



# Implications of Programming Language Selection for Serverless Data Processing Pipelines

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

August 17-24, 2020

School of Engineering and Technology  
University of Washington Tacoma  
CBDCom 2020: IEEE International Conference on Cloud and Big Data

1

## Outline

- ➔ Background and Motivation
  - Research Questions
  - Serverless Application Analytics Framework (SAAF)
  - TLQ Pipeline and Static Code Analysis
  - Experiments and Results
  - Conclusions

2

aws



Google Cloud

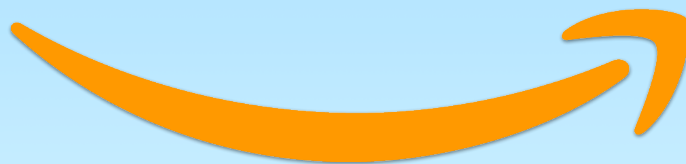


Azure



IBM Cloud

aws



# Serverless: Function-as-a-Service

- Developers create small applications called micro-services in a selection of supported languages by the cloud provider.
- Cloud providers automatically scale and manage cloud infrastructure instead of developers.

## The cost of FaaS:



- (Function Runtime) x (Memory Setting) x (Price)
- Billed only for runtime used.

### Basic information

#### Function name

Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

#### Runtime [Info](#)

Choose the language to use to write your function.

Go 1.x ▲

Latest supported

.NET Core 3.1 (C#/PowerShell)

Go 1.x

Java 11

Node.js 12.x

Python 3.8

Ruby 2.7

Other supported

.NET Core 2.1 (C#/PowerShell)

Java 8

Node.js 10.x

Python 2.7

Cancel

Create function

# Outline

- Background and Motivation
- ➔ Research Questions
  - Serverless Application Analytics Framework (SAAF)
  - TLQ Pipeline and Static Code Analysis
  - Experiments and Results
  - Conclusions

7

## Research Questions

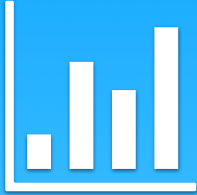


**RQ-1:** (Performance) How does the choice of programming language (Java, Go, Python, Node.js) impact the overall performance and throughput of a serverless data processing pipeline?

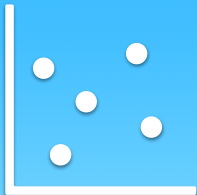
8



# Research Questions

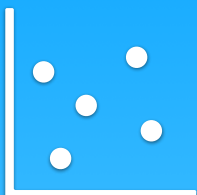


**RQ-1:** (Performance) How does the choice of programming language (Java, Go, Python, Node.js) impact the overall performance and throughput of a serverless data processing pipeline?



**RQ-2:** (Scalability) How does programming language choice impact the scalability of a serverless data processing pipeline when processing many concurrent data payloads?

# Research Questions



**RQ-2:** (Scalability) How does programming language choice impact the scalability of a serverless data processing pipeline when processing many concurrent data payloads?

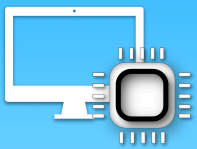


**RQ-3:** (Infrastructure State) How does the choice of programming language impact cold FaaS performance compared to warm FaaS performance for a data processing pipeline?

# Research Questions



**RQ-3:** (Infrastructure State) How does the choice of programming language impact cold FaaS performance compared to warm FaaS performance for a data processing pipeline?



**RQ-4:** (Memory/Cost) How does performance vary for a serverless data processing pipeline across alternate memory settings for implementations in different languages.

11

# Outline

- Background and Motivation
- Research Questions
- ➔ Serverless Application Analytics Framework (SAAF)
- TLQ Pipeline and Static Code Analysis
- Experiments and Results
- Conclusions

12

# Serverless Application Analytics Framework (SAAF)

## Using SAAF in a Function:

Using SAAF in a function is as simple importing the framework into your code. Attributes collected by SAAF will be appended to the function's return value. If you are using asynchronous functions, this data could be stored into a database and retrieved after the function is finished.

### 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 World!")  
  
    # Return attributes collected.  
    return inspector.attributes
```

### Example Output JSON:

The attributes collected can be customized by changing which functions are called. For more detailed descriptions of each variable and the functions that collect them, 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",  
  "vmcpusteatlDelta": "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.

### 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 user code.
cpuNice	Time spent normally executing nice code.

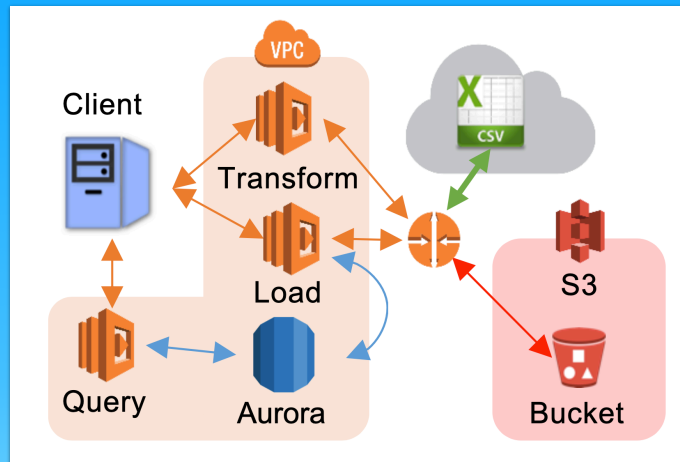
13

## Outline

- Background and Motivation
- Research Questions
- Serverless Application Analytics Framework (SAAF)
- ➔ TLQ Pipeline and Static Code Analysis
- Experiments and Results
- Conclusions

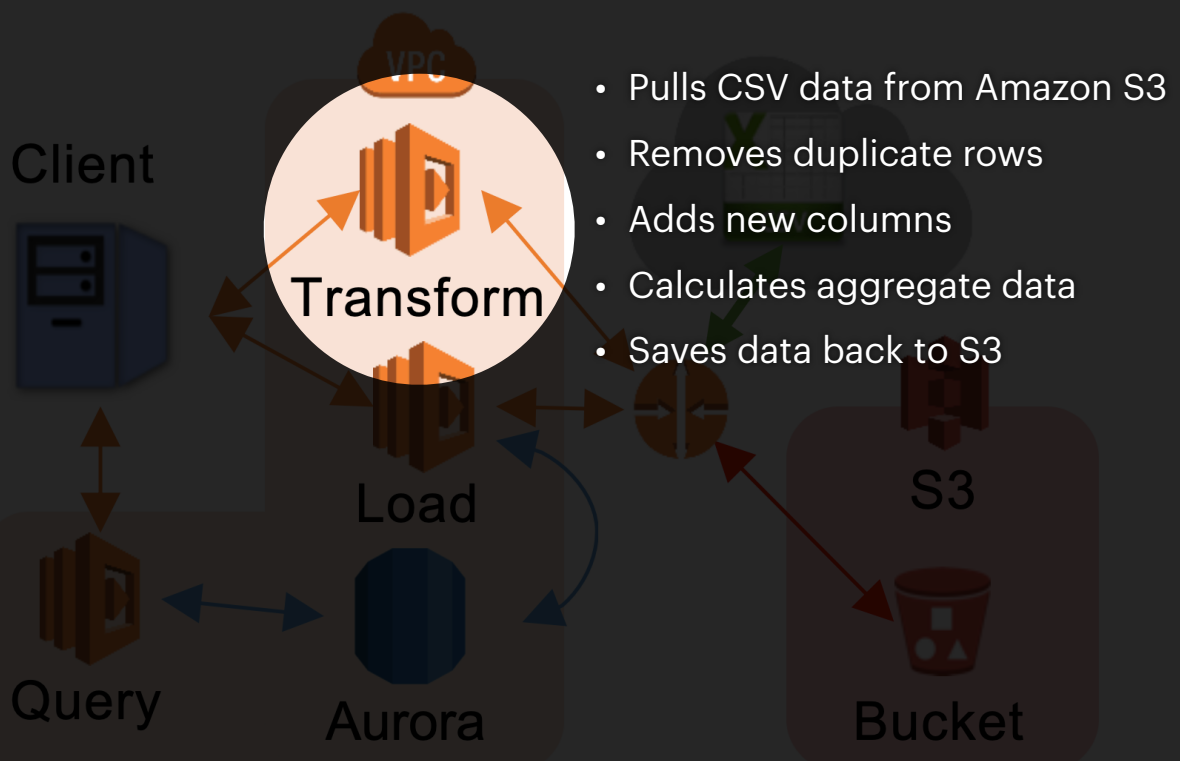
14

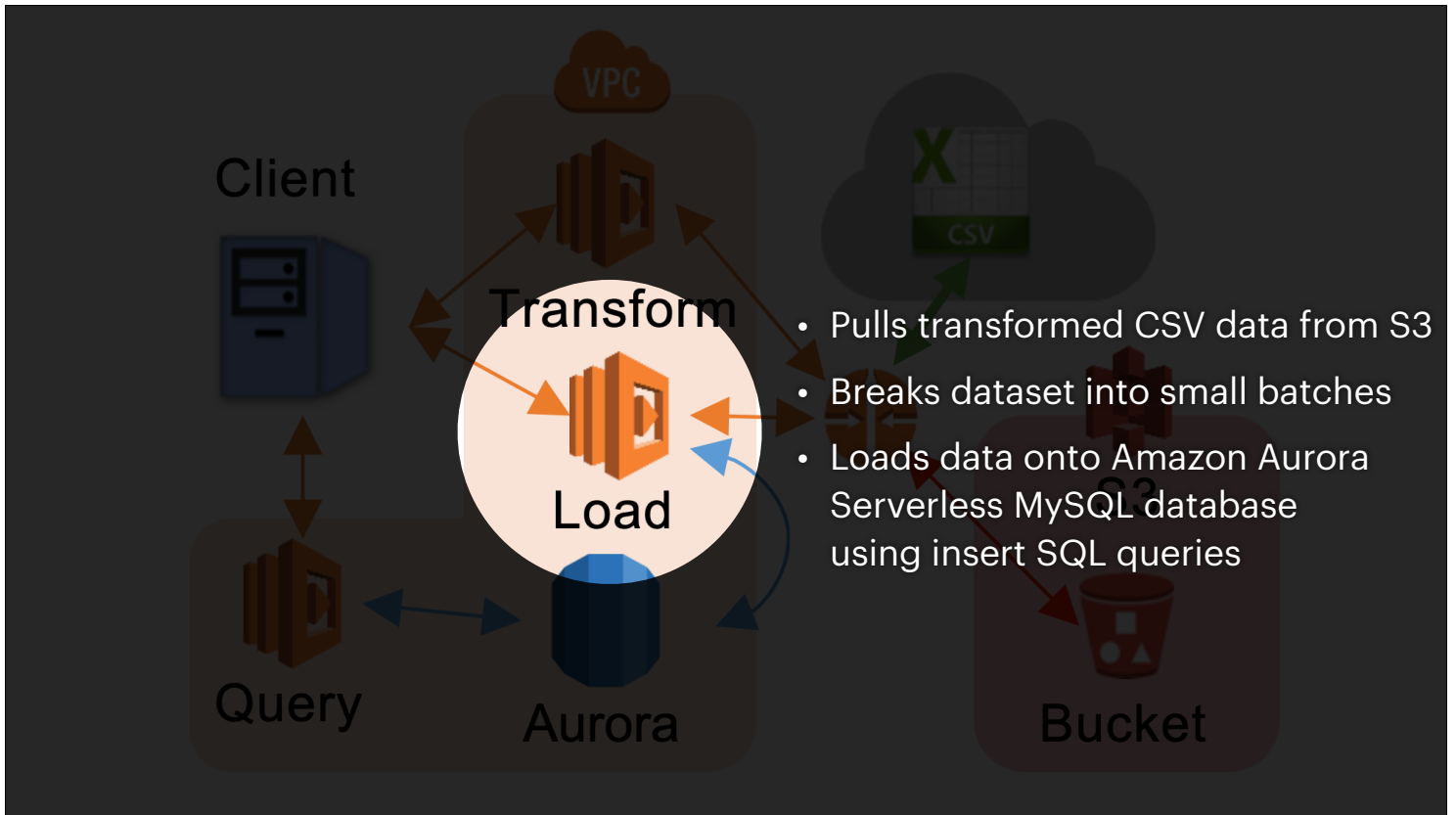
# Transform-Load-Query Pipeline

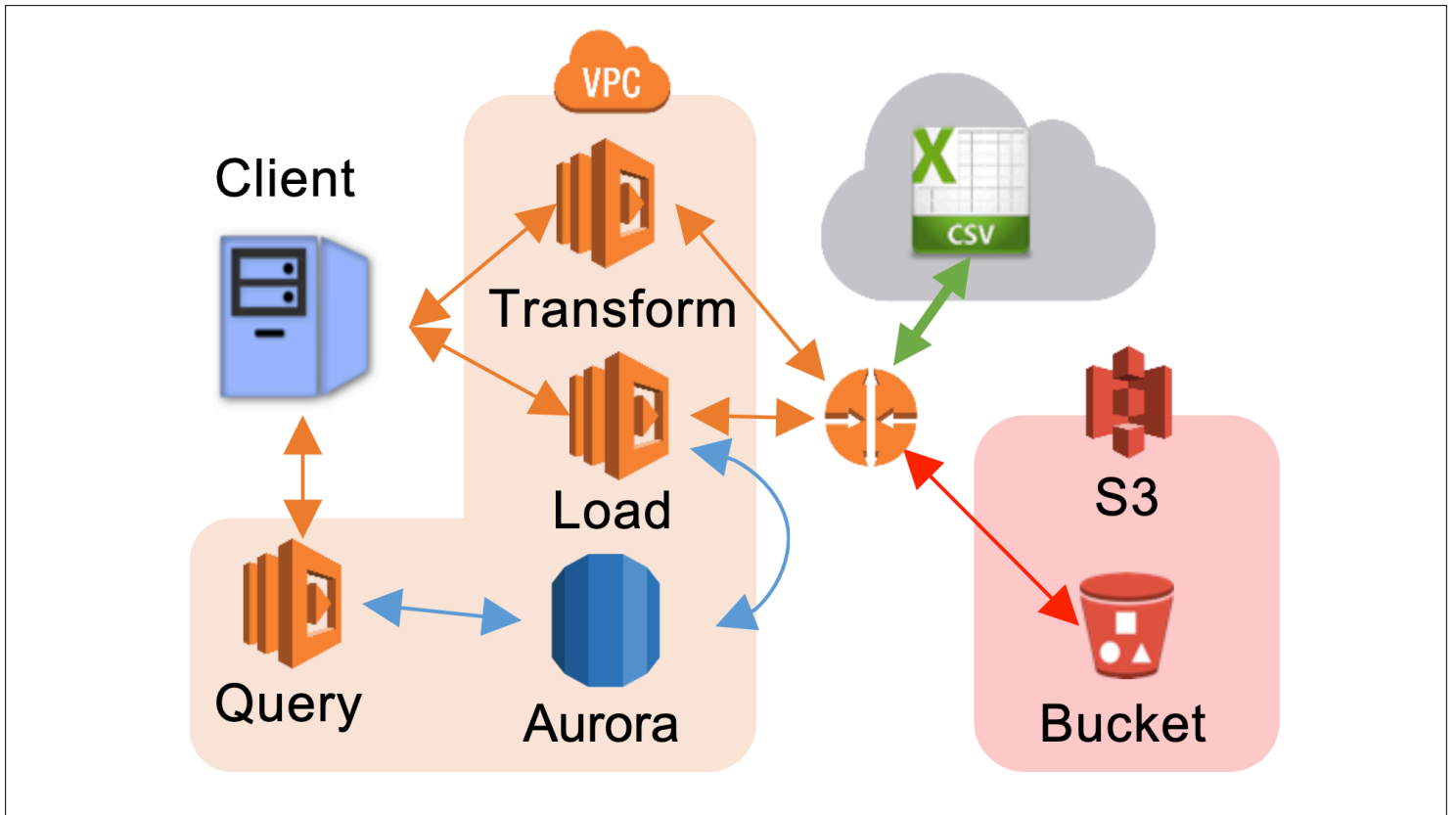


We developed a three-function data processing pipeline creating functionally identical versions in Java, Go, Node.js, and Python.

15







# Static Code Analysis

Service	Lang	Funcs	Vars	SLOC	Loops	Cloud Service Usage
Transform	Java	3	40	86	2	S3 Get/Put
Transform	Python	3	28	64	3	S3 Get/Put
Transform	Go	3	30	77	1	S3 Get/Put
Transform	Node.js	3	24	96	1	S3 Get/Put
Load	Java	3	25	77	2	S3 Get, DB Conn x1
Load	Python	3	21	57	3	S3 Get, DB Conn x1
Load	Go	3	15	65	1	S3 Get, DB Conn x1
Load	Node.js	4	18	83	1	S3 Get, DB Conn x1
Query	Java	4	36	111	7	S3 Put, DB Conn x2
Query	Python	5	44	96	9	S3 Put, DB Conn x2
Query	Go	4	34	104	8	S3 Put, DB Conn x2
Query	Node.js	5	17	74	1	S3 Put, DB Conn x2

Code Available at [github.com/wlloydw/FaaSProgLangComp](https://github.com/wlloydw/FaaSProgLangComp)



# Outline

- Background and Motivation
- Research Questions
- Serverless Application Analytics Framework (SAAF)
- TLQ Pipeline and Static Code Analysis
- ➔ Experiments and Results
- Conclusions

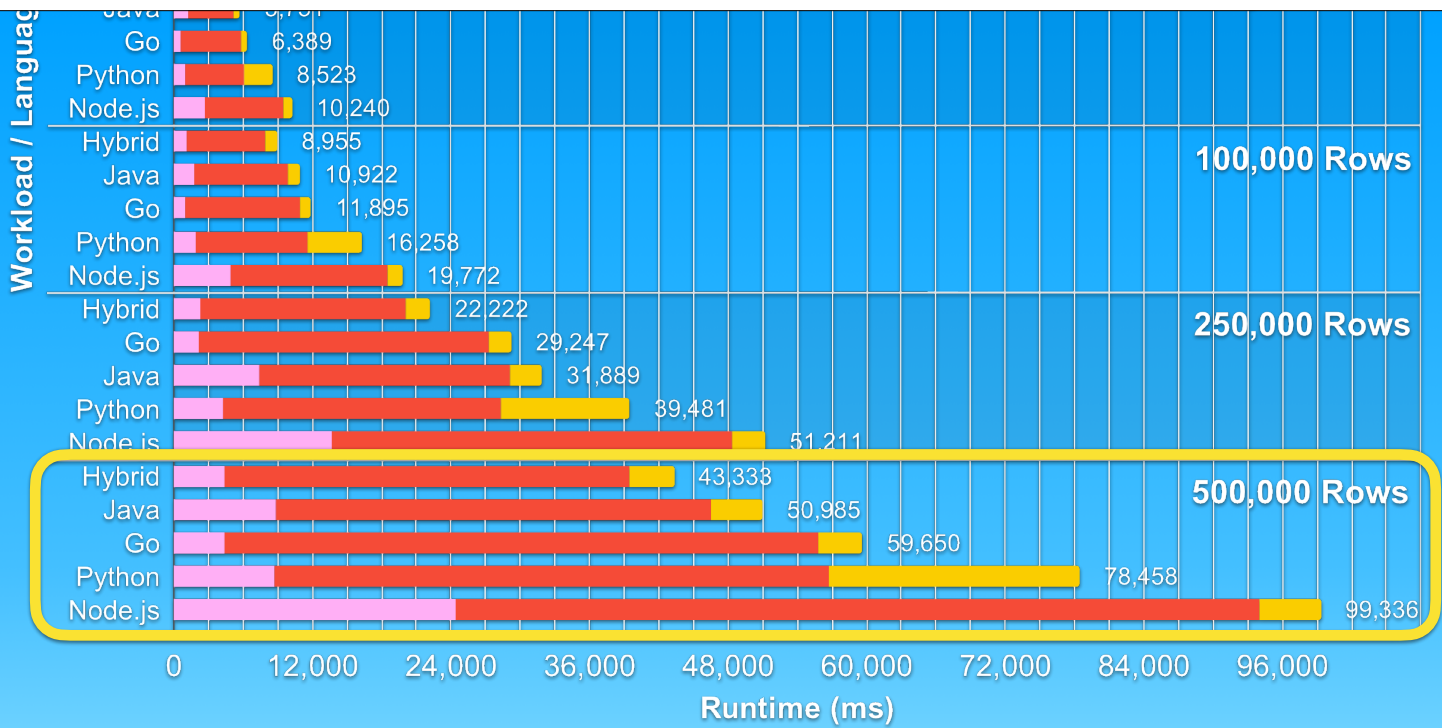
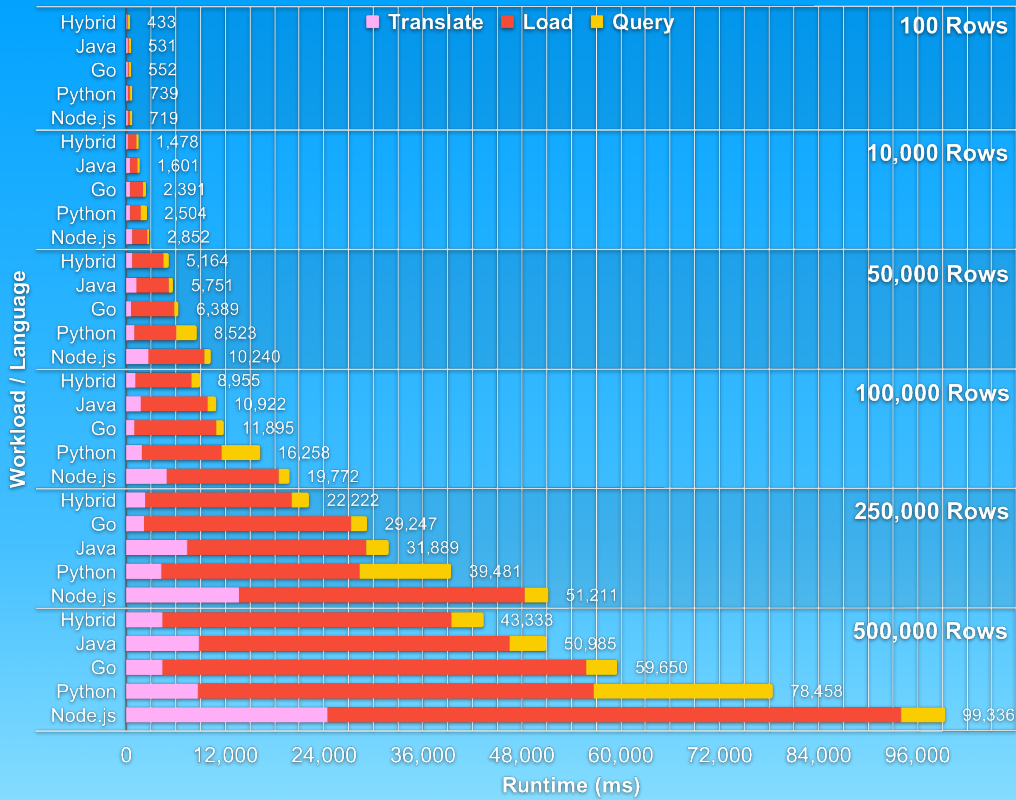
21

## Experiment 1: Overall Performance Comparison

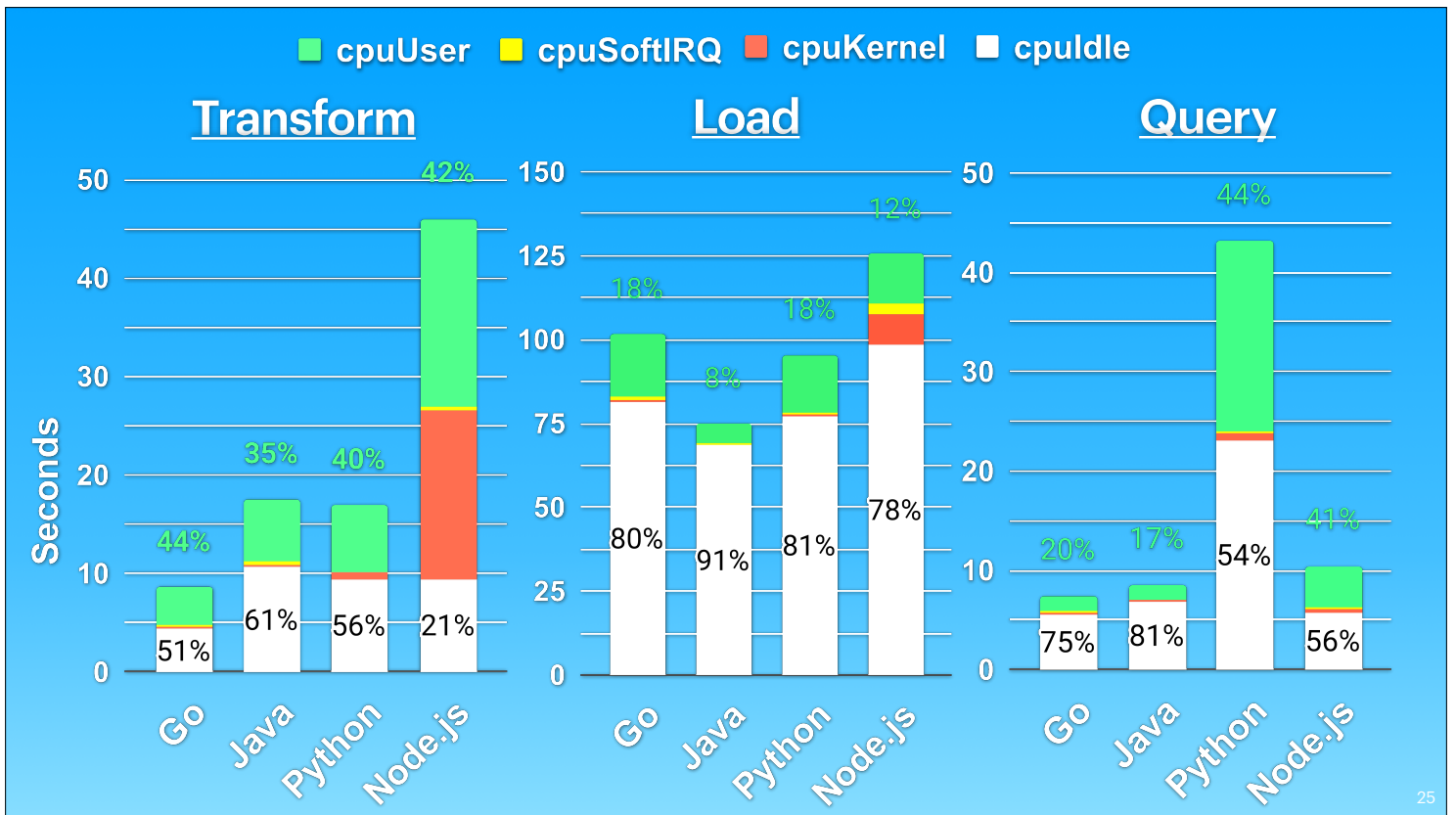


Compare function runtime across different workload sizes.

22



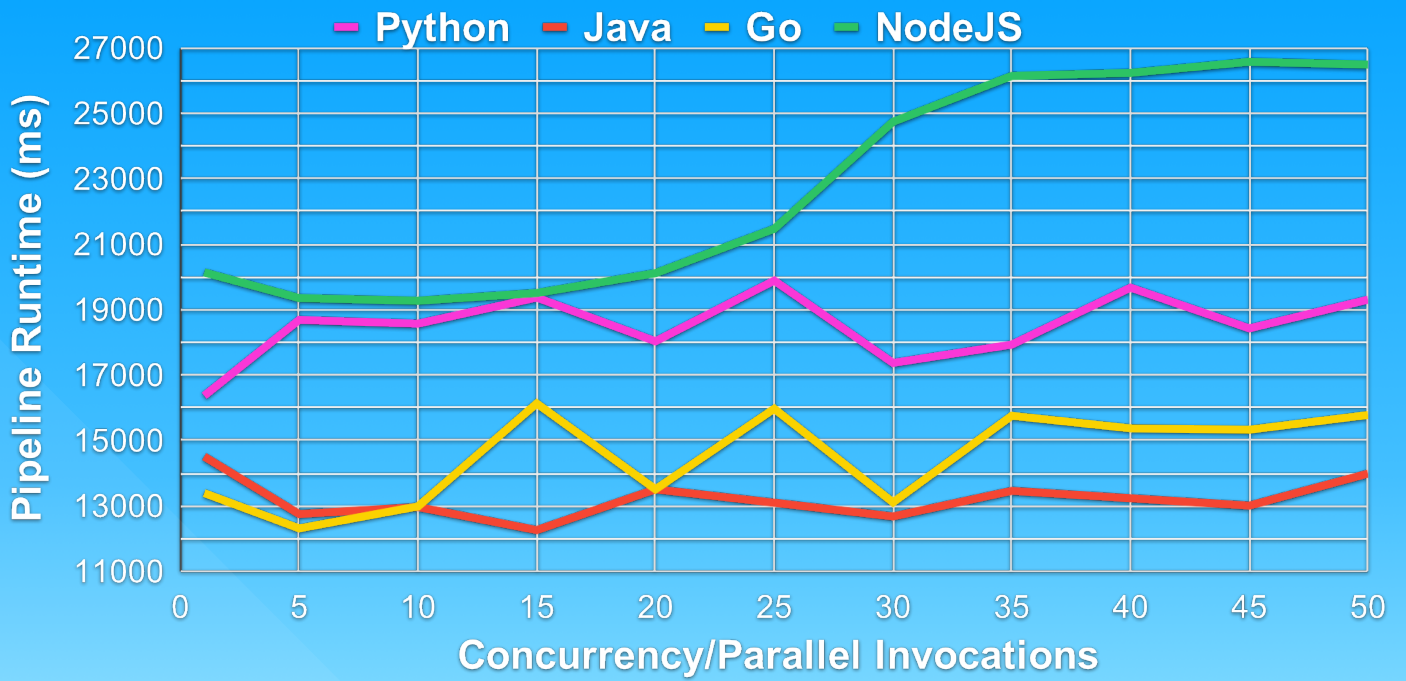
Hybrid Pipeline outperformed Java by 17%, Go by 37%, Python by 81%, and Node.js by 129%.



## Experiment 2: Scalability Performance Testing



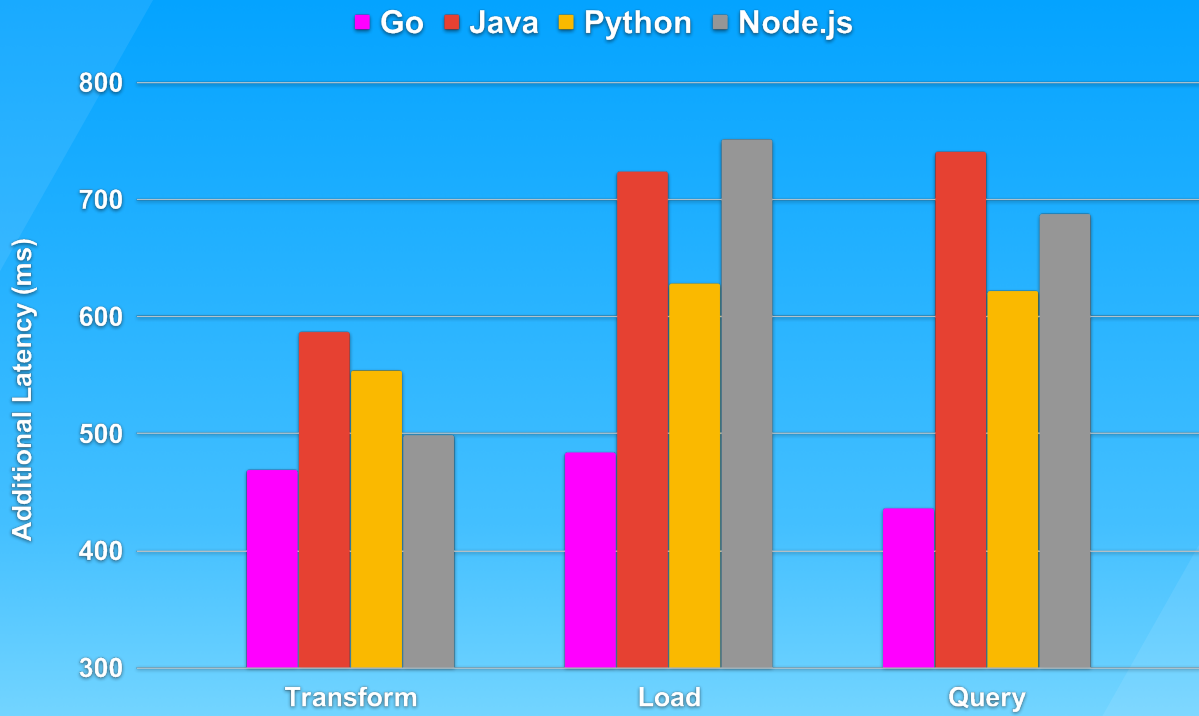
Compare function runtime as the number of concurrent calls is increased.



## Experiment 3: Cold/Warm Performance

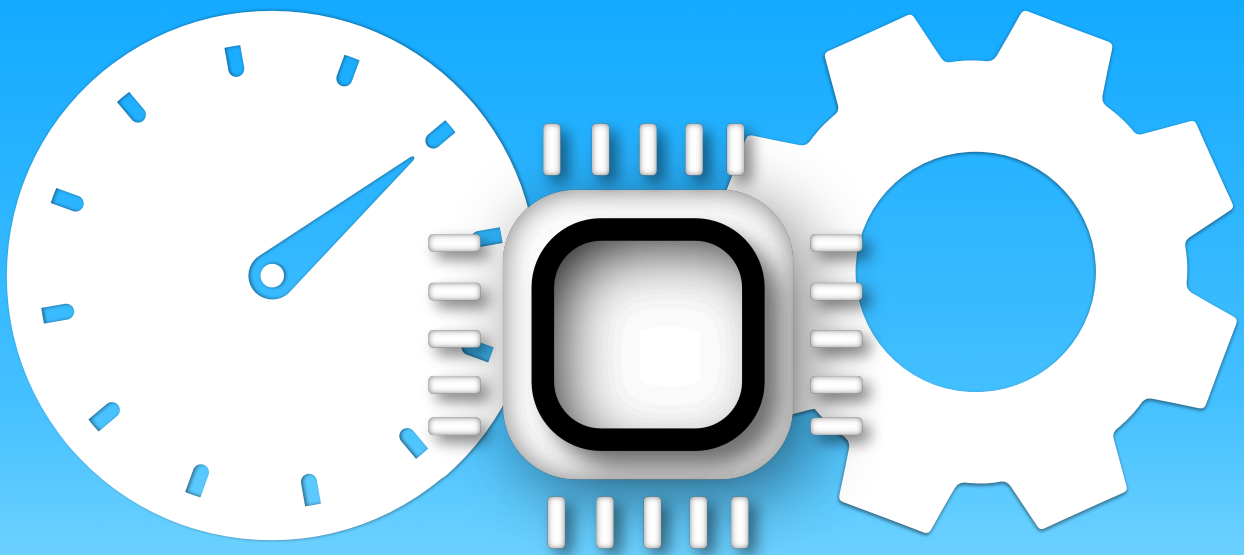


Compare function latency between cold and warm FaaS Infrastructure.



Go: 463 ms, Java: 684 ms, Python 602 ms, Node.js 645 ms

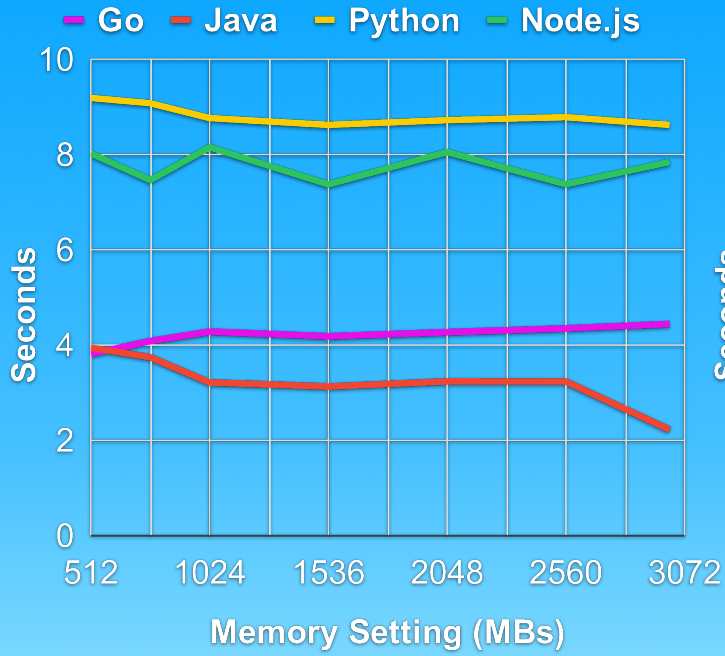
## Experiment 4: Memory Configuration Comparison



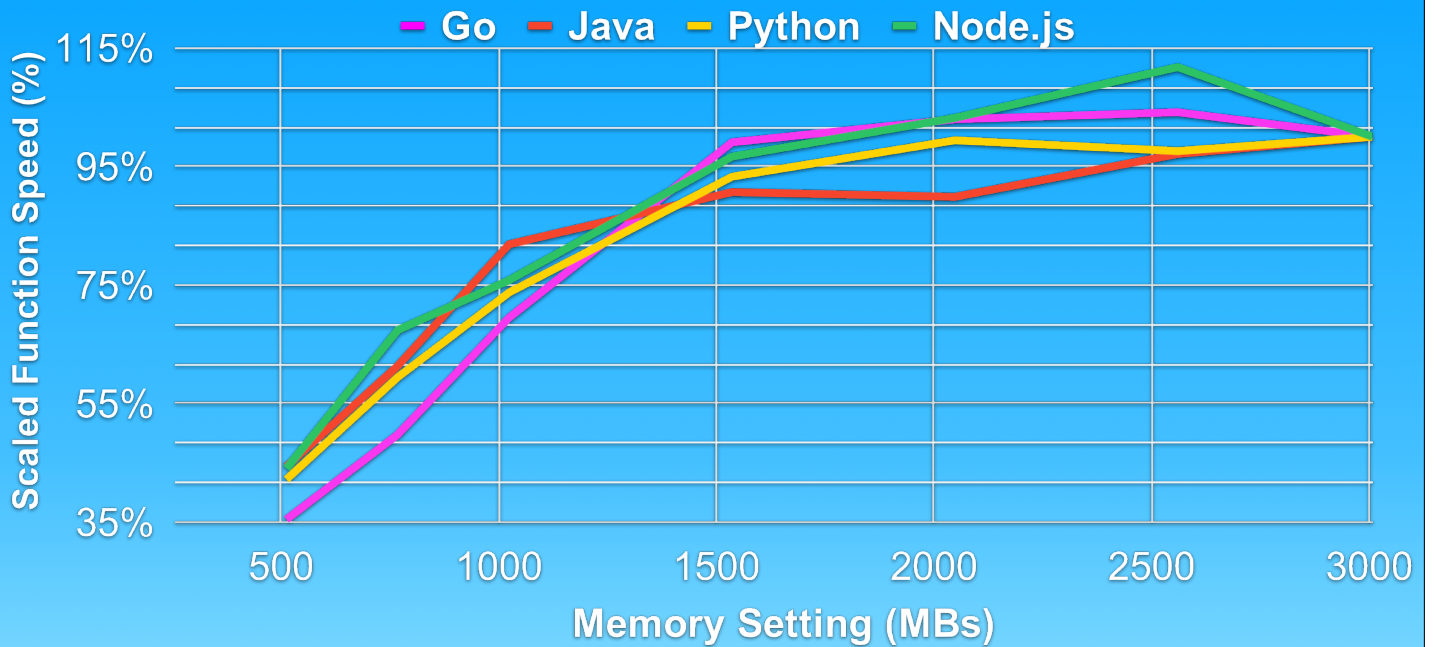
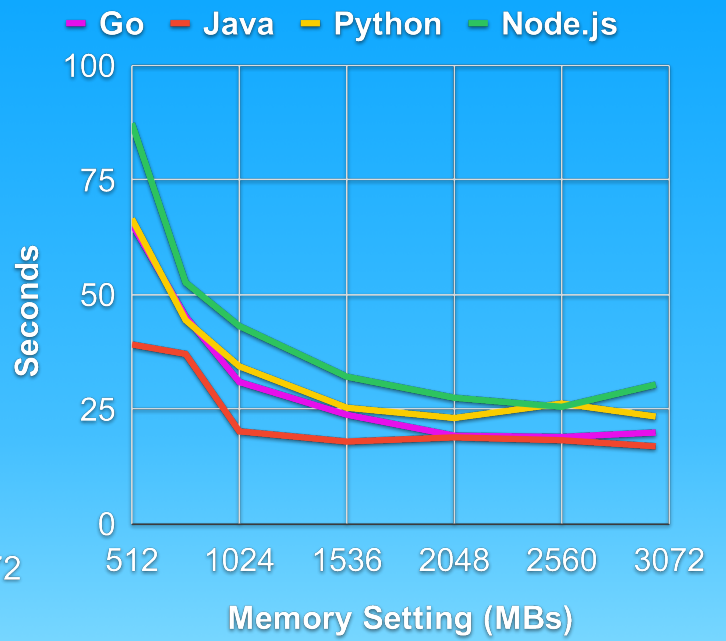
Compare FaaS performance scaling as memory setting is changed.



### CPU User Time by Memory Setting



### CPU Idle Time by Memory Setting





# Outline

- Background and Motivation
- Research Questions
- Serverless Application Analytics Framework (SAAF)
- TLQ Pipeline and Static Code Analysis
- Experiments and Results



Conclusions

33

# Conclusions



**RQ-1:** (Performance) How does the choice of programming language (Java, Go, Python, Node.js) impact the overall performance and throughput of a serverless data processing pipeline?

For a single language, Java offered the best performance, outperforming Node.js by 94%. The fastest pipeline used a hybrid combination of both Go and Java functions.

34

## Conclusions



**RQ-2:** (Scalability) How does programming language choice impact the scalability of a serverless data processing pipeline when processing many concurrent data payloads?

All languages performed similarly with Node.js performing negatively for workloads with higher concurrency.

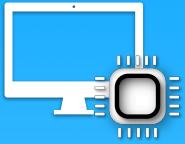
## Conclusions



**RQ-3:** (Infrastructure State) How does the choice of programming language impact cold FaaS performance compared to warm FaaS performance for a data processing pipeline?

Java, Python, and Node.js had similar latency, while Go had about 33% less latency than Java.

# Conclusions



**RQ-4:** (Memory/Cost) How does performance vary for a serverless data processing pipeline across alternate memory settings for implementations in different languages.

Performance scaled approximately linearly for memory sizes up to 1.5 GBs for all pipelines. Beyond 1.5 GB, no major performance improvements were observed.

37



38

# Thank You for Watching

## Questions or comments?

Please email:  
rcording@uw.edu or wlloyd@uw.edu

## Download Serverless Application Analytics Framework

[github.com/wlloyduw/saaf](https://github.com/wlloyduw/saaf)



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