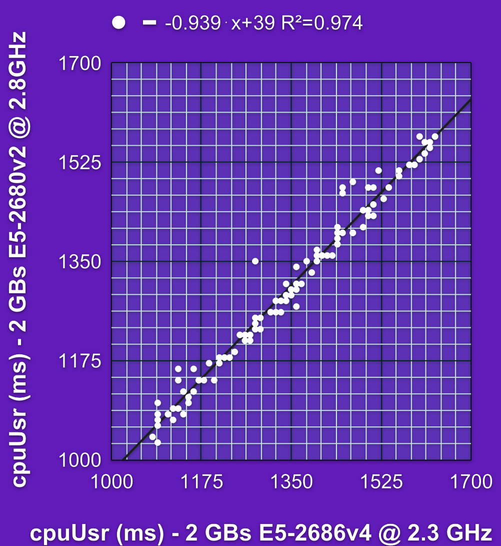
Best Performance: Transform
Best Performance: Query
Low Cold Latency
Low High-Tenancy Impact

Low High-Tenancy Impact
Low Cold Latency
Good Memory Scaling

SAAF Outline

- Introduction to Serverless Computing
  - Motivation
  - Delivery models and platforms
  - Advantages and challenges
- Serverless Application Analytics Framework (SAAF)
  - Design of SAAF, Supported Languages, Metrics
  - Tools: FaaS Runner, Publish Script
- Analysis Examples with SAAF
  - Programming language comparison, performance modeling
  - Scalability testing, Resource utilization profiling
  - Tracking infrastructure reuse
- Conclusions
- SAAF Demo

Runtime Prediction Scenarios

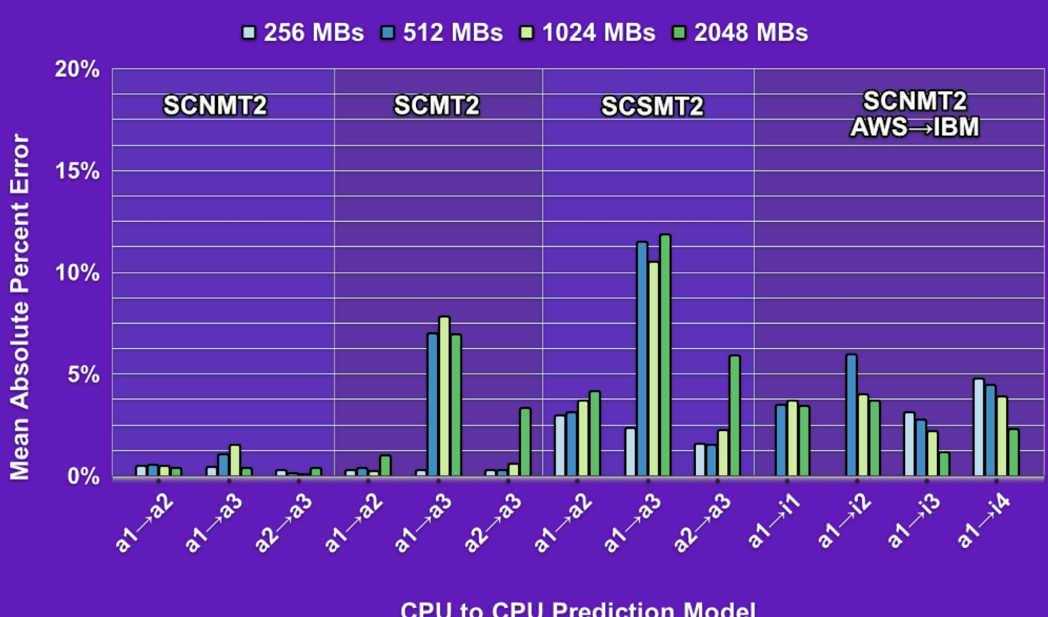


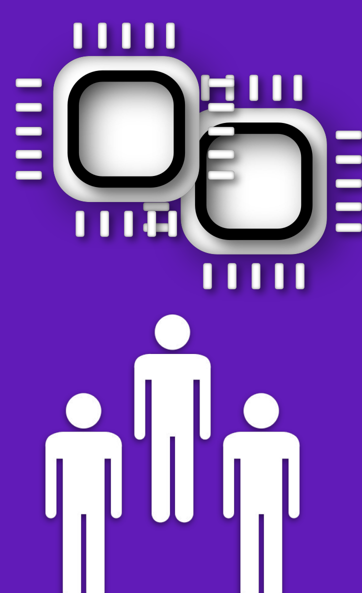
CPU:	Memory:	Platform:	
256 MBs **a1** → **a2**	a1 **256MBs** → **512MBs**	256MBs **a1** → **i1**	1024MBs **a1** → **i1**
256 MBs **a1** → **a3**	a1 **256MBs** → **1024MBs**	256MBs **a1** → **i2**	1024MBs **a1** → **i2**
256 MBs **a2** → **a3**	a1 **256MBs** → **2048MBs**	256MBs **a1** → **i3**	1024MBs **a1** → **i3**
512 MBs **a1** → **a2**	a2 **256MBs** → **512MBs**	256MBs **a1** → **i4**	1024MBs **a1** → **i4**
512 MBs **a1** → **a3**	a2 **256MBs** → **1024MBs**	512MBs **a1** → **i1**	2048MBs **a1** → **i1**
512 MBs **a2** → **a3**	a2 **256MBs** → **2048MBs**	512MBs **a1** → **i2**	2048MBs **a1** → **i2**
1024 MBs **a1** → **a2**	a3 **256MBs** → **512MBs**	512MBs **a1** → **i3**	2048MBs **a1** → **i3**
1024 MBs **a1** → **a3**	a3 **256MBs** → **1024MBs**	512MBs **a1** → **i4**	2048MBs **a1** → **i4**
1024 MBs **a2** → **a3**	a3 **256MBs** → **2048MBs**		
2048 MBs **a1** → **a2**			
2048 MBs **a1** → **a3**			
2048 MBs **a2** → **a3**			

Prediction Scenarios

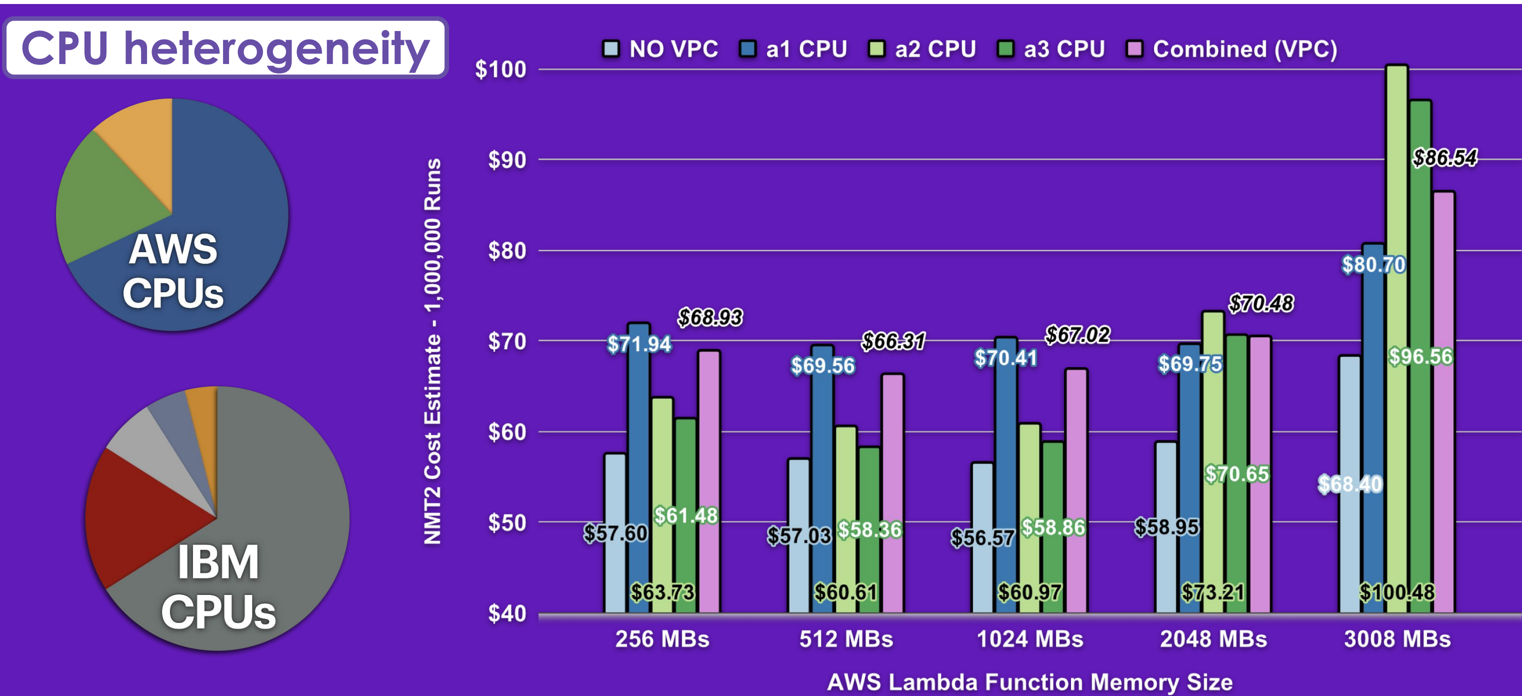
$$\text{Runtime} = \frac{(\text{cpuUsr} + \text{cpuKrn} + \text{cpuIdle} + \text{cpuIOWait} + \text{cpuIntSrvc} + \text{cpuSftIntSrvc})}{(\#\text{ of cores})}$$

Runtime Prediction - Mean Absolute Percentage Error





SAAF: Predicting Hosting Costs



SAAF Outline

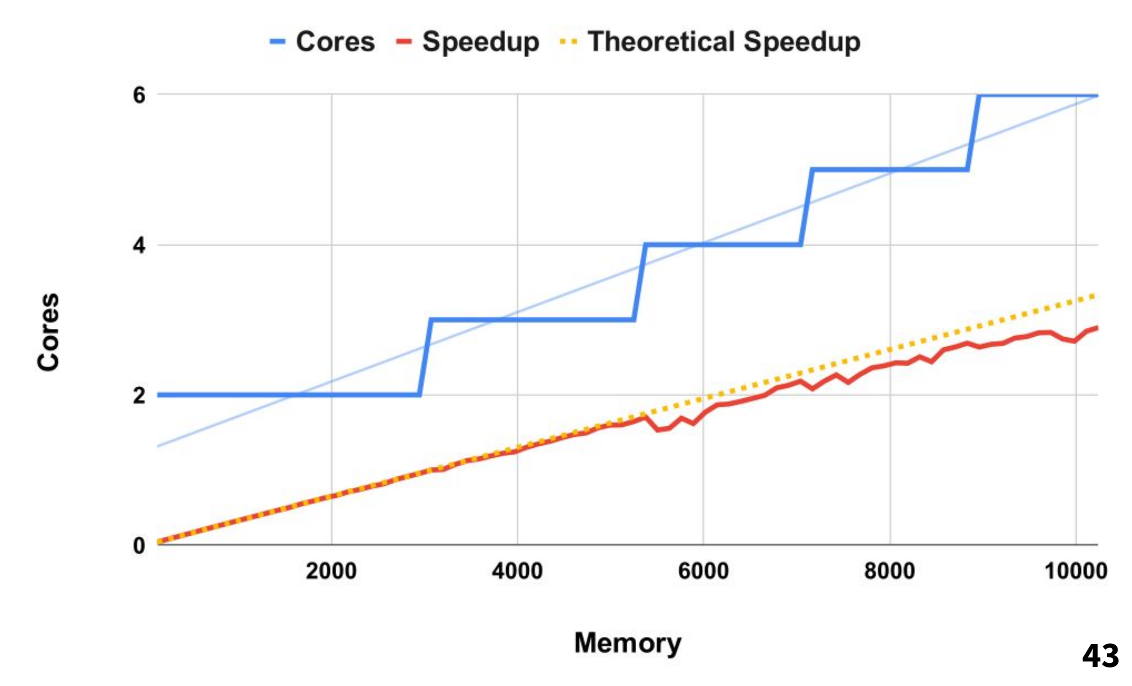
- Introduction to Serverless Computing
  - Motivation
  - Delivery models and platforms
  - Advantages and challenges
- Serverless Application Analytics Framework (SAAF)
  - Design of SAAF, Supported Languages, Metrics
  - Tools: FaaS Runner, Publish Script
- Analysis Examples with SAAF
  - Programming language comparison, performance modeling
  - Scalability testing, Resource utilization profiling
  - Tracking infrastructure reuse
- Conclusions
- SAAF Demo

AWS Lambda - Scalable Performance Test

Scalability from:
0 to 10 GB

sysbench: prime number generation w/ 12 threads

Doubling memory doubles performance until adding the 4th vCPU:



SAAF Outline

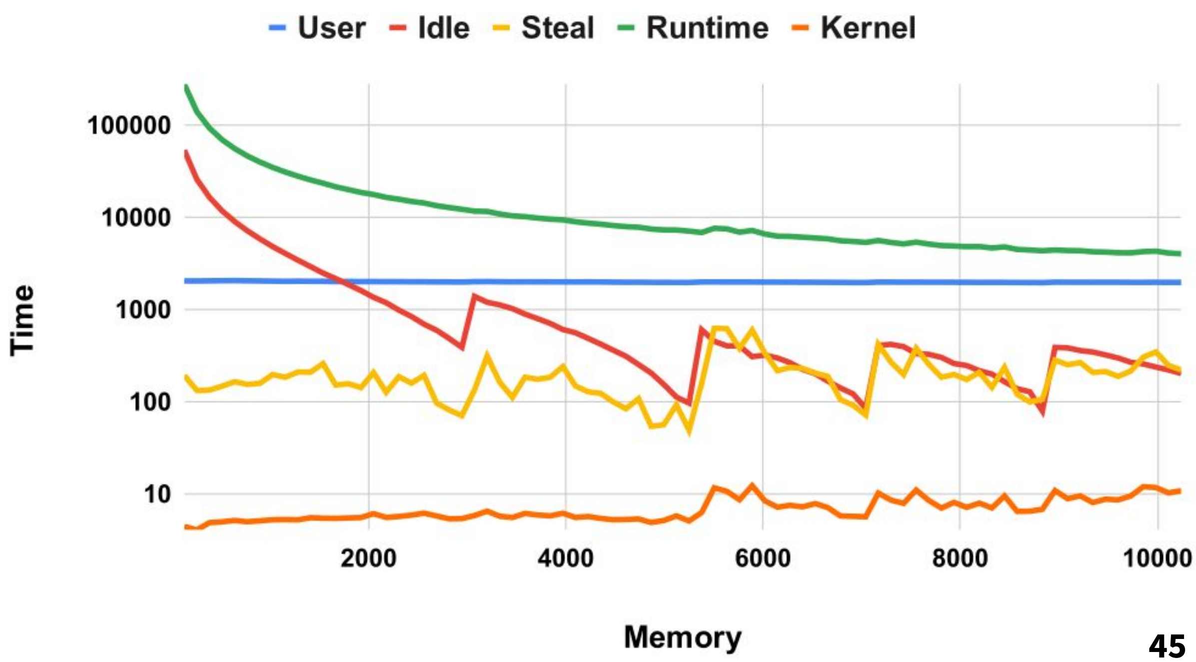
- Introduction to Serverless Computing
  - Motivation
  - Delivery models and platforms
  - Advantages and challenges
- Serverless Application Analytics Framework (SAAF)
  - Design of SAAF, Supported Languages, Metrics
  - Tools: FaaS Runner, Publish Script
- Analysis Examples with SAAF
  - Programming language comparison, performance modeling
  - Scalability testing, Resource utilization profiling
  - Tracking infrastructure reuse
- Conclusions
- SAAF Demo

AWS Lambda - Linux CPU time Accounting Metrics

sysbench: prime number generation with 12 threads

from 0 to 10 GB

log scale



SAAF Outline

- Introduction to Serverless Computing
  - Motivation
  - Delivery models and platforms
  - Advantages and challenges
- Serverless Application Analytics Framework (SAAF)
  - Design of SAAF, Supported Languages, Metrics
  - Tools: FaaS Runner, Publish Script
- Analysis Examples with SAAF
  - Programming language comparison, performance modeling
  - Scalability testing, Resource utilization profiling
  - Tracking infrastructure reuse
- Conclusions
- SAAF Demo

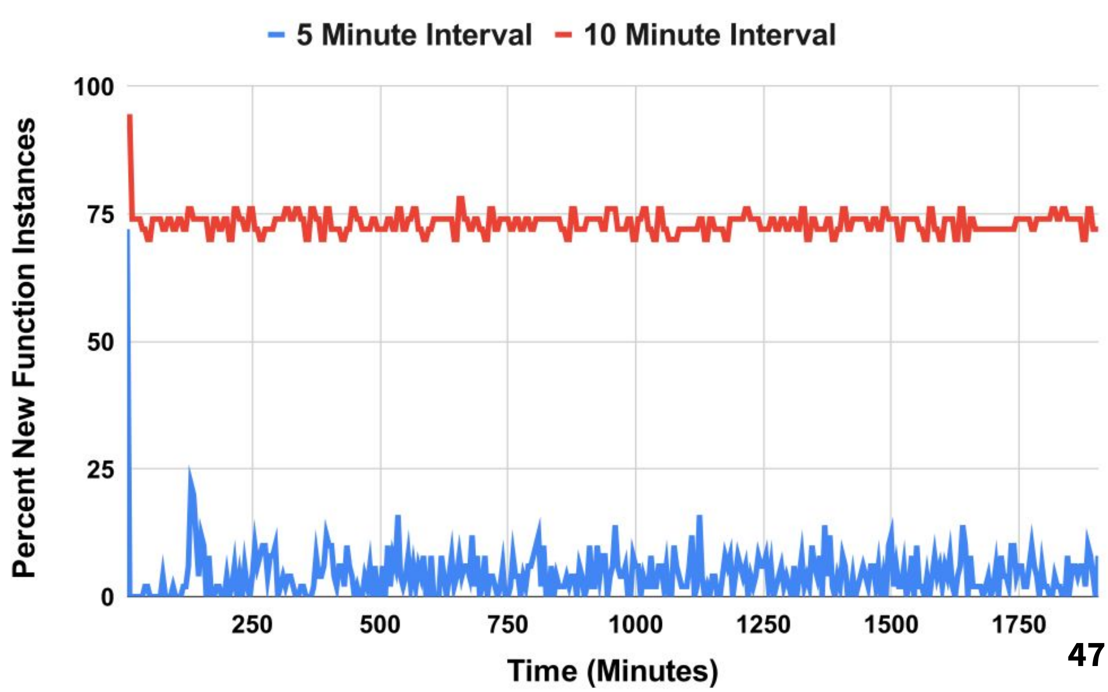
AWS Lambda - Infrastructure Reuse Testing

COLD infrastructure is common with the serverless freeze-thaw life cycle

Experiment:
50 concurrent calls

5-min vs 10-min delay

Evaluate % function instances



SAAF Outline

- Introduction to Serverless Computing
  - Motivation
  - Delivery models and platforms
  - Advantages and challenges
- Serverless Application Analytics Framework (SAAF)
  - Design of SAAF, Supported Languages, Metrics
  - Tools: FaaS Runner, Publish Script
- Analysis Examples with SAAF
  - Programming language comparison, performance modeling
  - Scalability testing, Resource utilization profiling
  - Tracking infrastructure reuse
- Conclusions
- SAAF Demo

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

SAAF Outline

- Introduction to Serverless Computing
  - Motivation
  - Delivery models and platforms
  - Advantages and challenges
- Serverless Application Analytics Framework (SAAF)
  - Design of SAAF, Supported Languages, Metrics
  - Tools: FaaS Runner, Publish Script
- Analysis Examples with SAAF
  - Programming language comparison, performance modeling
  - Scalability testing, Resource utilization profiling
  - Tracking infrastructure reuse
- Conclusions
- SAAF Demo

SAAF Demonstration

- SAAF Overview
- Writing a Function with SAAF
- Deploying Functions to all Platforms
- Running Experiments with FaaS Runner
- Generating Reports
- Working with Results in R
- Interactive FaaS with Jupyter

Thank You!

Questions or comments?
Please email: rcording@uw.edu or wlloyd@uw.edu

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

SAAF Online Tutorial:
https://github.com/wlloyduw/SAAF/blob/master/tutorial

Paper Link:
http://faculty.washington.edu/wlloyd/papers/ICPE_SAAF_proof.pdf

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

text