Sky Computing for Serverless:
Infrastructure Assessment to Support Performance Enhancement

Robert Cordingly, Xinghan Chen, Ling-Hong Hung, Wes Lloyd
rcording@uw.edu

School of Engineering and Technology
University of Washington Tacoma

IEEE/ACM 18th International Conference on Utility and Cloud Computing (UCC 25) December 1-4, 2025, Nantes France

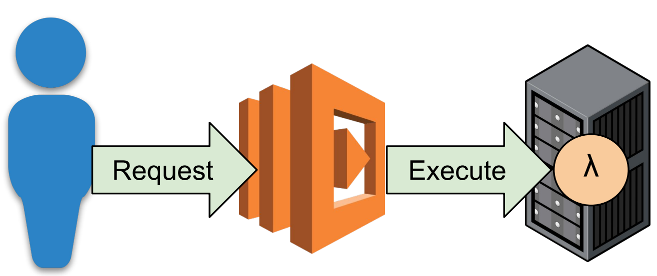
Background and Motivation

What is Serverless?

Serverless function-as-a-service (FaaS) platforms offer many desirable features:

- Rapid elastic scaling
- Scale to zero
- No infrastructure management
- Fine grained billing
- Fault tolerance
- No up front cost to deploy an application

How FaaS Platforms Work



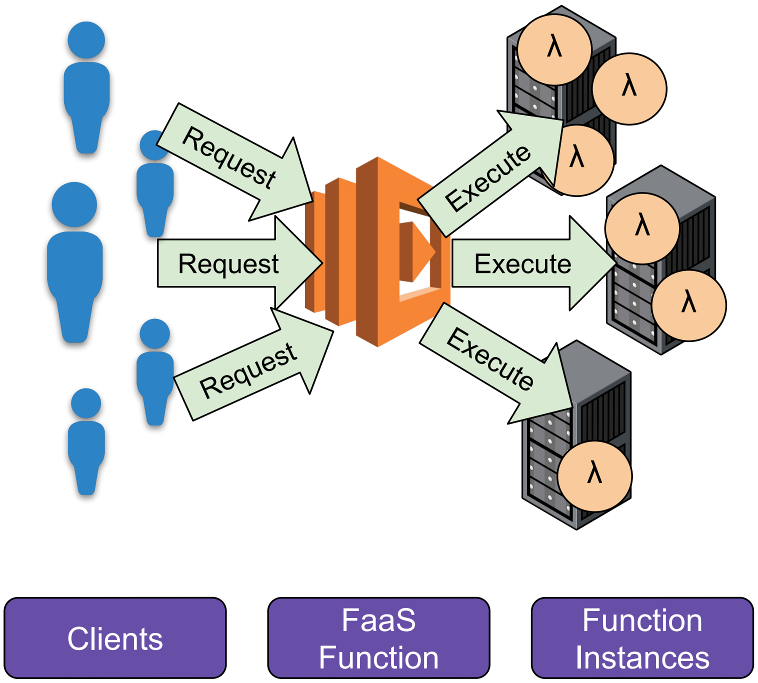
Client | FaaS Platform | Function Instance

- Clients make requests to the FaaS platform which manage the infrastructure automatically

- FaaS platforms create and execute users code inside environments known as function instances

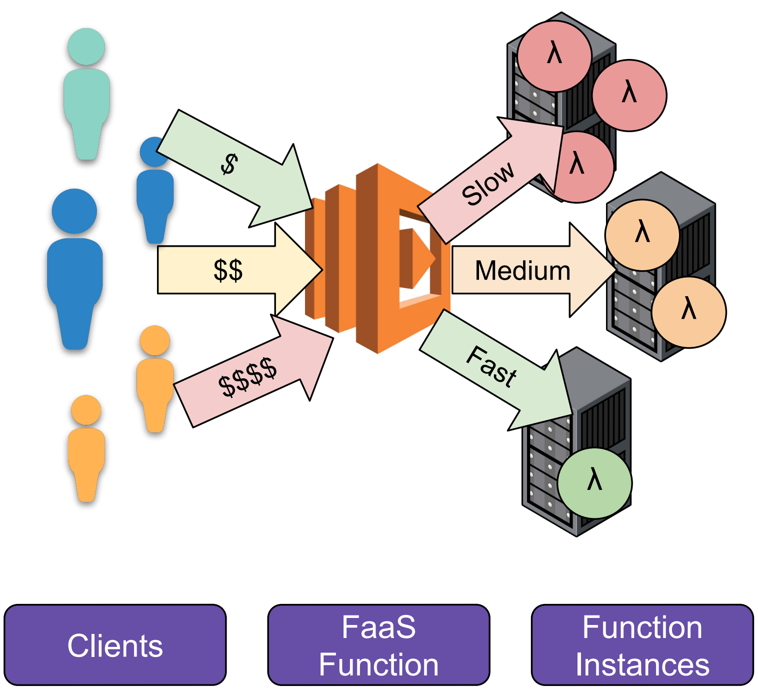
- Function instances can be hosted on a variety of different kinds of hardware

Automatic Scaling



- As more requests are made, the FaaS platform scales the number of function instances
- Function instances for one function can be spread across many host servers

Inconsistent Performance



- Heterogeneous hardware leads to some hosts being faster than others
- Since FaaS platforms bill based off runtime, slower hosts perform worse and are more expensive

Request Routing

- If we were able to observe all of the available infrastructure of serverless regions, we could route requests to regions with more favorable infrastructure

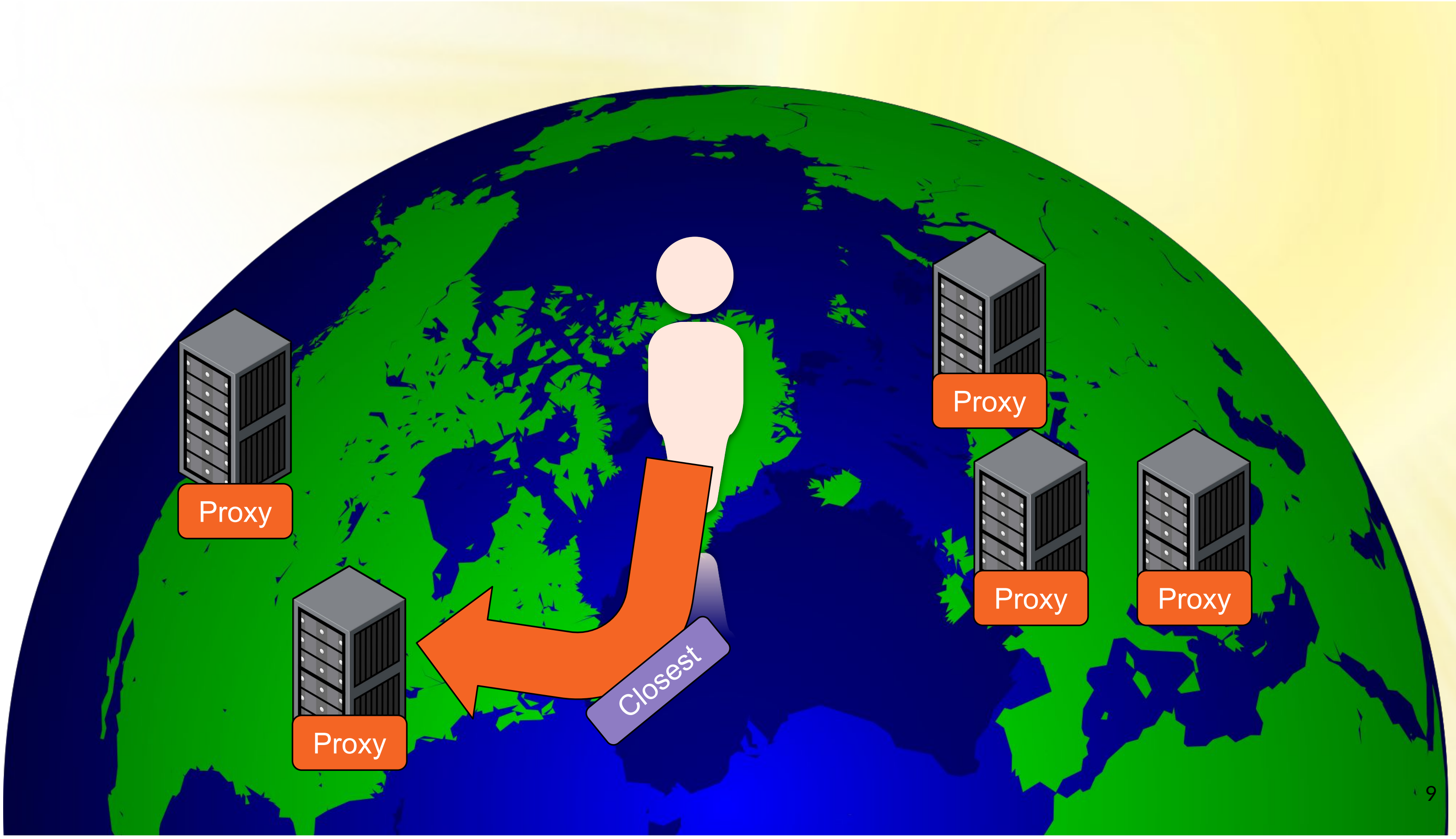
- Our previous work built a pserverless sky proxy system to route requests around the world (IC2E 2023)

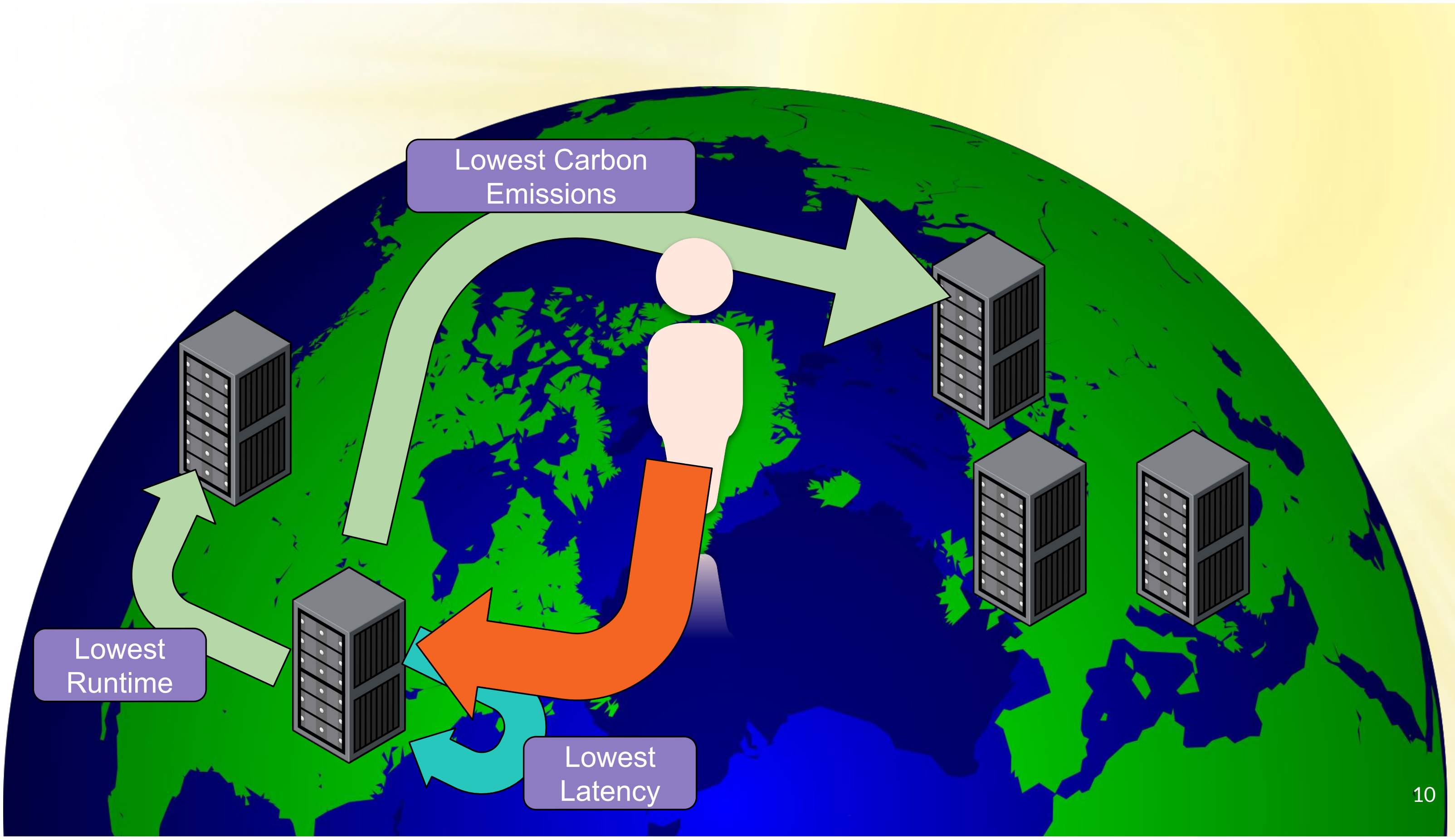
What is Sky Computing?

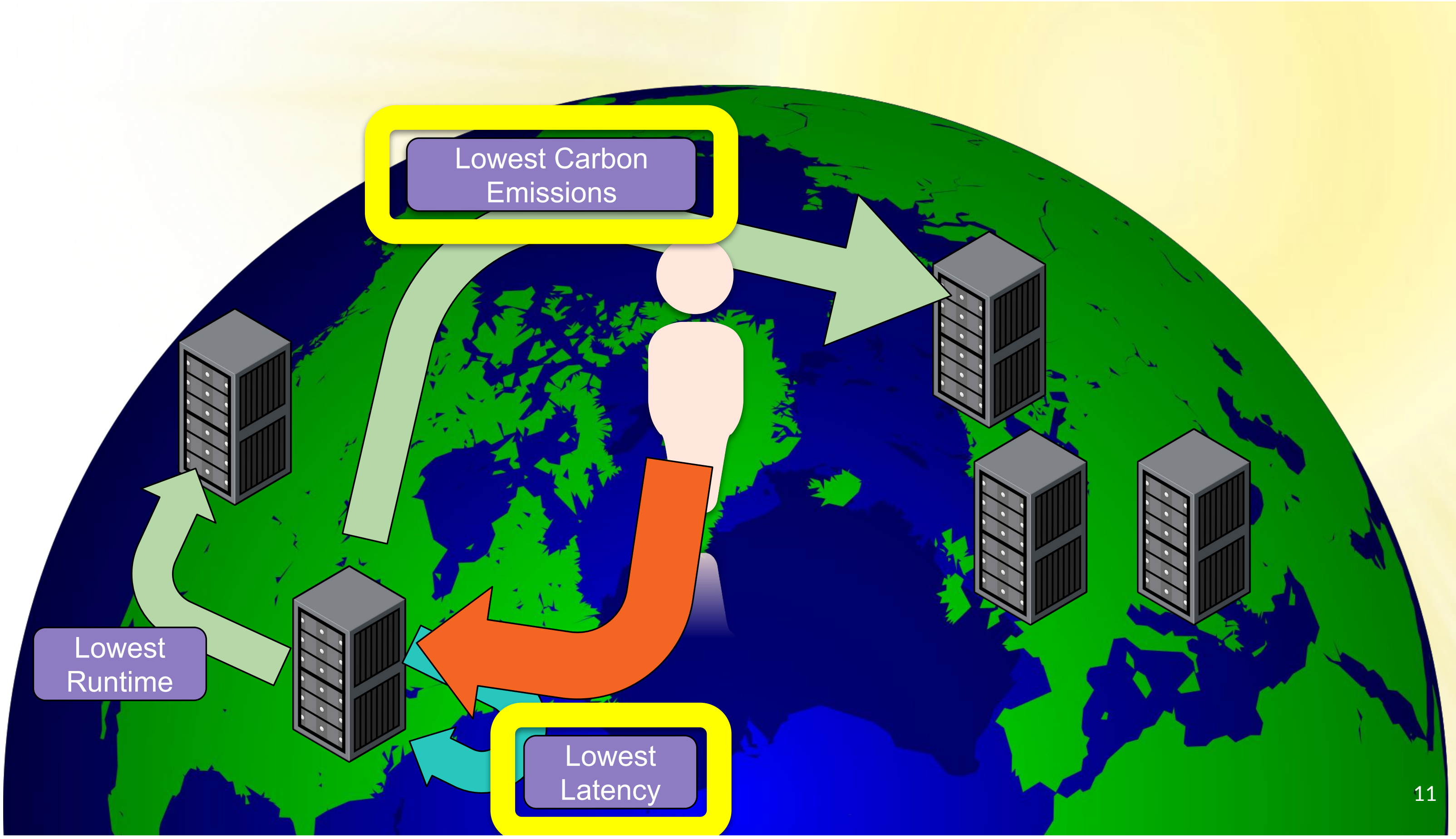
- The Sky sits above the clouds
- Consists of compatibility layers allowing aggregation of resources between cloud regions, availability zones, or cloud providers

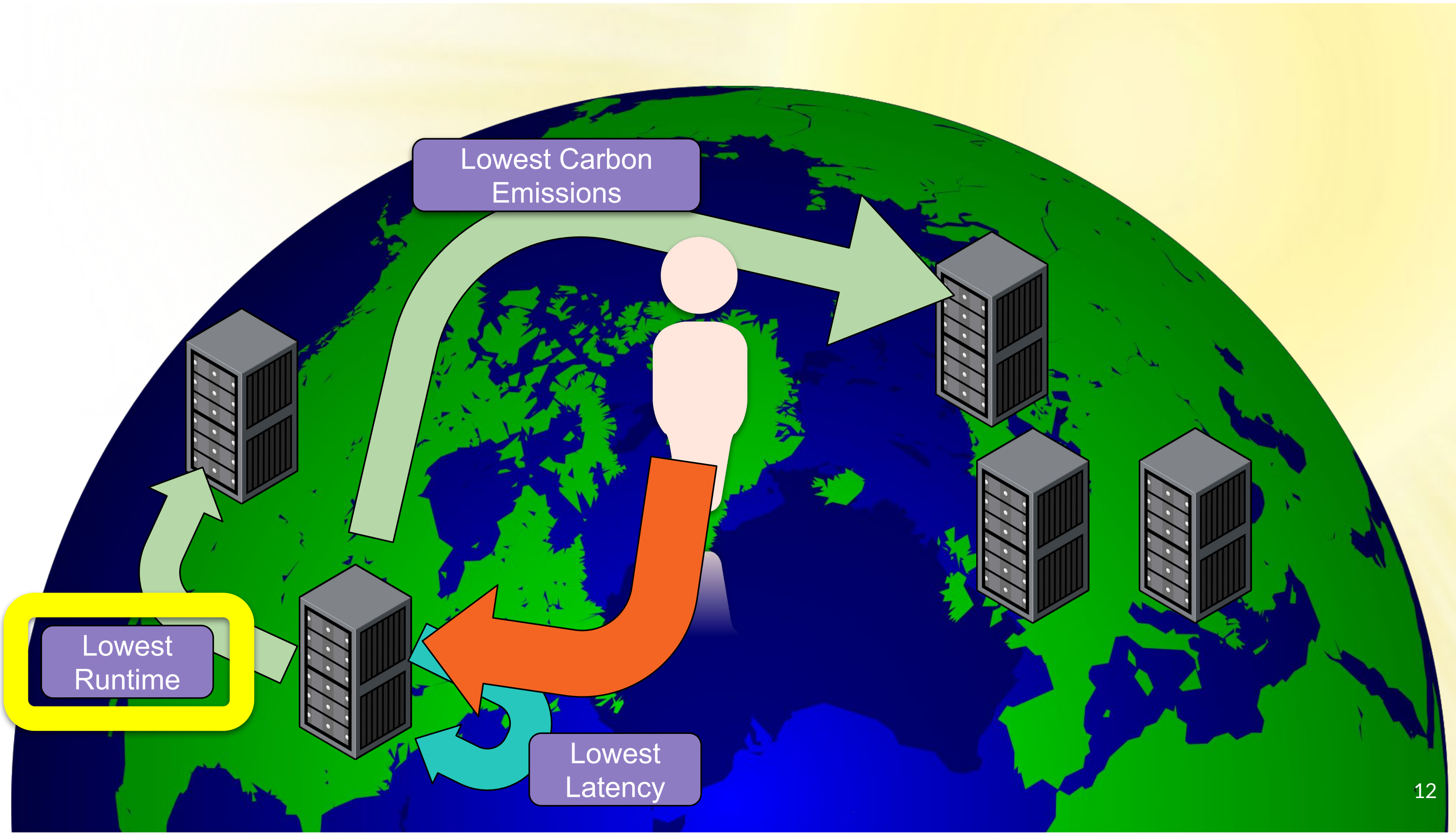
Goals for Serverless Sky Computing:

- Allow applications to take advantage of resources of multiple regions and cloud providers
- Improve performance and reduce costs of FaaS applications


Proxy
Proxy
Proxy
Proxy
Proxy
Closest
9


Lowest Carbon Emissions
Lowest Runtime
Lowest Latency
10

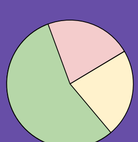

Lowest Carbon Emissions
Lowest Runtime
Lowest Latency


Lowest Carbon Emissions
Lowest Runtime
Lowest Latency

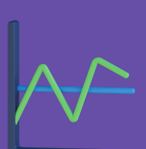
Research Questions

Research Questions

- RQ-1 (**Infrastructure Variation**): What CPU variations can be observed on FaaS platforms across different zones and regions? How does it change over time?

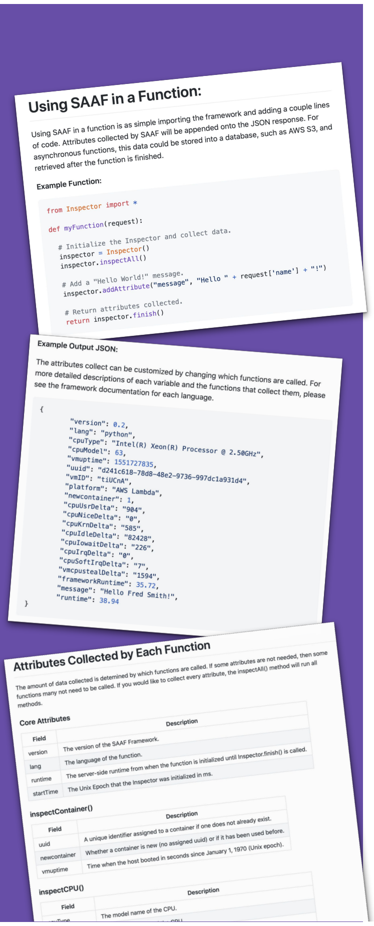


- RQ-2 (**Infrastructure Characterization**): How many samples are required to accurately infer the hardware pool? How can we balance the cost versus accuracy?



- RQ-3 (**Performance Optimization**): To what extent are runtime and cost improvements possible by targeting specific instructure?

Serverless Infrastructure Observation



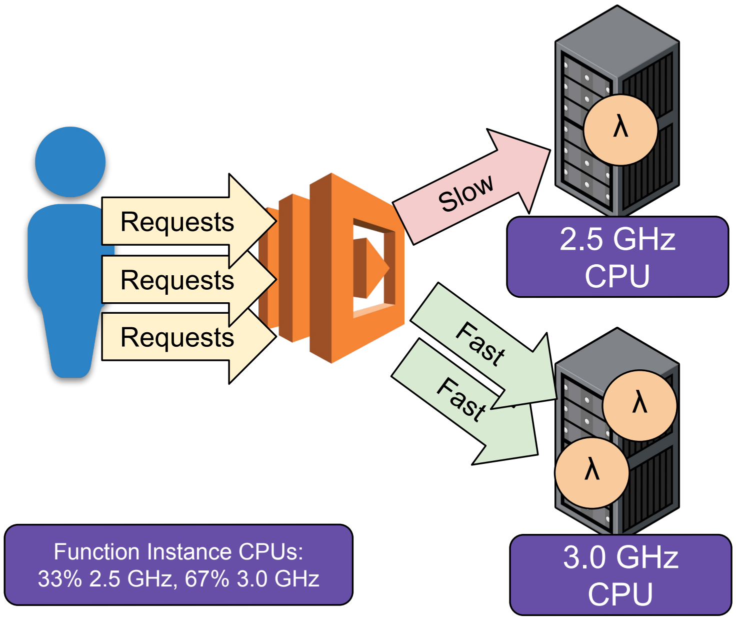
Supporting Tools - SAAF

We utilize the **Serverless Application Analytics Framework** to collect various metrics about the infrastructure and platform serverless functions are hosted with.

SAAF collected CPU Timing metrics, latency information, hardware specifications, runtime metrics, and more.

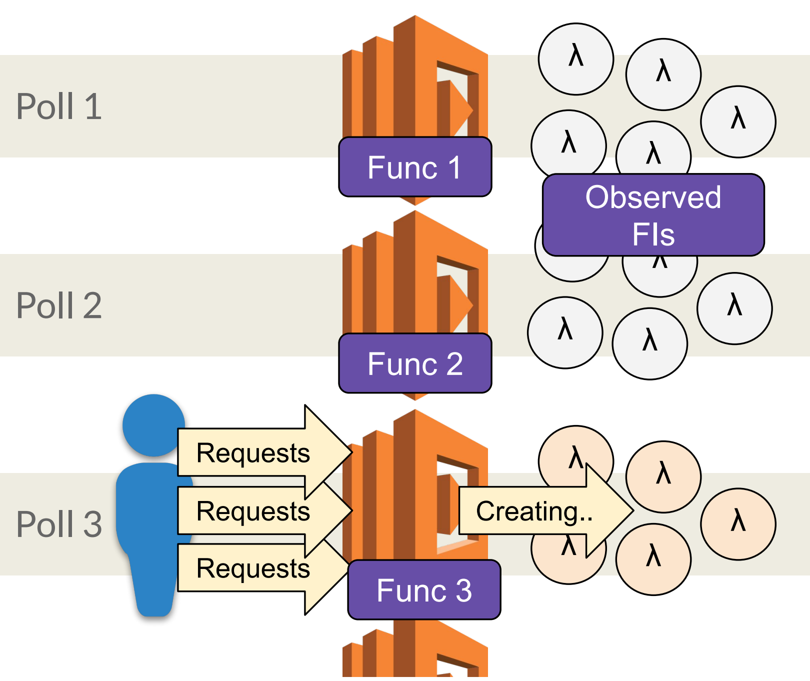
We utilize CPU metrics from SAAF to characterize function instances and observe details about the host infrastructure.

Observing Infrastructure



- We deployed “Hello World” sleep functions with SAAF that collect CPU information
- We then make parallel requests to view available infrastructure for a function
- With SAAF’s data we build a characterization of the pool of available infrastructure
- But this method is limited…

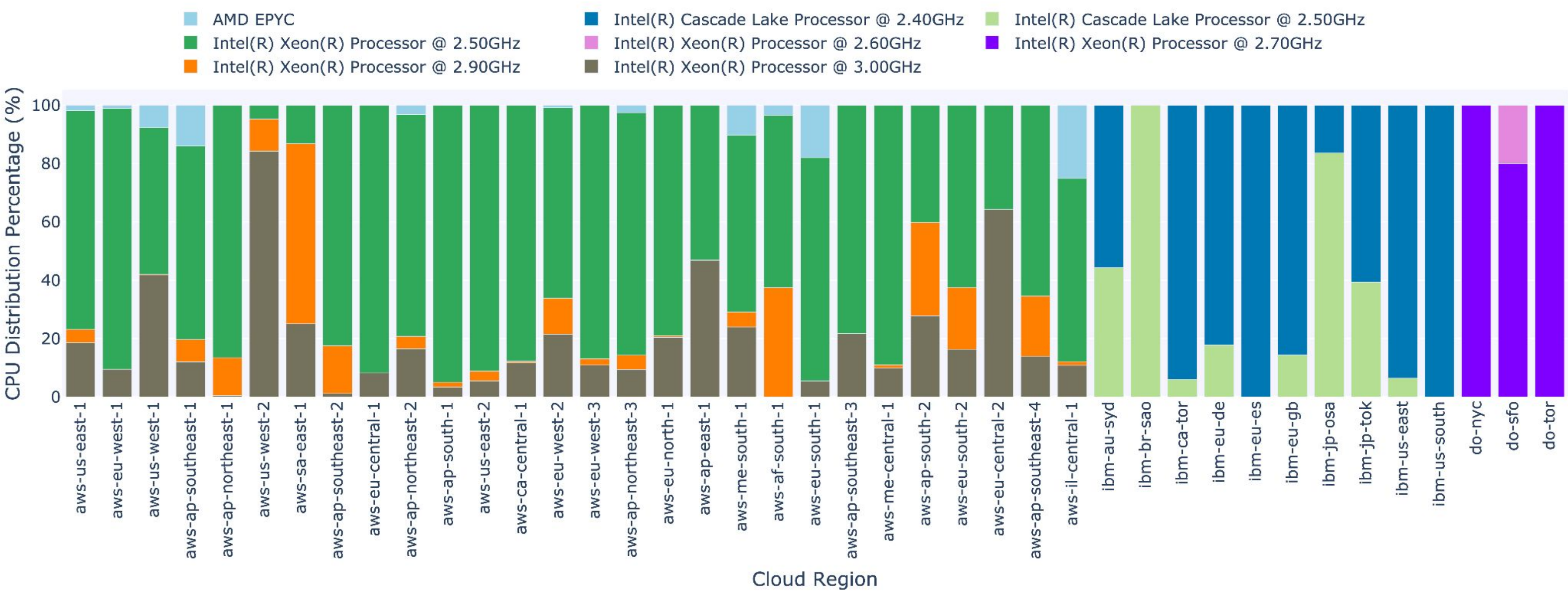
Creating Function Instances

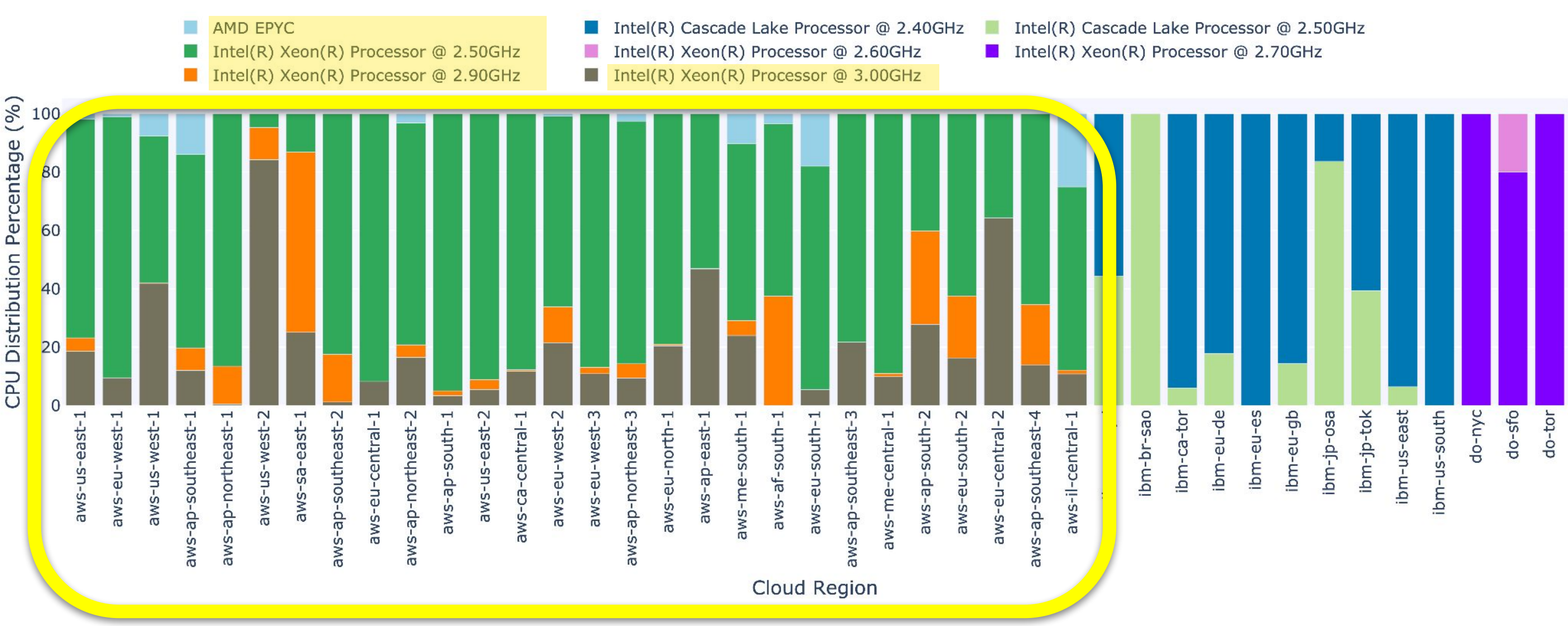


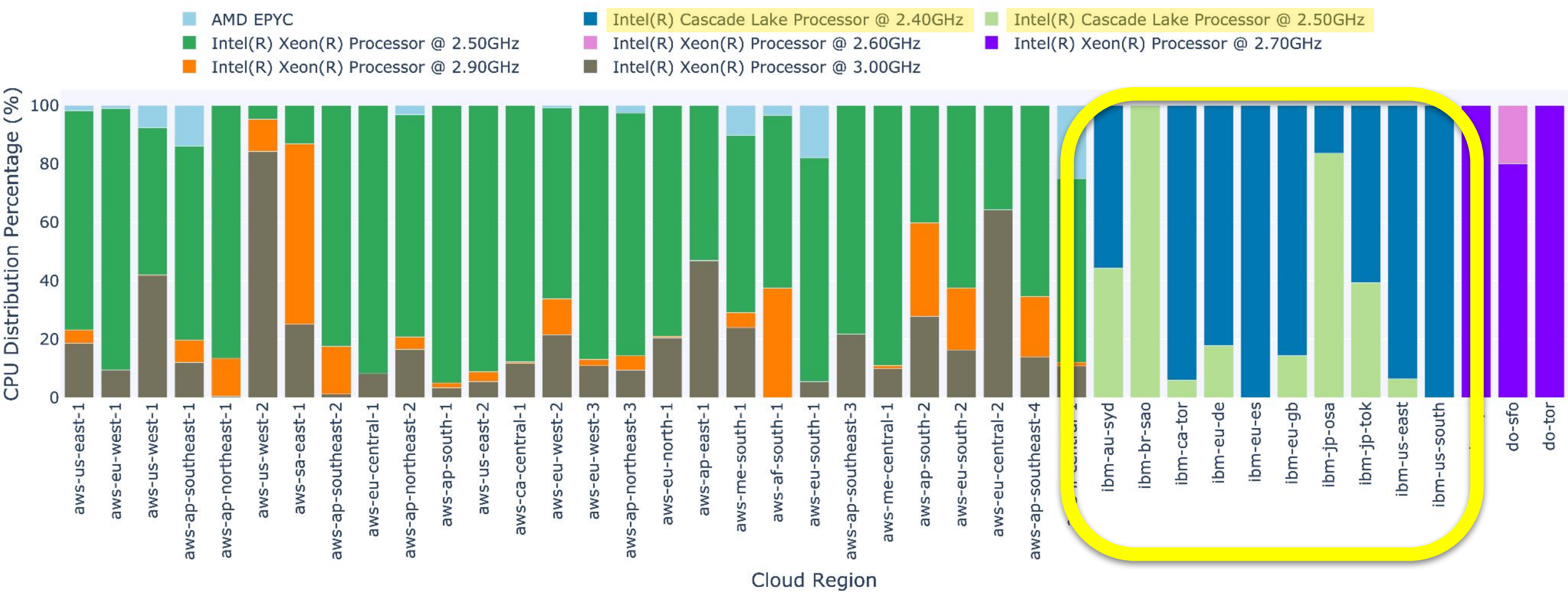
- To observe as much infrastructure as possible we need to create many function instances
- FaaS platforms limit the number of concurrent executions
(1000 on AWS Lambda)
- By utilizing many function deployments, we can create more function instances than the concurrency limit
- With this we can build accurate characterizations of available CPUs in availability zones (AZs)

Methodology and Results

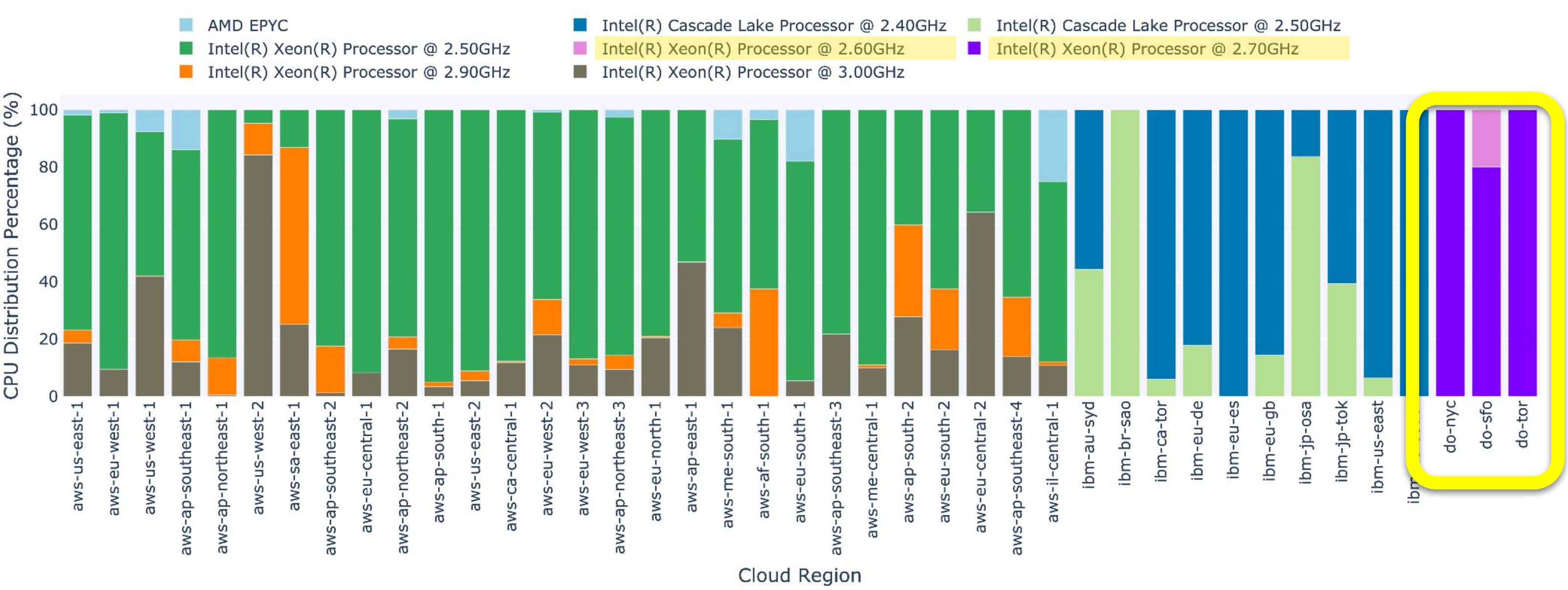
RQ-1 (Part 1): What CPU variations can be observed on FaaS platforms across different zones and regions?


AMD EPYC
Intel(R) Xeon(R) Processor @ 2.50GHz
Intel(R) Xeon(R) Processor @ 2.90GHz
Intel(R) Cascade Lake Processor @ 2.40GHz
Intel(R) Xeon(R) Processor @ 2.60GHz
Intel(R) Xeon(R) Processor @ 3.00GHz
Intel(R) Cascade Lake Processor @ 2.50GHz
Intel(R) Xeon(R) Processor @ 2.70GHz
CPU Distribution Percentage (%)
Cloud Region

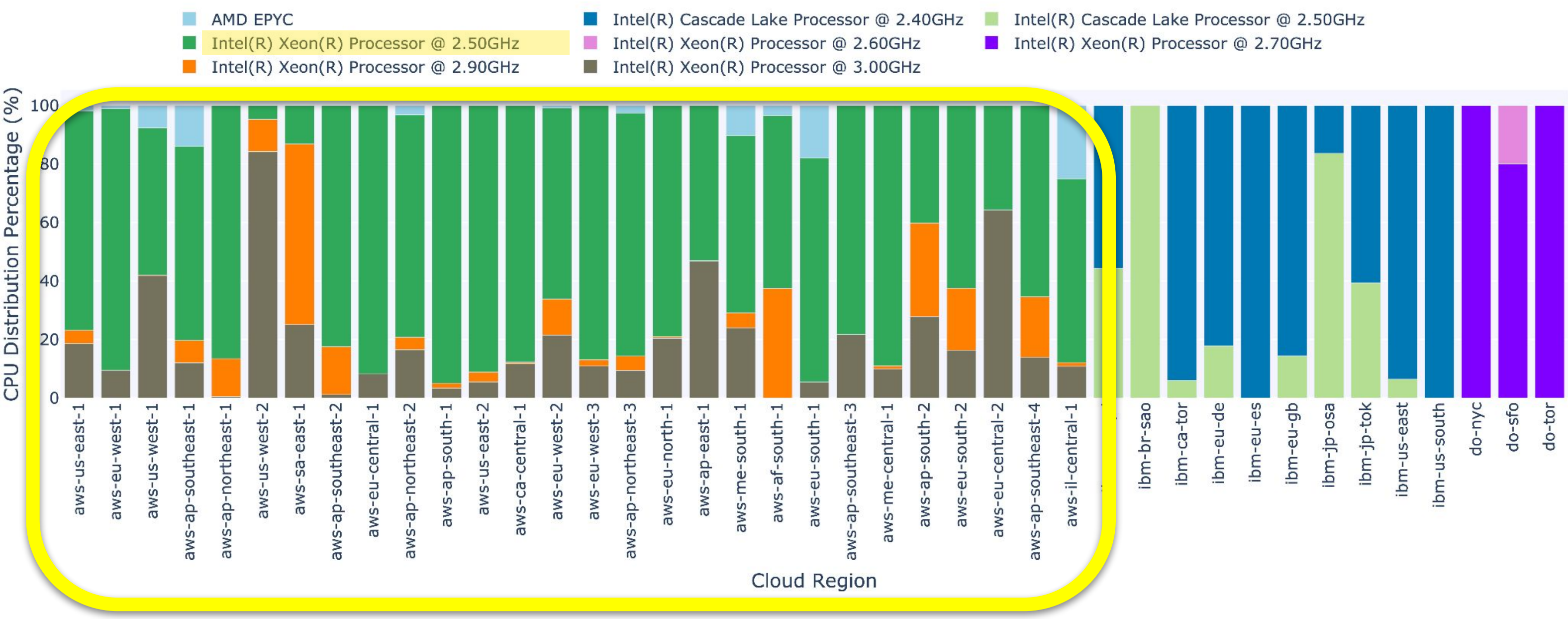

AMD EPYC
Intel(R) Xeon(R) Processor @ 2.50GHz
Intel(R) Xeon(R) Processor @ 2.90GHz
Intel(R) Cascade Lake Processor @ 2.40GHz
Intel(R) Xeon(R) Processor @ 2.60GHz
Intel(R) Xeon(R) Processor @ 3.00GHz
Intel(R) Cascade Lake Processor @ 2.50GHz
Intel(R) Xeon(R) Processor @ 2.70GHz
CPU Distribution Percentage (%)
Cloud Region


AMD EPYC
Intel(R) Xeon(R) Processor @ 2.50GHz
Intel(R) Xeon(R) Processor @ 2.90GHz
Intel(R) Cascade Lake Processor @ 2.40GHz
Intel(R) Xeon(R) Processor @ 2.60GHz
Intel(R) Xeon(R) Processor @ 3.00GHz
Intel(R) Cascade Lake Processor @ 2.50GHz
Intel(R) Xeon(R) Processor @ 2.70GHz
CPU Distribution Percentage (%)
0
20
40
60
80
100
aws-us-east-1
aws-eu-west-1
aws-us-west-1
aws-ap-southeast-1
aws-ap-northeast-1
aws-us-west-2
aws-sa-east-1
aws-ap-southeast-2
aws-eu-central-1
aws-ap-northeast-2
aws-ap-south-1
aws-us-east-2
aws-ca-central-1
aws-eu-west-2
aws-eu-west-3
aws-ap-northeast-3
aws-eu-north-1
aws-ap-east-1
aws-me-south-1
aws-af-south-1
aws-eu-south-1
aws-ap-southeast-3
aws-me-central-1
aws-ap-south-2
aws-eu-south-2
aws-eu-central-2
aws-ap-southeast-4
ibm-au-syd
ibm-br-sao
ibm-ca-tor
ibm-eu-de
ibm-eu-es
ibm-eu-gb
ibm-jp-osa
ibm-jp-tok
ibm-us-east
ibm-us-south
do-sfo
do-tor
Cloud Region

23

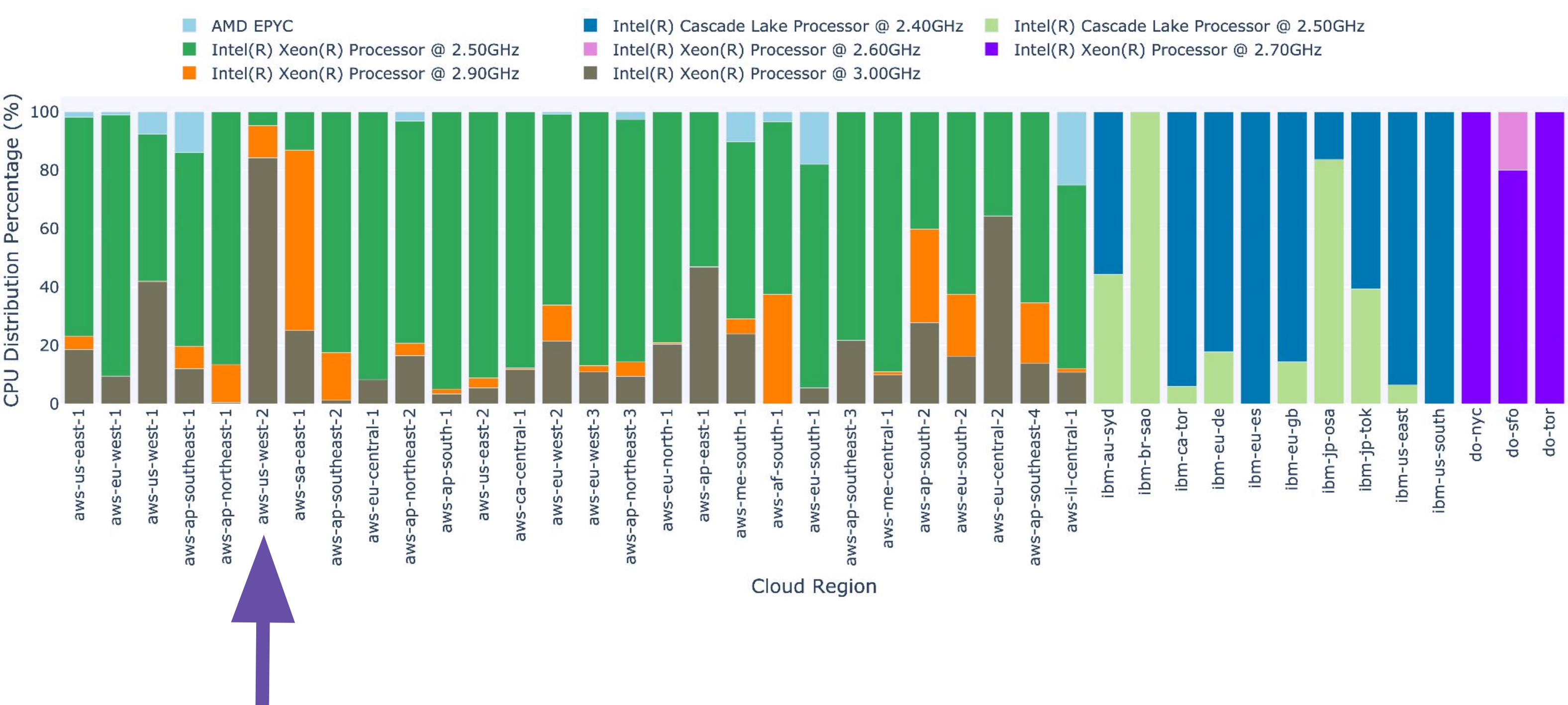

AMD EPYC
Intel(R) Xeon(R) Processor @ 2.50GHz
Intel(R) Xeon(R) Processor @ 2.90GHz
Intel(R) Cascade Lake Processor @ 2.40GHz
Intel(R) Xeon(R) Processor @ 2.60GHz
Intel(R) Xeon(R) Processor @ 3.00GHz
Intel(R) Cascade Lake Processor @ 2.50GHz
Intel(R) Xeon(R) Processor @ 2.70GHz
CPU Distribution Percentage (%)
0
20
40
60
80
100
aws-us-east-1
aws-eu-west-1
aws-us-west-1
aws-ap-southeast-1
aws-ap-northeast-1
aws-us-west-2
aws-sa-east-1
aws-ap-southeast-2
aws-eu-central-1
aws-ap-northeast-2
aws-ap-south-1
aws-us-east-2
aws-ca-central-1
aws-eu-west-2
aws-eu-west-3
aws-ap-northeast-3
aws-eu-north-1
aws-ap-east-1
aws-me-south-1
aws-af-south-1
aws-eu-south-1
aws-ap-southeast-3
aws-me-central-1
aws-ap-south-2
aws-eu-south-2
aws-eu-central-2
aws-ap-southeast-4
aws-il-central-1
ibm-au-syd
ibm-br-sao
ibm-ca-tor
ibm-eu-de
ibm-eu-es
ibm-eu-gb
ibm-jp-osa
ibm-jp-tok
ibm-us-east
do-nyc
do-sfo
do-tor
Cloud Region

24

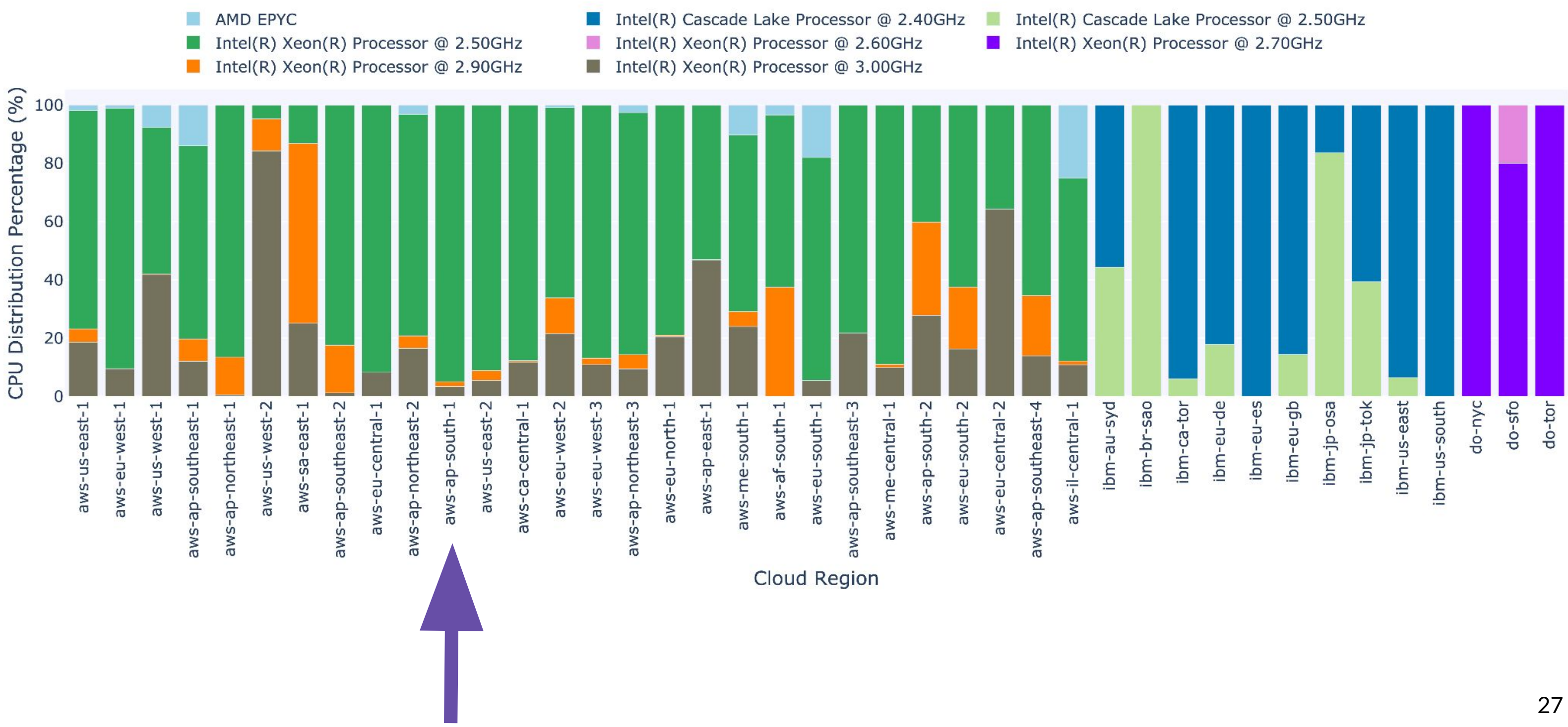


AMD EPYC
Intel(R) Xeon(R) Processor @ 2.50GHz
Intel(R) Xeon(R) Processor @ 2.90GHz
Intel(R) Cascade Lake Processor @ 2.40GHz
Intel(R) Xeon(R) Processor @ 2.60GHz
Intel(R) Xeon(R) Processor @ 3.00GHz
Intel(R) Cascade Lake Processor @ 2.50GHz
Intel(R) Xeon(R) Processor @ 2.70GHz
CPU Distribution Percentage (%)
Cloud Region

25

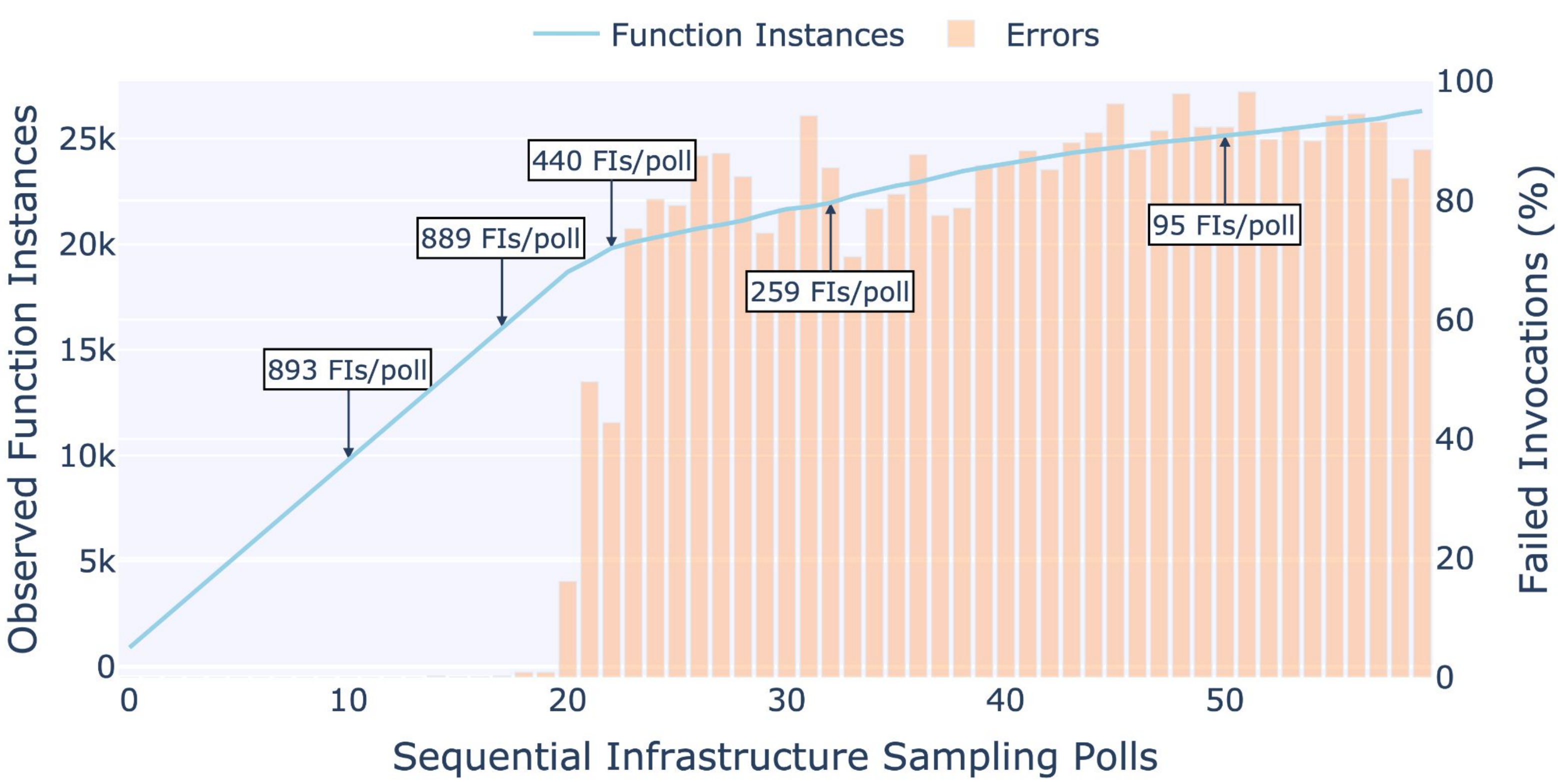


AMD EPYC
Intel(R) Xeon(R) Processor @ 2.50GHz
Intel(R) Xeon(R) Processor @ 2.90GHz
Intel(R) Cascade Lake Processor @ 2.40GHz
Intel(R) Xeon(R) Processor @ 2.60GHz
Intel(R) Xeon(R) Processor @ 3.00GHz
Intel(R) Cascade Lake Processor @ 2.50GHz
Intel(R) Xeon(R) Processor @ 2.70GHz
CPU Distribution Percentage (%)
Cloud Region



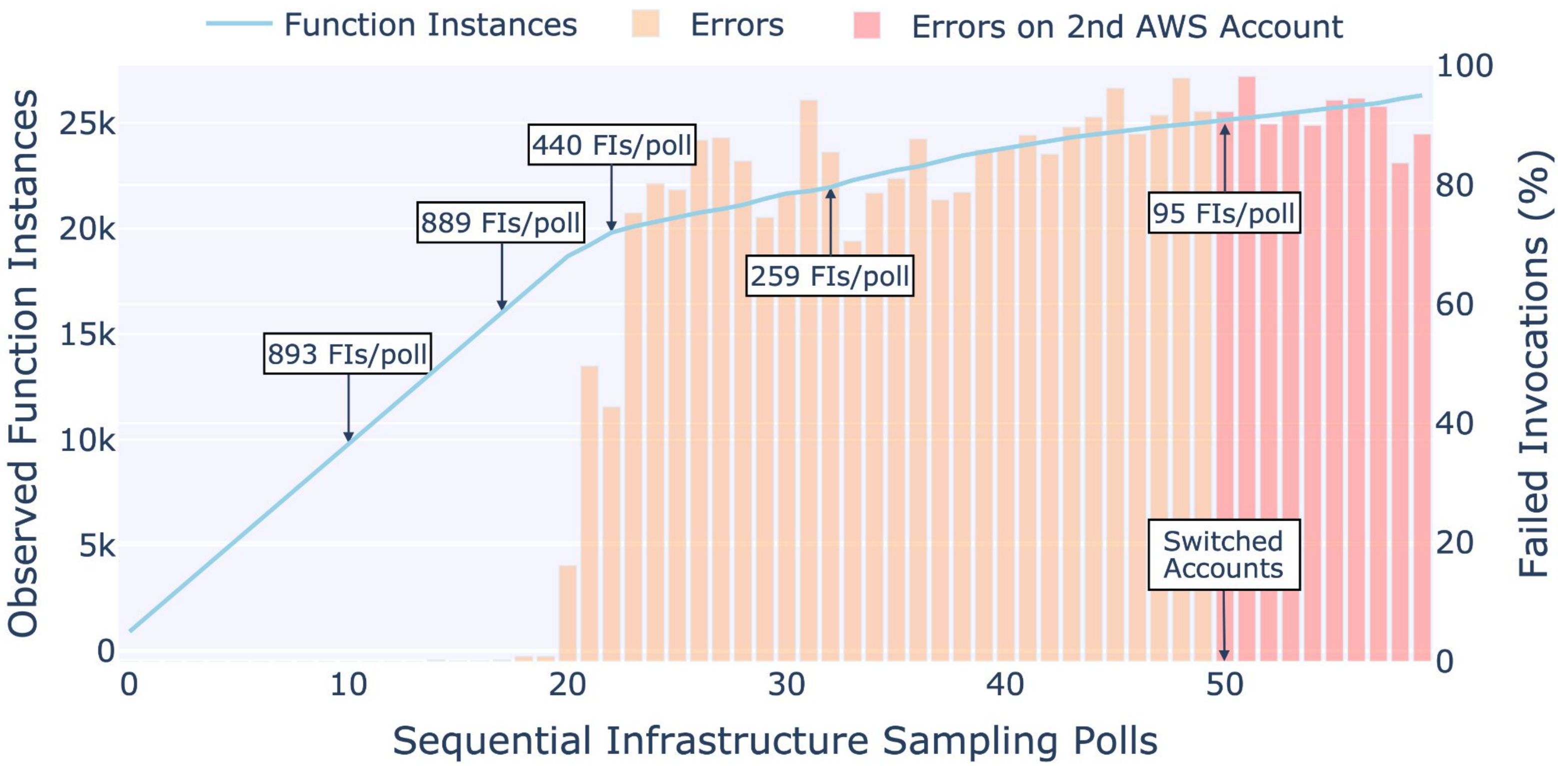
AMD EPYC
Intel(R) Cascade Lake Processor @ 2.40GHz
Intel(R) Cascade Lake Processor @ 2.50GHz
Intel(R) Xeon(R) Processor @ 2.50GHz
Intel(R) Xeon(R) Processor @ 2.60GHz
Intel(R) Xeon(R) Processor @ 2.70GHz
Intel(R) Xeon(R) Processor @ 2.90GHz
Intel(R) Xeon(R) Processor @ 3.00GHz
CPU Distribution Percentage (%)
Cloud Region
aws-us-east-1
aws-eu-west-1
aws-us-west-1
aws-ap-southeast-1
aws-ap-northeast-1
aws-us-west-2
aws-sa-east-1
aws-ap-southeast-2
aws-eu-central-1
aws-ap-northeast-2
aws-ap-south-1
aws-us-east-2
aws-ca-central-1
aws-eu-west-2
aws-eu-west-3
aws-ap-northeast-3
aws-eu-north-1
aws-ap-east-1
aws-me-south-1
aws-af-south-1
aws-eu-south-1
aws-ap-southeast-3
aws-me-central-1
aws-ap-south-2
aws-eu-south-2
aws-eu-central-2
aws-ap-southeast-4
aws-il-central-1
ibm-au-syd
ibm-br-sao
ibm-ca-tor
ibm-eu-de
ibm-eu-es
ibm-eu-gb
ibm-jp-osa
ibm-jp-tok
ibm-us-east
ibm-us-south
do-nyc
do-sfo
do-tor

27

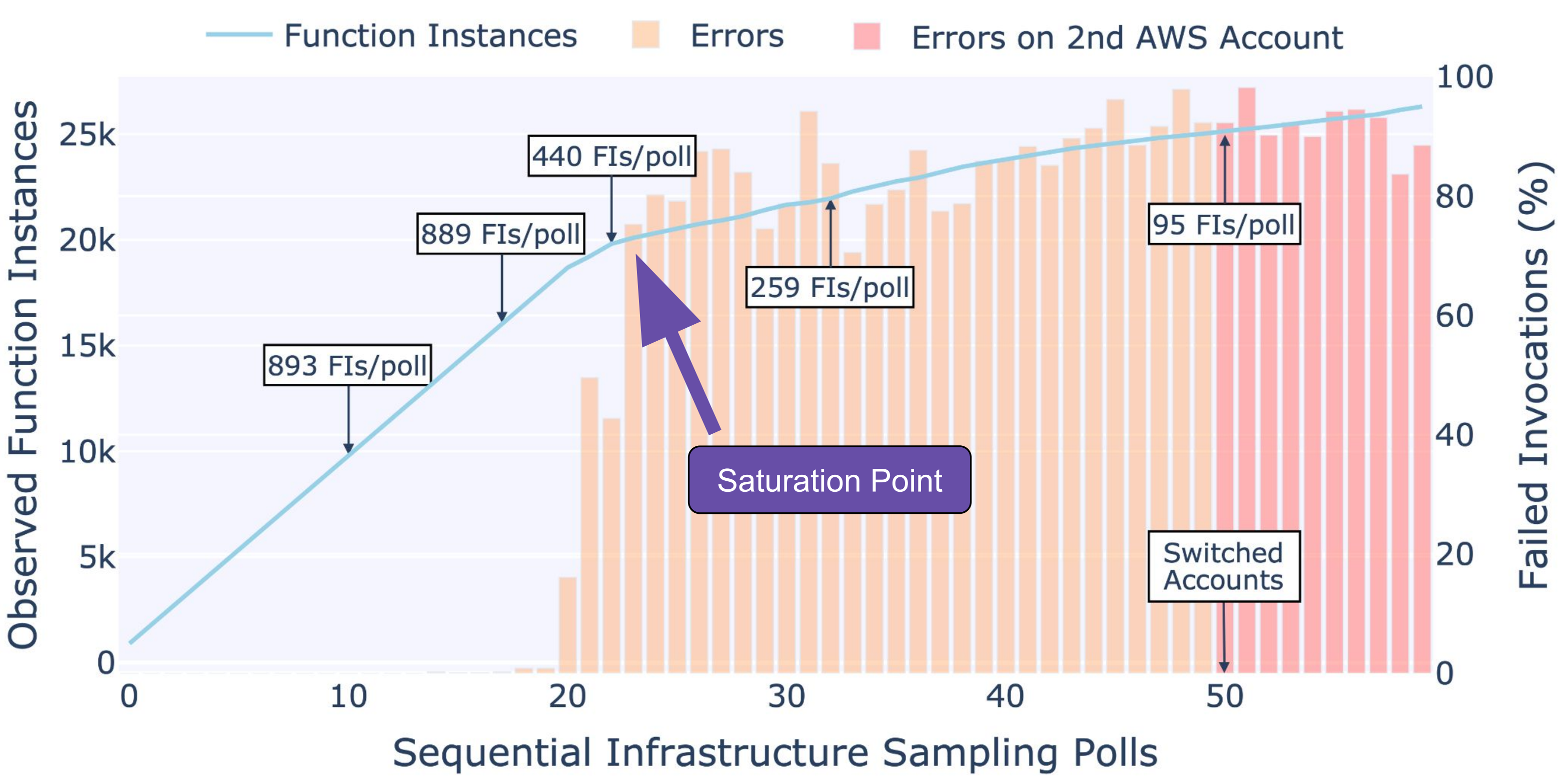


Function Instances
Errors
Observed Function Instances
Failed Invocations (%)
Sequential Infrastructure Sampling Polls
893 FIs/poll
889 FIs/poll
440 FIs/poll
259 FIs/poll
95 FIs/poll

28

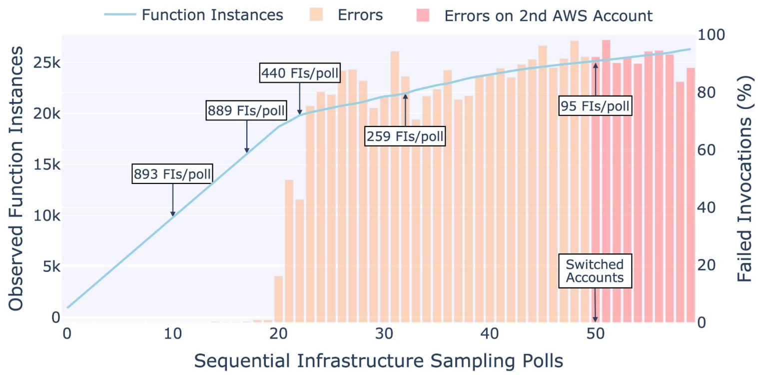


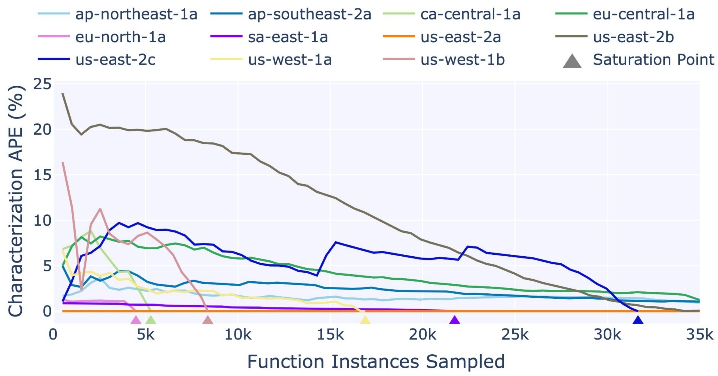
Function Instances
Errors
Errors on 2nd AWS Account
Observed Function Instances
Failed Invocations (%)
Sequential Infrastructure Sampling Polls
893 FIs/poll
889 FIs/poll
440 FIs/poll
259 FIs/poll
95 FIs/poll
Switched Accounts



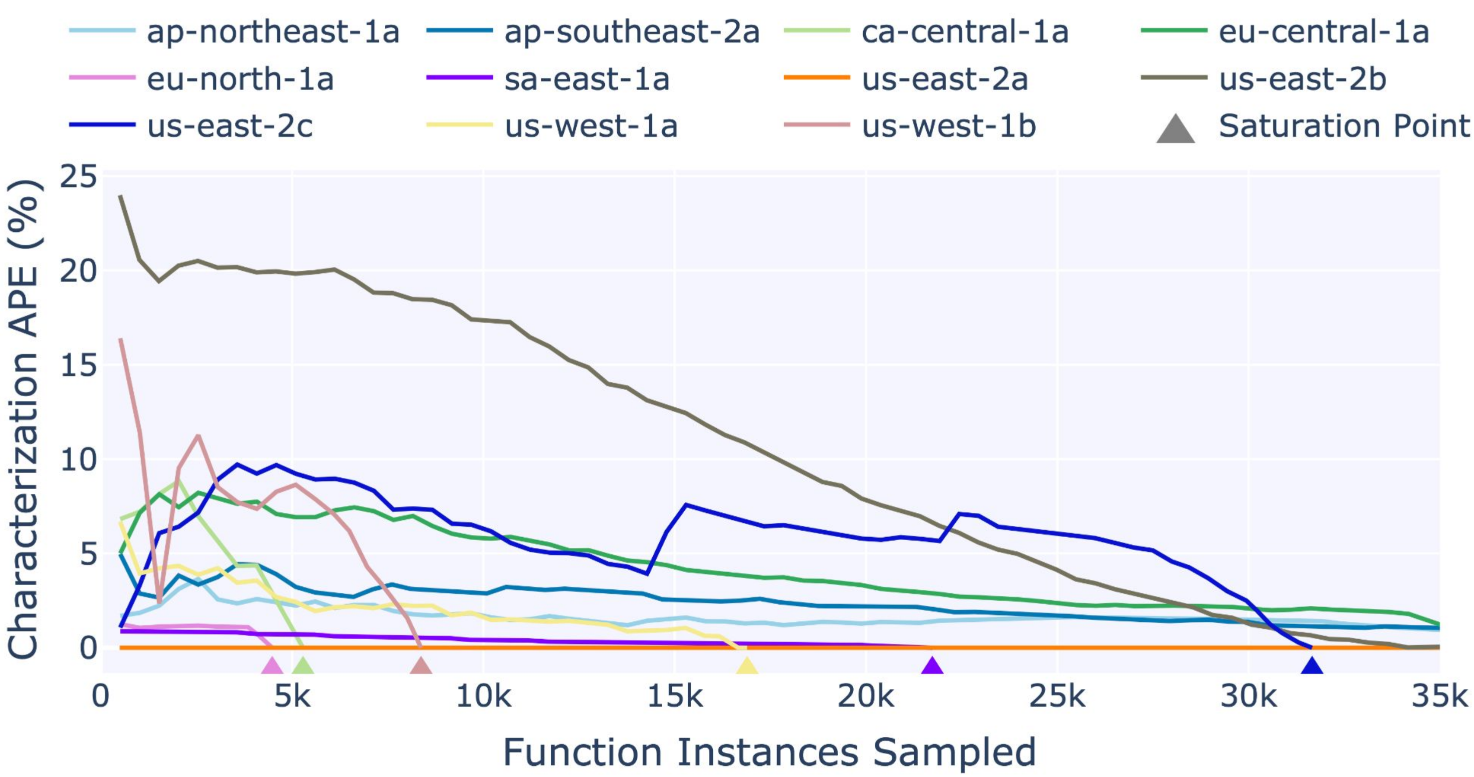
Function Instances
Errors
Errors on 2nd AWS Account
Observed Function Instances
Failed Invocations (%)
Sequential Infrastructure Sampling Polls
893 FIs/poll
889 FIs/poll
440 FIs/poll
259 FIs/poll
95 FIs/poll
Saturation Point
Switched Accounts

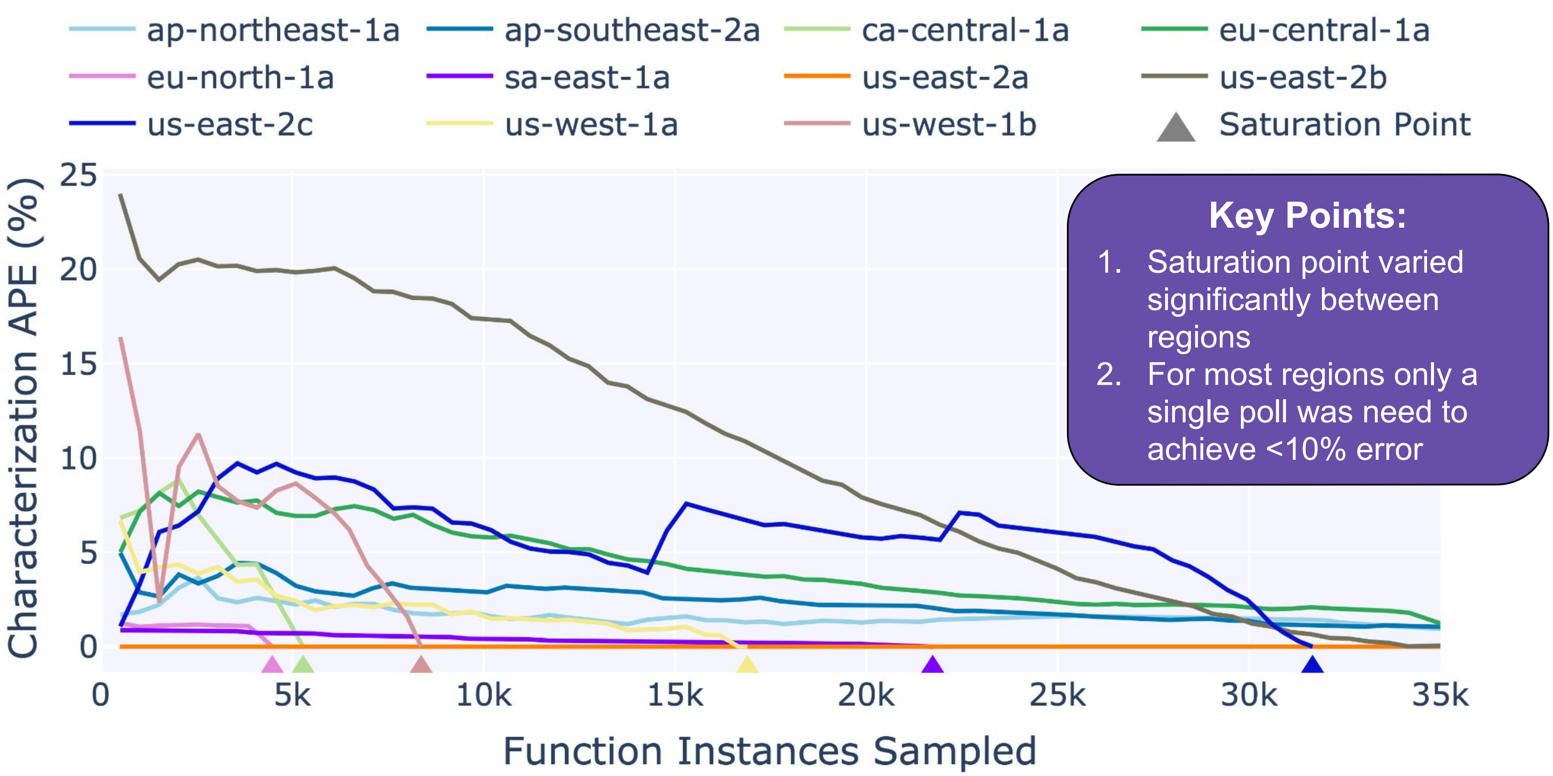
Sanity Checks

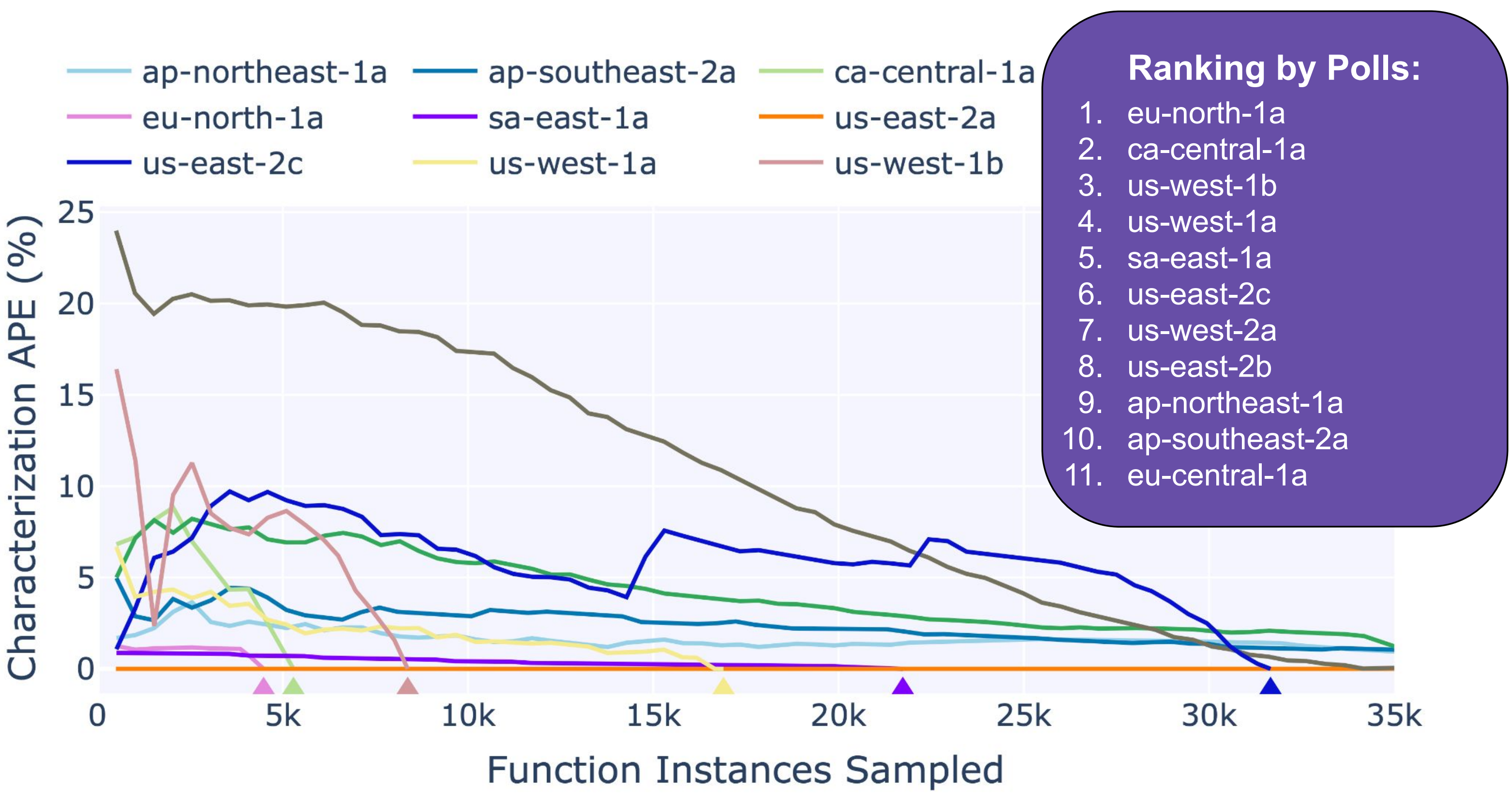




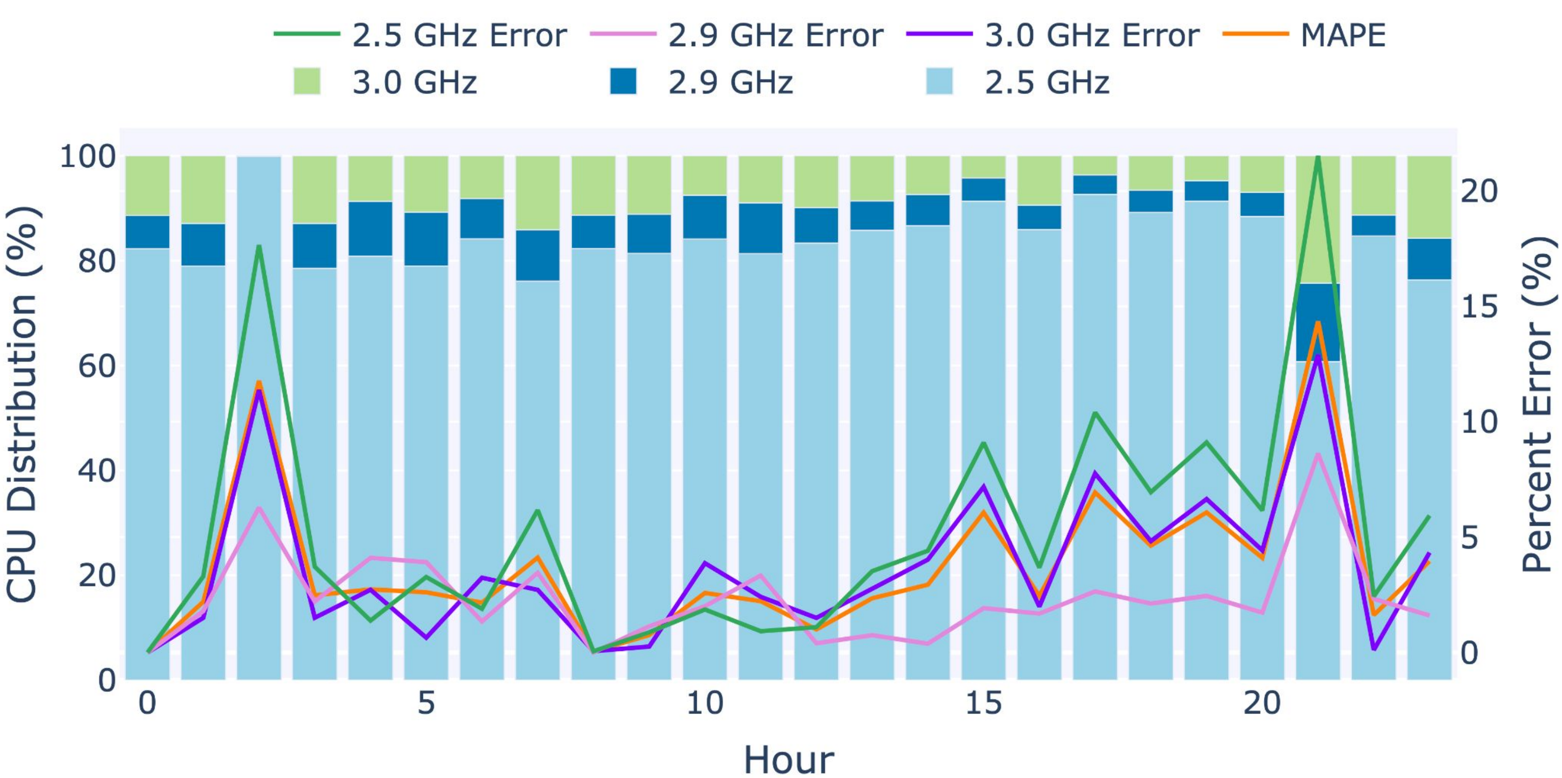
- To verify this was not a simple rate limit we:
  - used separate AWS accounts with:
    - different email addresses
    - different payment methods
    - different locations/network
    - different deployed functions
    - different FaaS configurations (checksum/RAM)
    - different invocation methods:
      - Function URLs
      - API Gateway
      - AWS CLI
- The only similarity was functions shared an availability zone
- Even with all of these tests we still observed these errors




ap-northeast-1a
ap-southeast-2a
ca-central-1a
eu-central-1a
eu-north-1a
sa-east-1a
us-east-2a
us-east-2b
us-east-2c
us-west-1a
us-west-1b
Saturation Point
Characterization APE (%)
Function Instances Sampled
0
5k
10k
15k
20k
25k
30k
35k
0
5
10
15
20
25
Key Points:
1. Saturation point varied significantly between regions
2. For most regions only a single poll was need to achieve <10% error


ap-northeast-1a
ap-southeast-2a
ca-central-1a
eu-north-1a
sa-east-1a
us-east-2a
us-east-2c
us-west-1a
us-west-1b
Characterization APE (%)
Function Instances Sampled
0
5k
10k
15k
20k
25k
30k
35k
0
5
10
15
20
25
Ranking by Polls:
1. eu-north-1a
2. ca-central-1a
3. us-west-1b
4. us-west-1a
5. sa-east-1a
6. us-east-2c
7. us-west-2a
8. us-east-2b
9. ap-northeast-1a
10. ap-southeast-2a
11. eu-central-1a

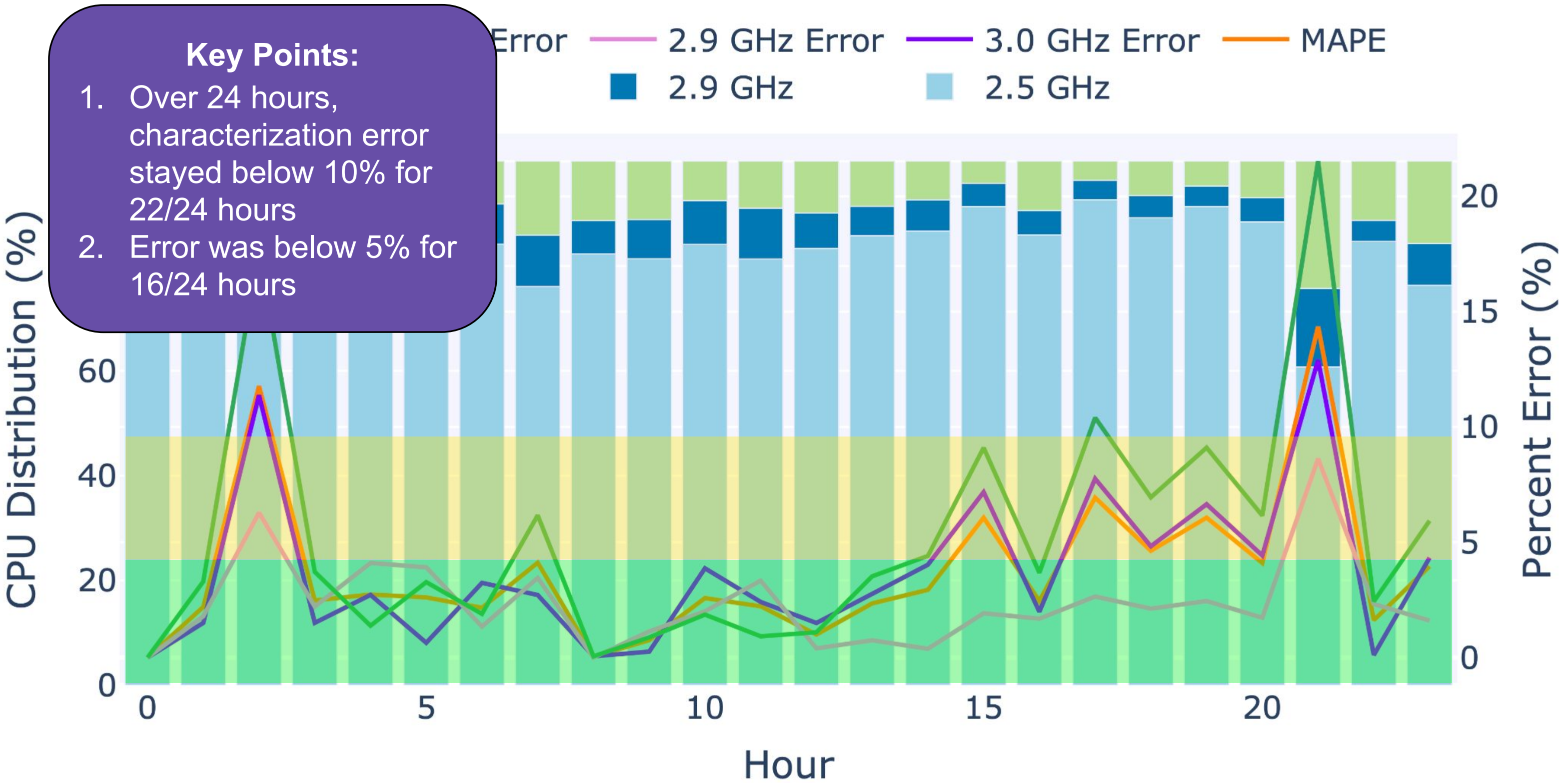
RQ-1 (Part 2): How does characterization accuracy change over time?



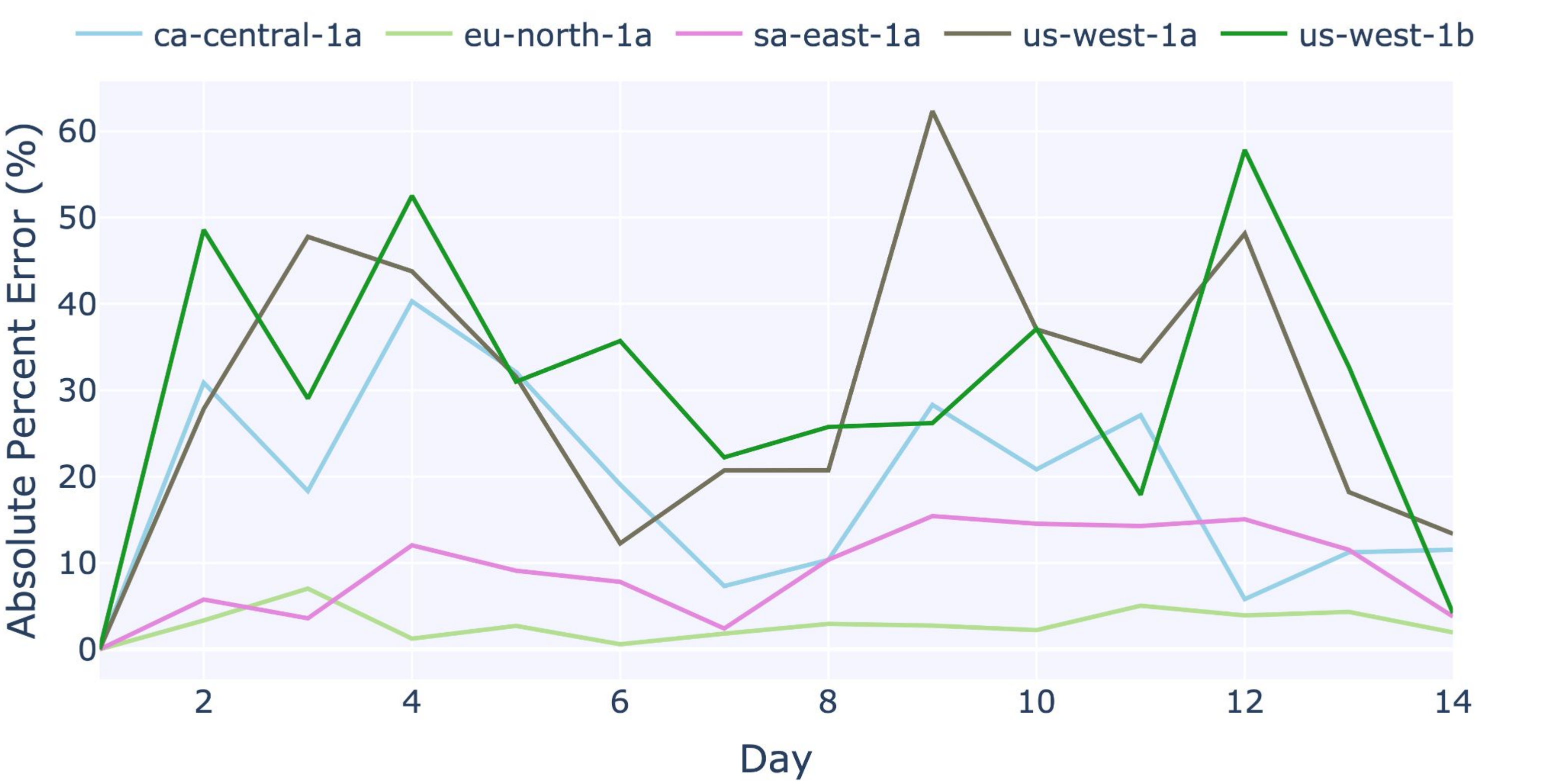
- Characterization accuracy over 24 hours on us-west-1b

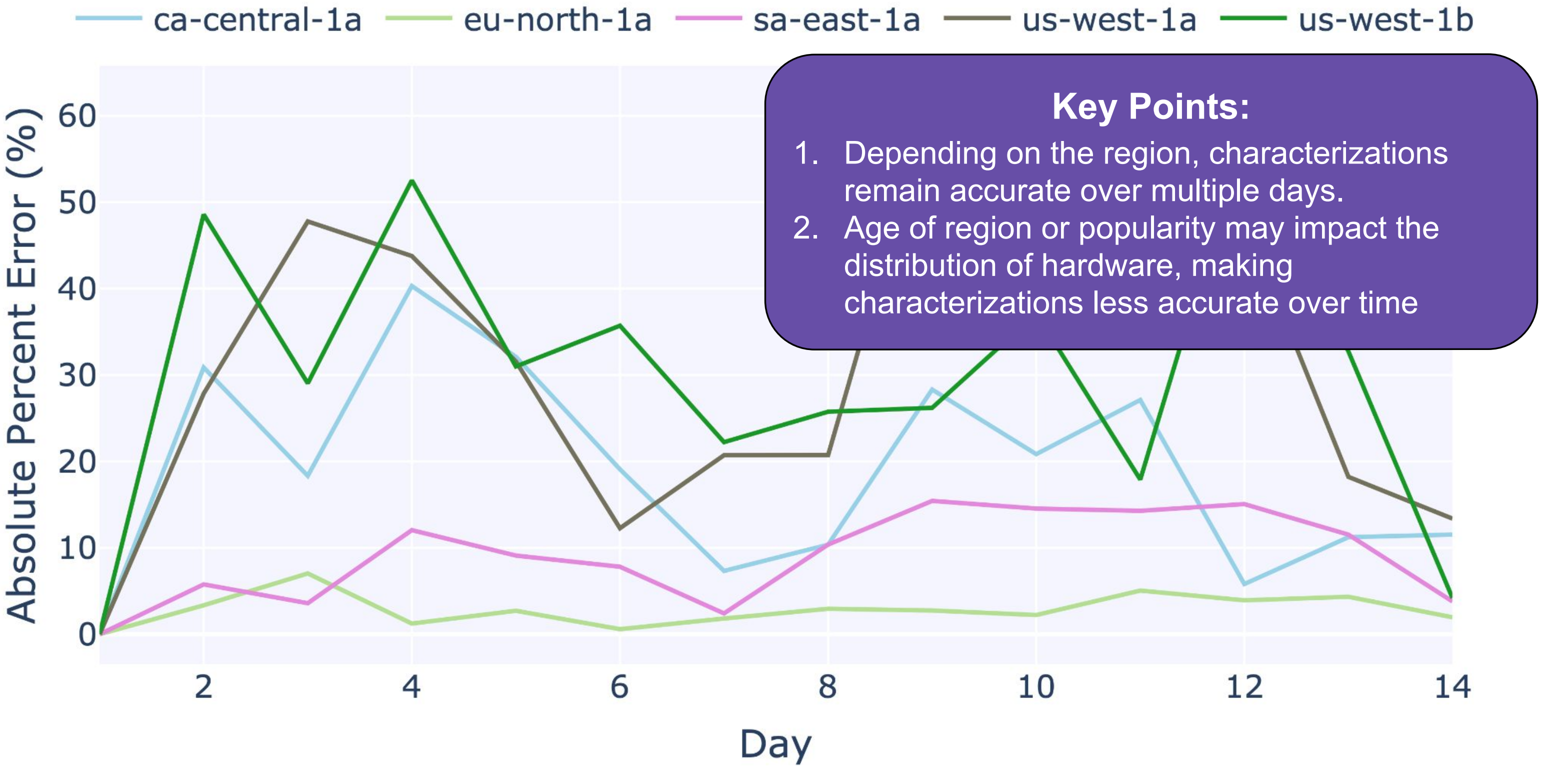
Key Points:

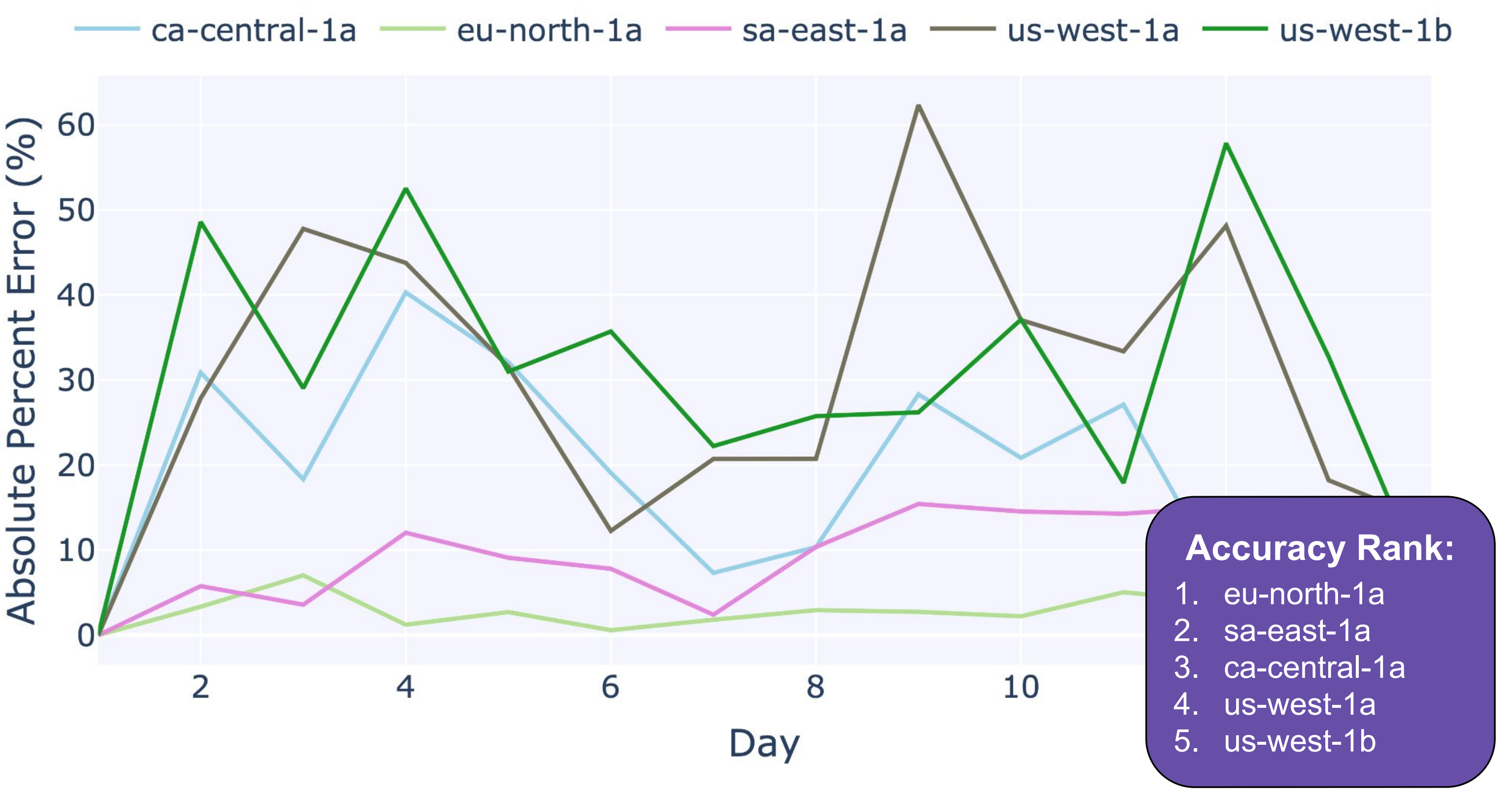
1. Over 24 hours, characterization error stayed below 10% for 22/24 hours
2. Error was below 5% for 16/24 hours



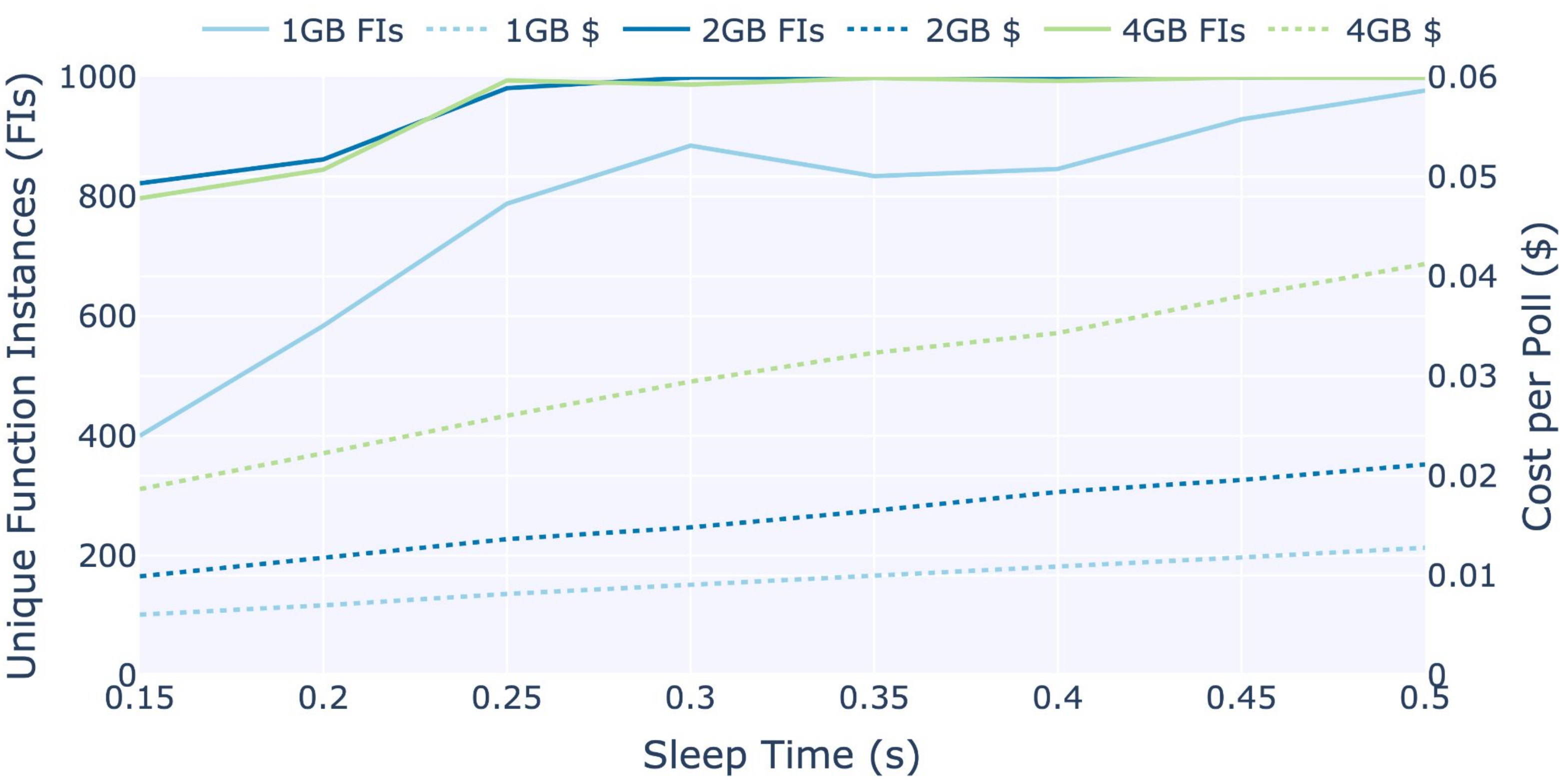
- Characterization accuracy over 24 hours on us-west-1b

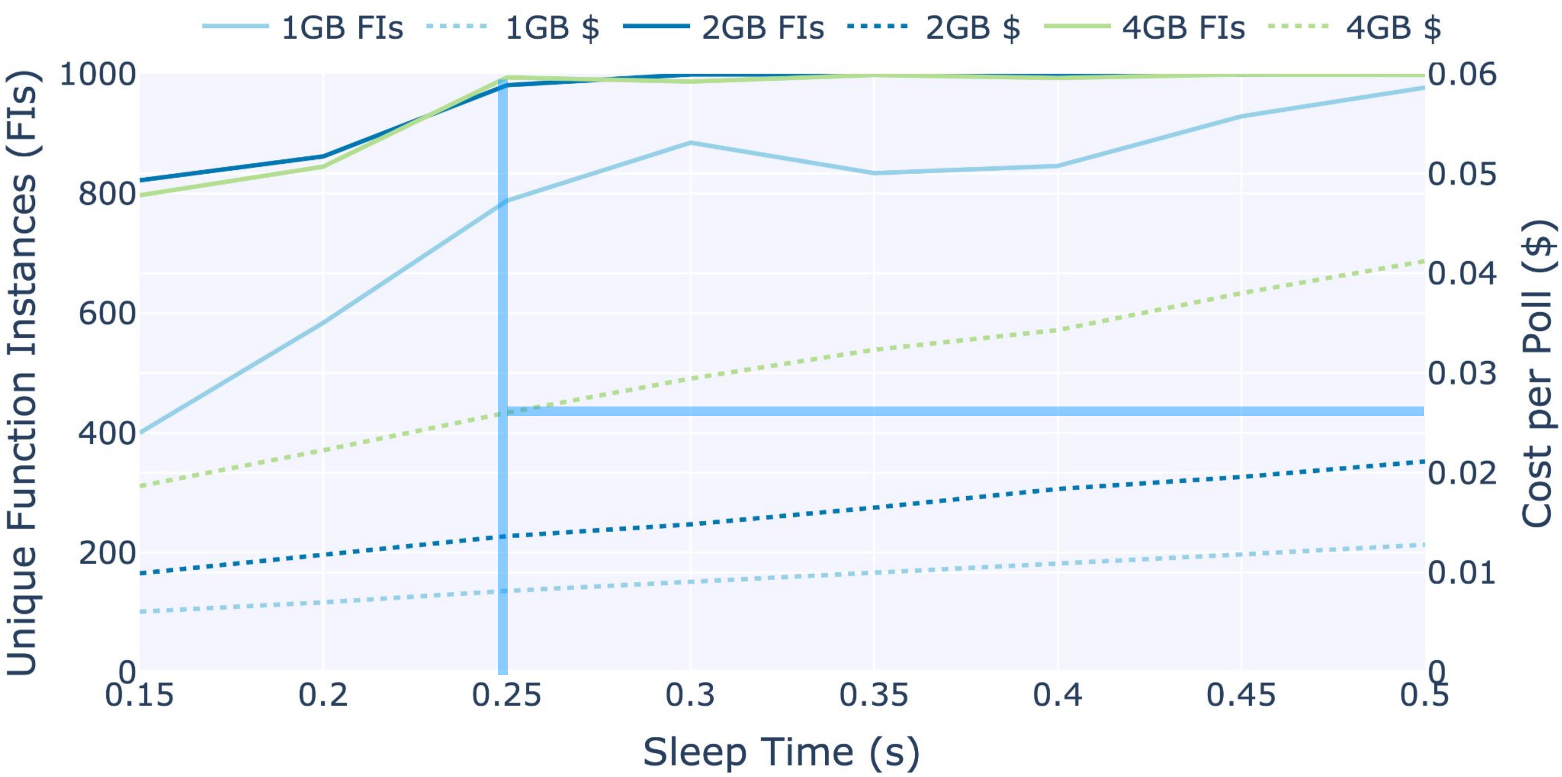


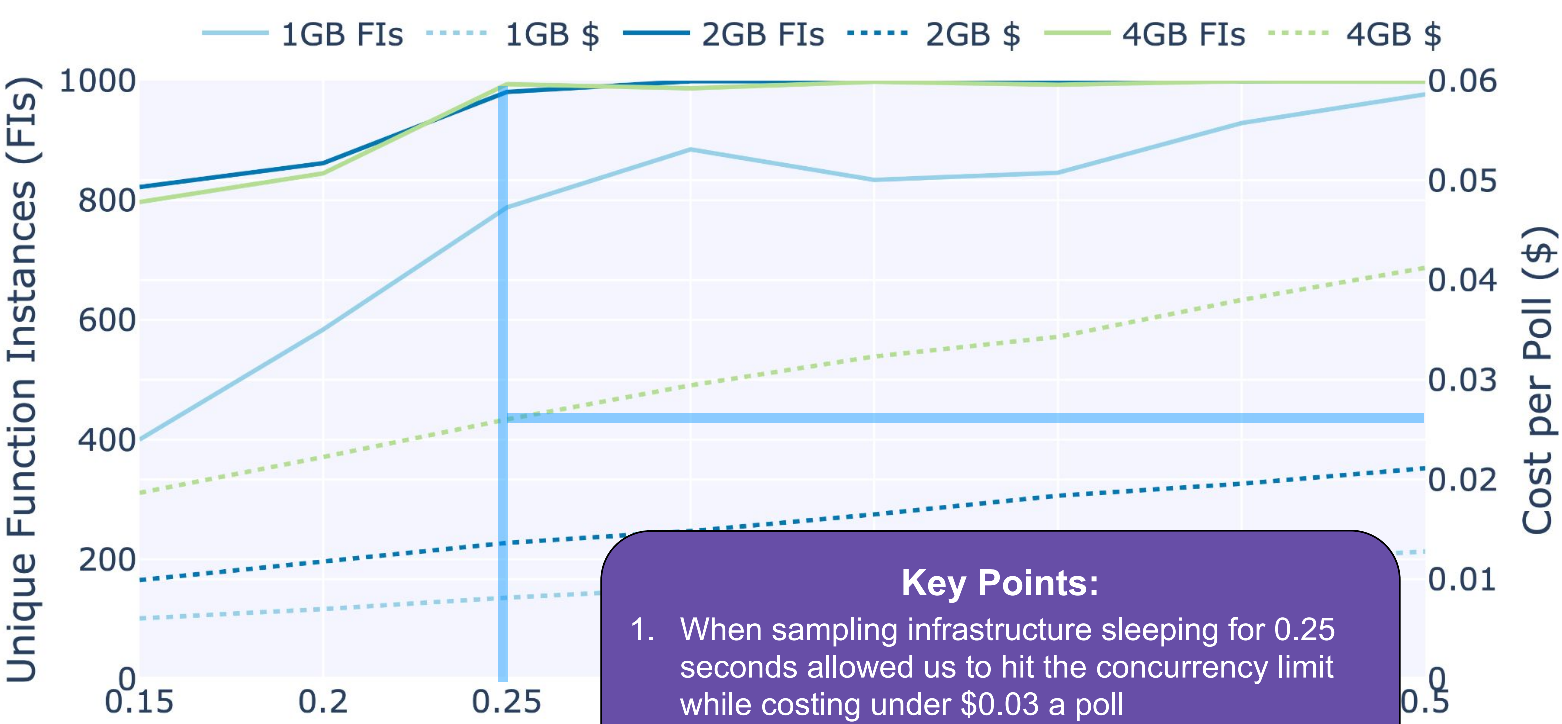

ca-central-1a
eu-north-1a
sa-east-1a
us-west-1a
us-west-1b
Absolute Percent Error (%)
Day
Key Points:
1. Depending on the region, characterizations remain accurate over multiple days.
2. Age of region or popularity may impact the distribution of hardware, making characterizations less accurate over time


ca-central-1a
eu-north-1a
sa-east-1a
us-west-1a
us-west-1b
Absolute Percent Error (%)
Day
Accuracy Rank:
1. eu-north-1a
2. sa-east-1a
3. ca-central-1a
4. us-west-1a
5. us-west-1b

RQ-2: How many samples are required to accurately infer the hardware pool? How can we balance the cost versus accuracy?

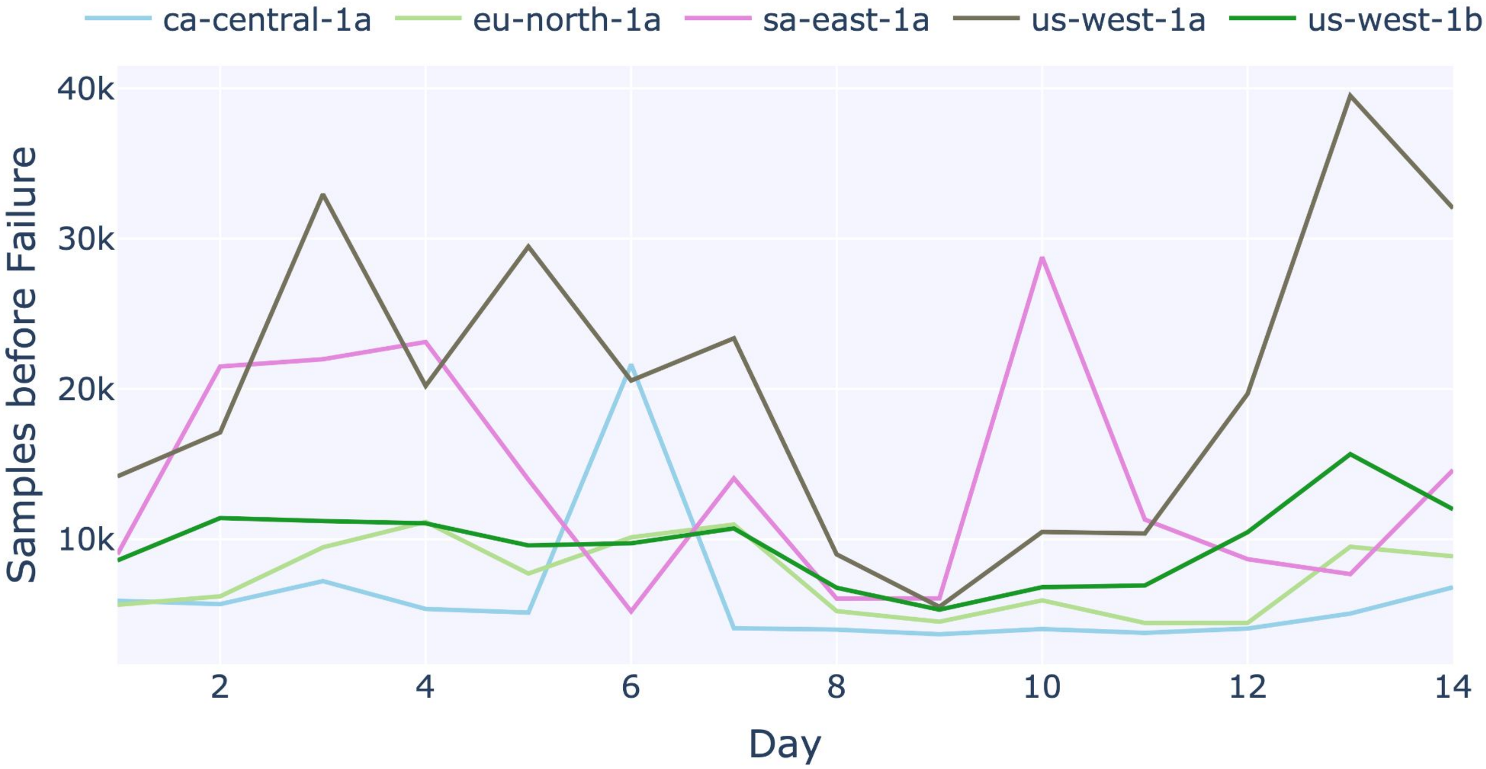




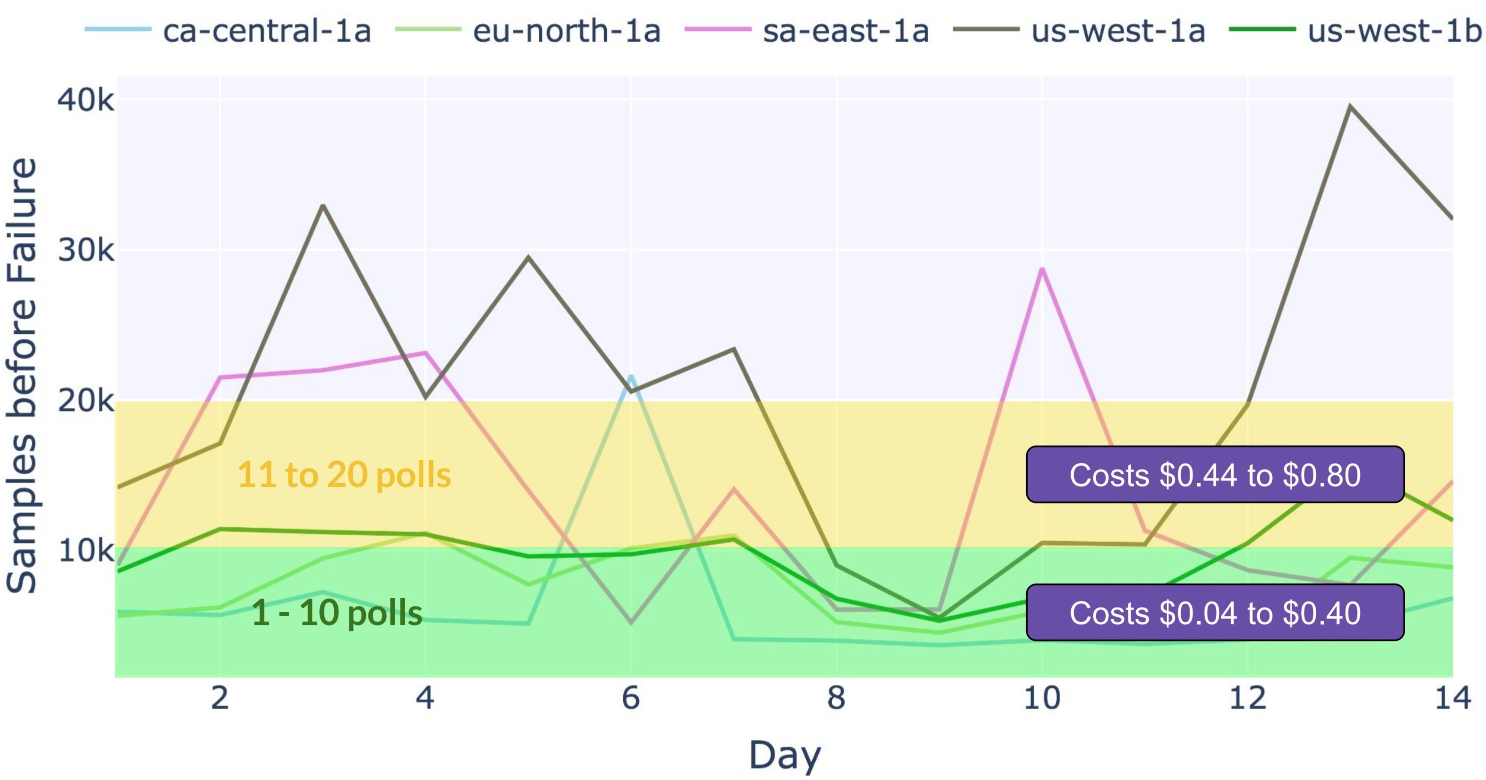


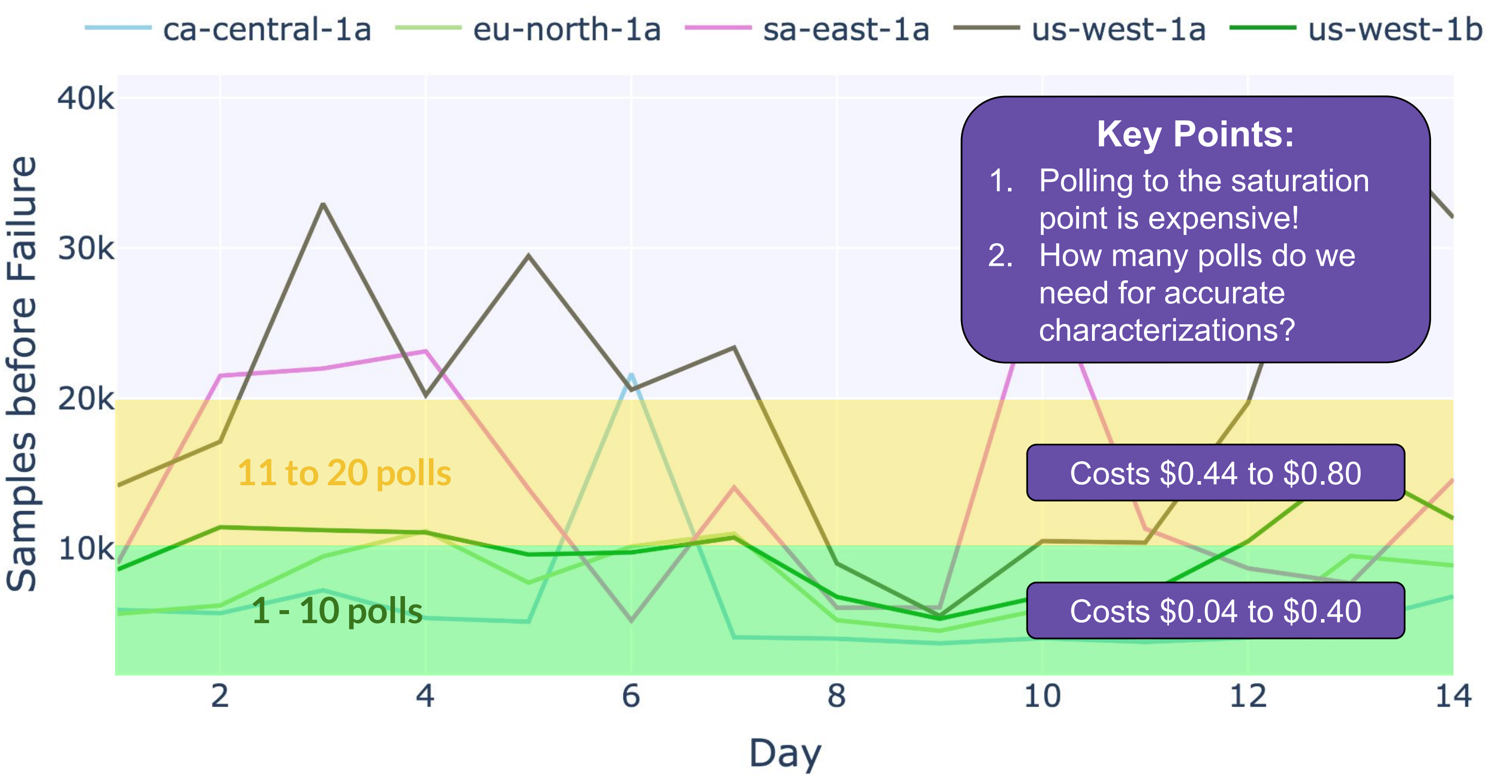
Key Points:

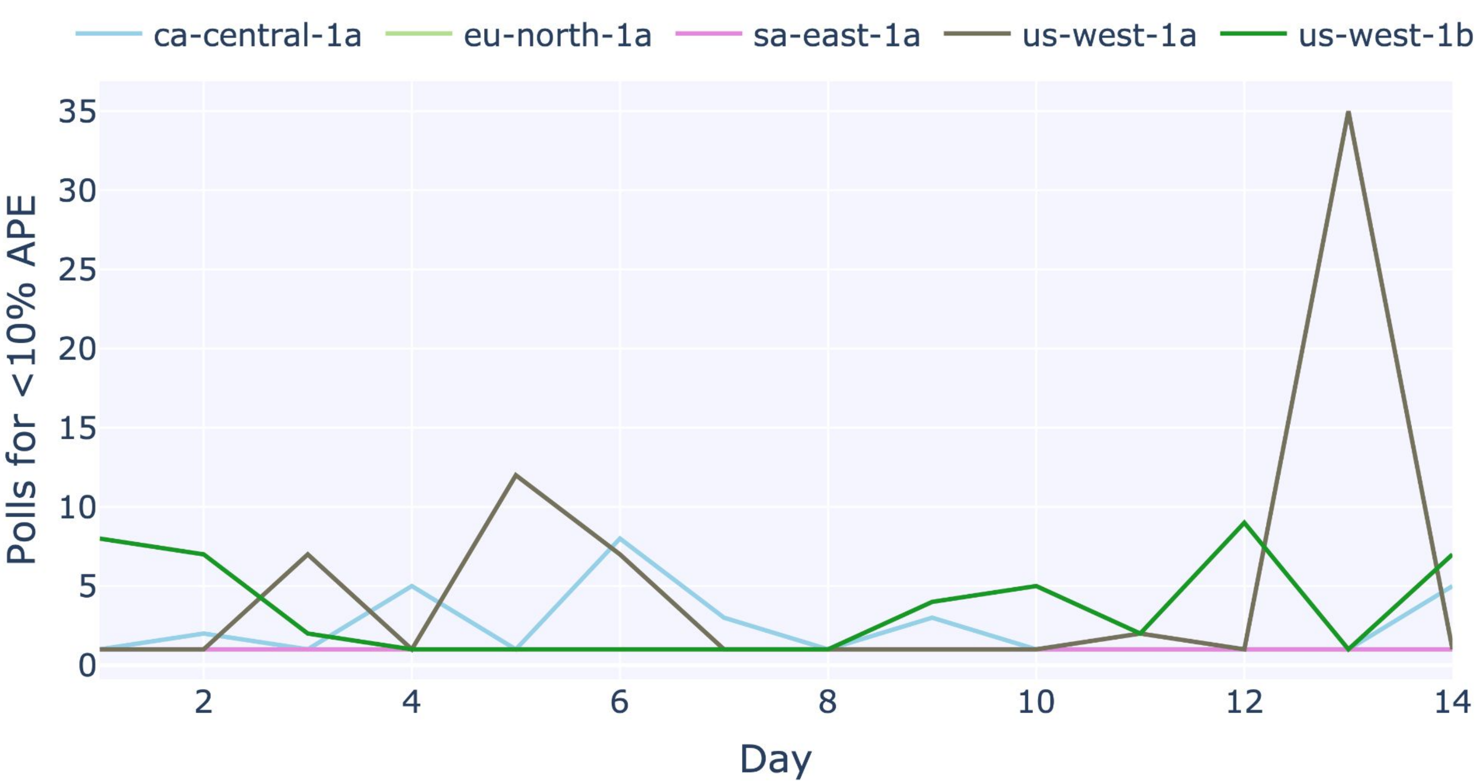
1. When sampling infrastructure sleeping for 0.25 seconds allowed us to hit the concurrency limit while costing under $0.03 a poll
2. For later experiments we used 10 GBs memory settings, resulting in ~$0.04 cost per poll

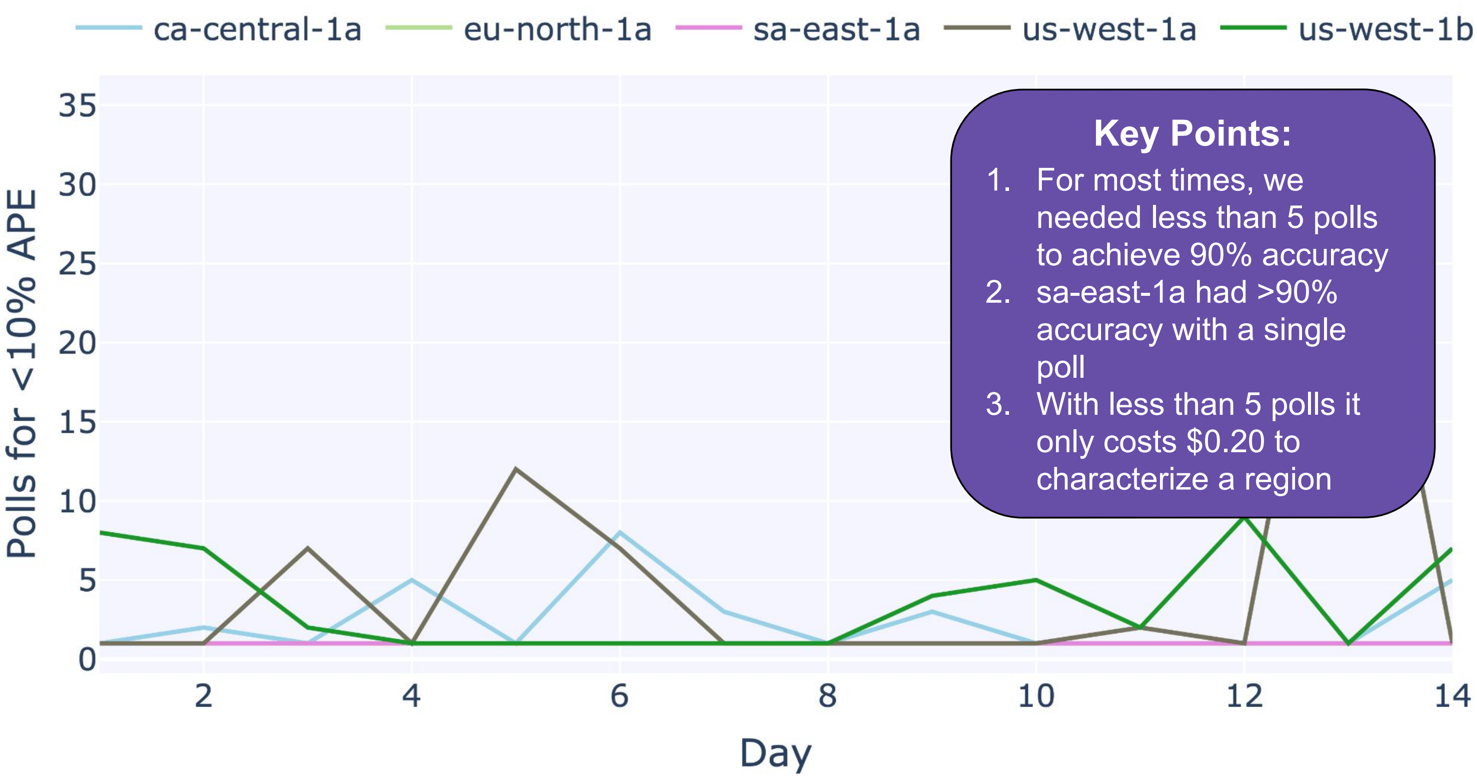

ca-central-1a
eu-north-1a
sa-east-1a
us-west-1a
us-west-1b
Samples before Failure
40k
30k
20k
10k
2
4
6
8
10
12
14
Day

45

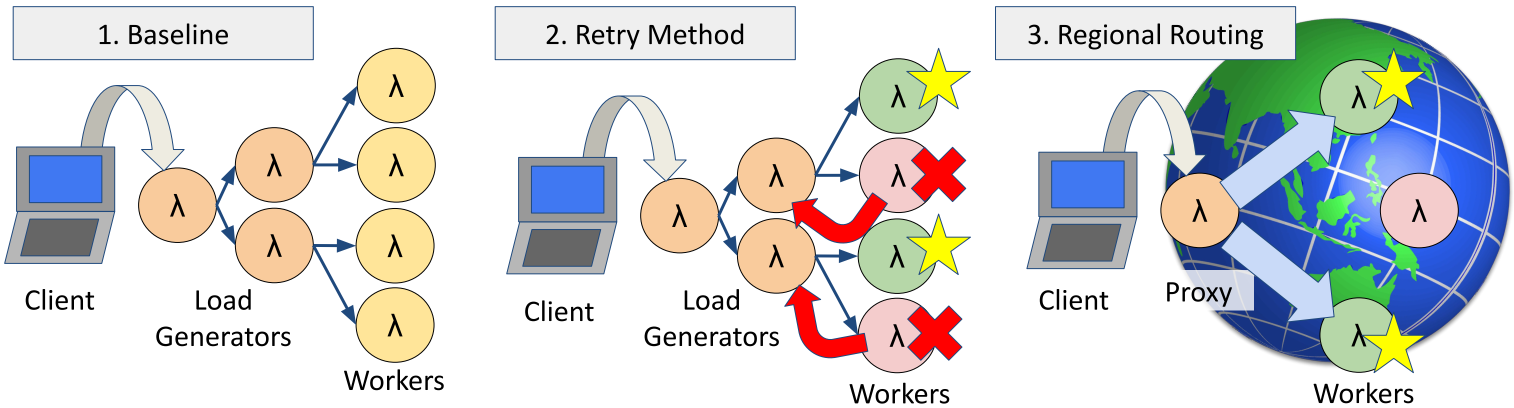

ca-central-1a
eu-north-1a
sa-east-1a
us-west-1a
us-west-1b
Samples before Failure
40k
30k
20k
10k
11 to 20 polls
Costs $0.44 to $0.80
1 - 10 polls
Costs $0.04 to $0.40
2
4
6
8
10
12
14
Day


ca-central-1a
eu-north-1a
sa-east-1a
us-west-1a
us-west-1b
Key Points:
1. Polling to the saturation point is expensive!
2. How many polls do we need for accurate characterizations?
11 to 20 polls
Costs $0.44 to $0.80
1 - 10 polls
Costs $0.04 to $0.40
Samples before Failure
40k
30k
20k
10k
2
4
6
8
10
12
14
Day


ca-central-1a
eu-north-1a
sa-east-1a
us-west-1a
us-west-1b
Polls for <10% APE
35
30
25
20
15
10
5
0
2
4
6
8
10
12
14
Day



RQ-3: To what extent are runtime and cost improvements possible by targeting specific instructure?

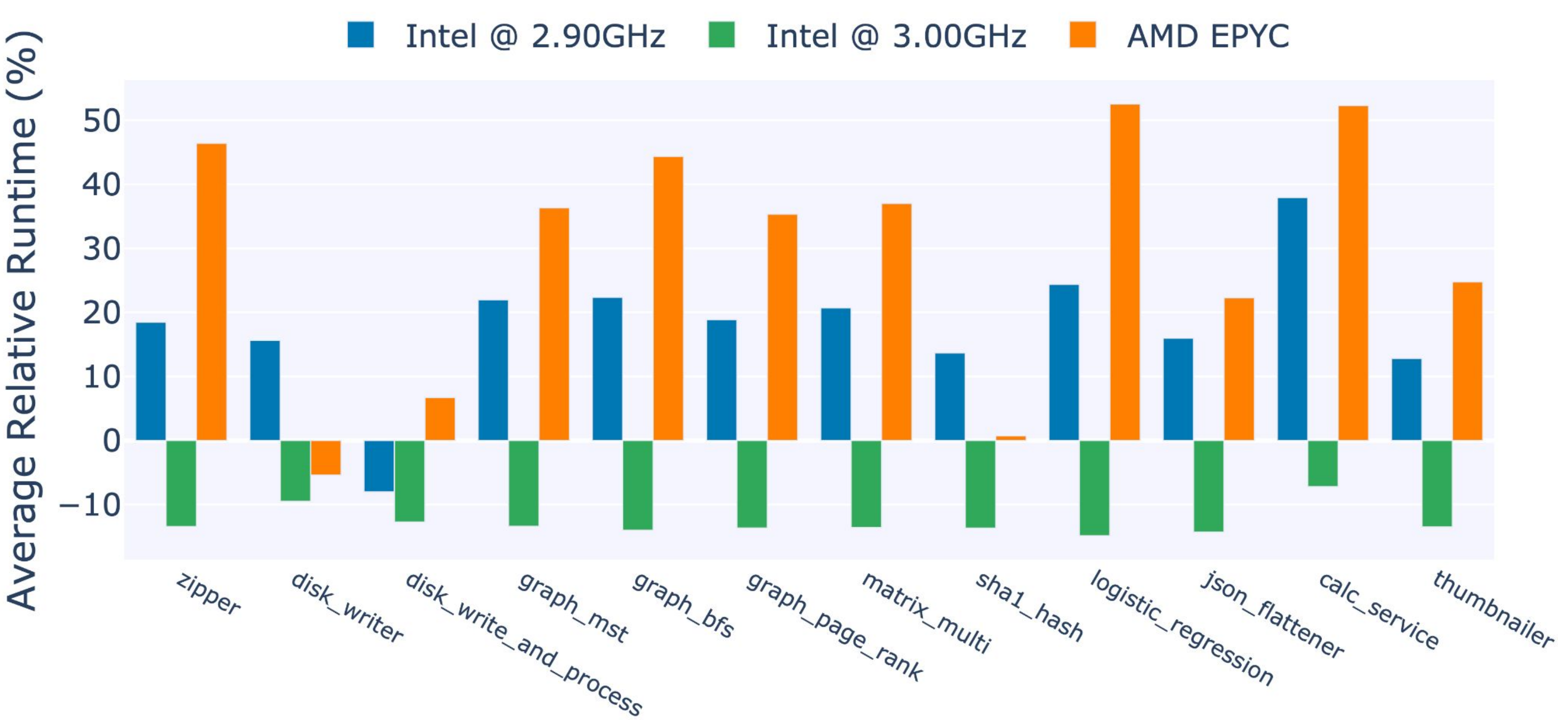


Retry Logic

1. **Retry Slow:** Invocations retry on the two slowest CPUs
2. **Focus Fastest:** Invocations retry on everything but the fastest CPU

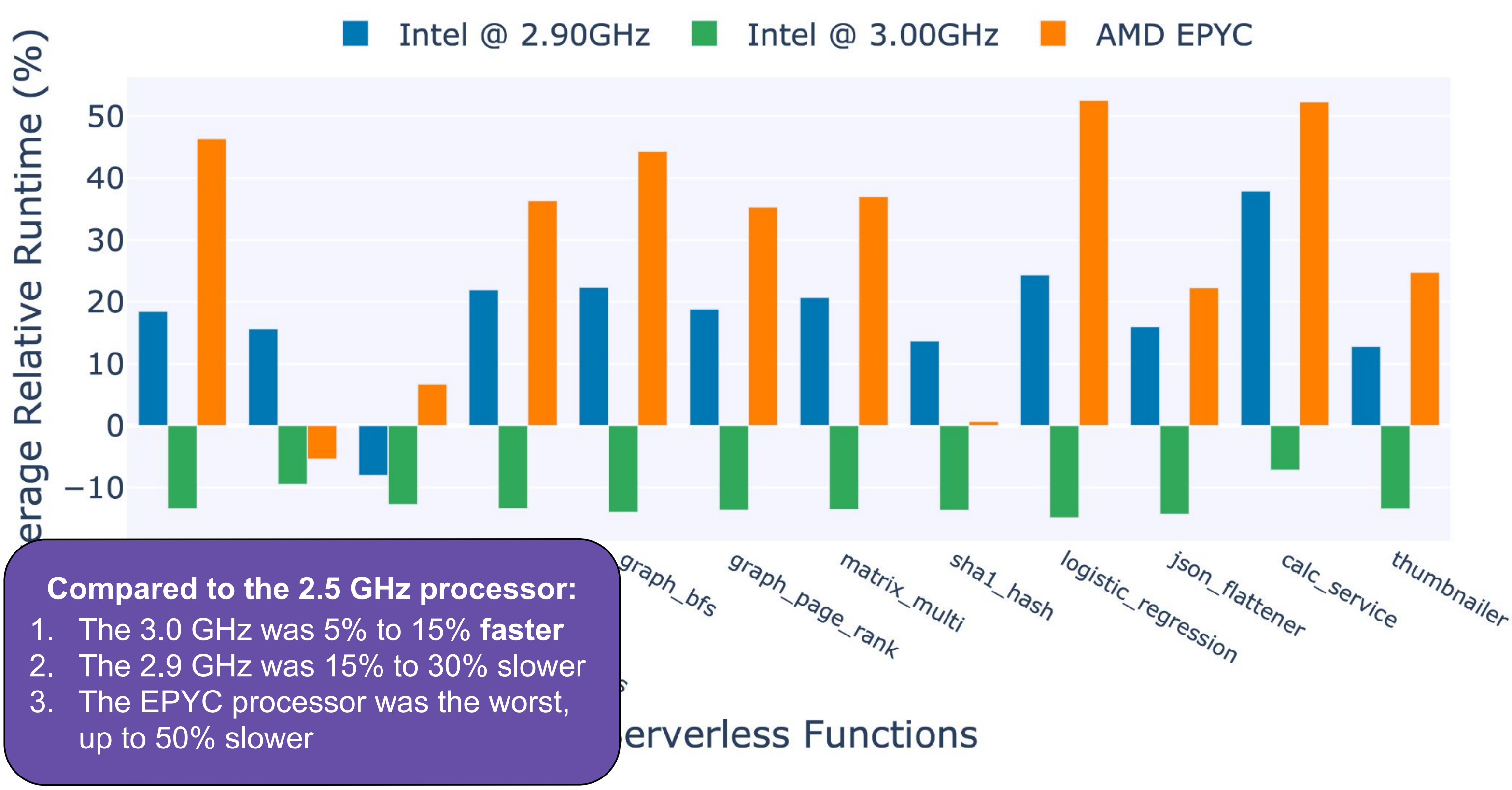
Hybrid Approach

Combine both Retries and Regional Routing

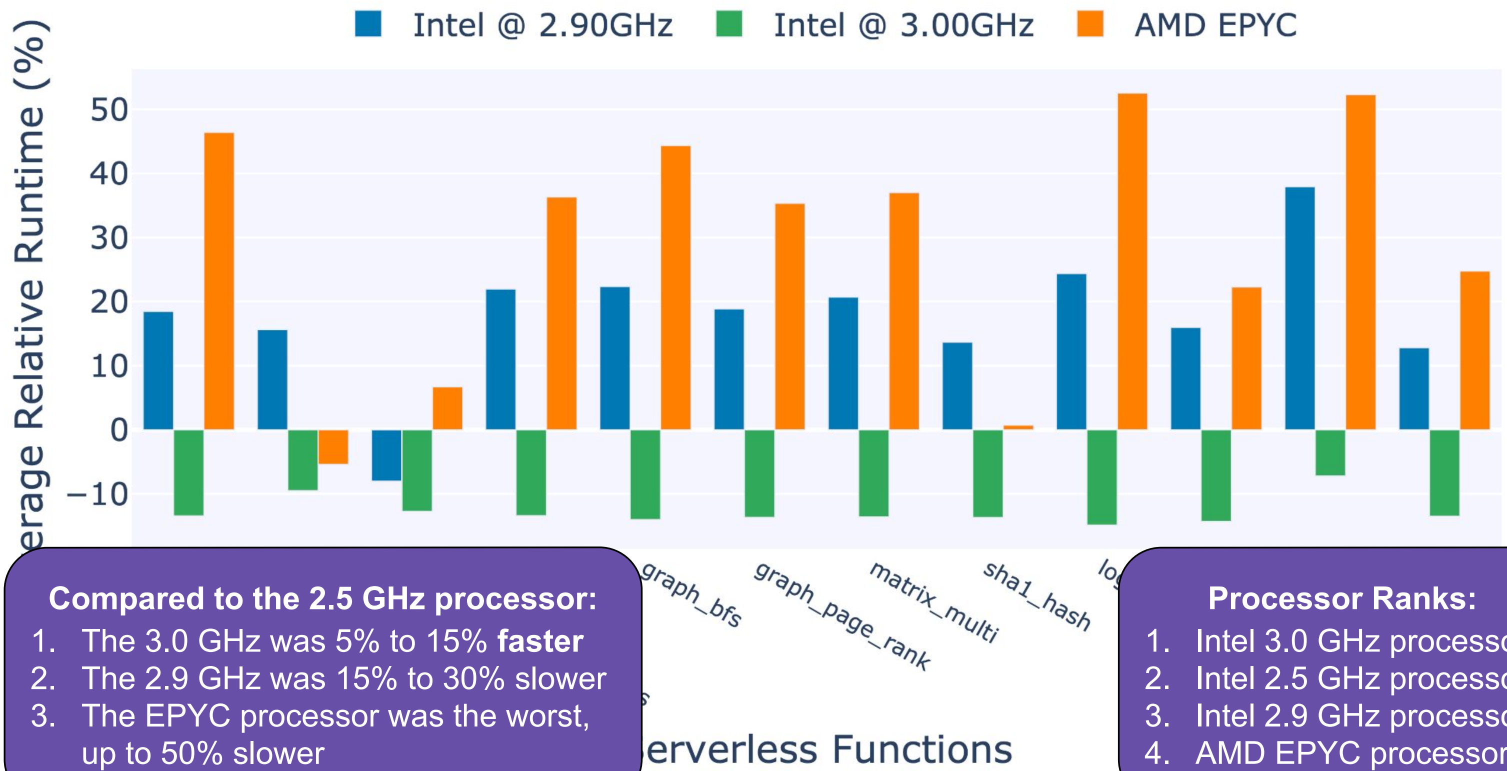


- Runtime relative to the most common Intel 2.5 GHz processor observed on AWS Lambda

- Runtime relative to the most common Intel 2.5 GHz processor observed on AWS Lambda



- Runtime relative to the most common Intel 2.5 GHz processor observed on AWS Lambda



Compared to the 2.5 GHz processor:

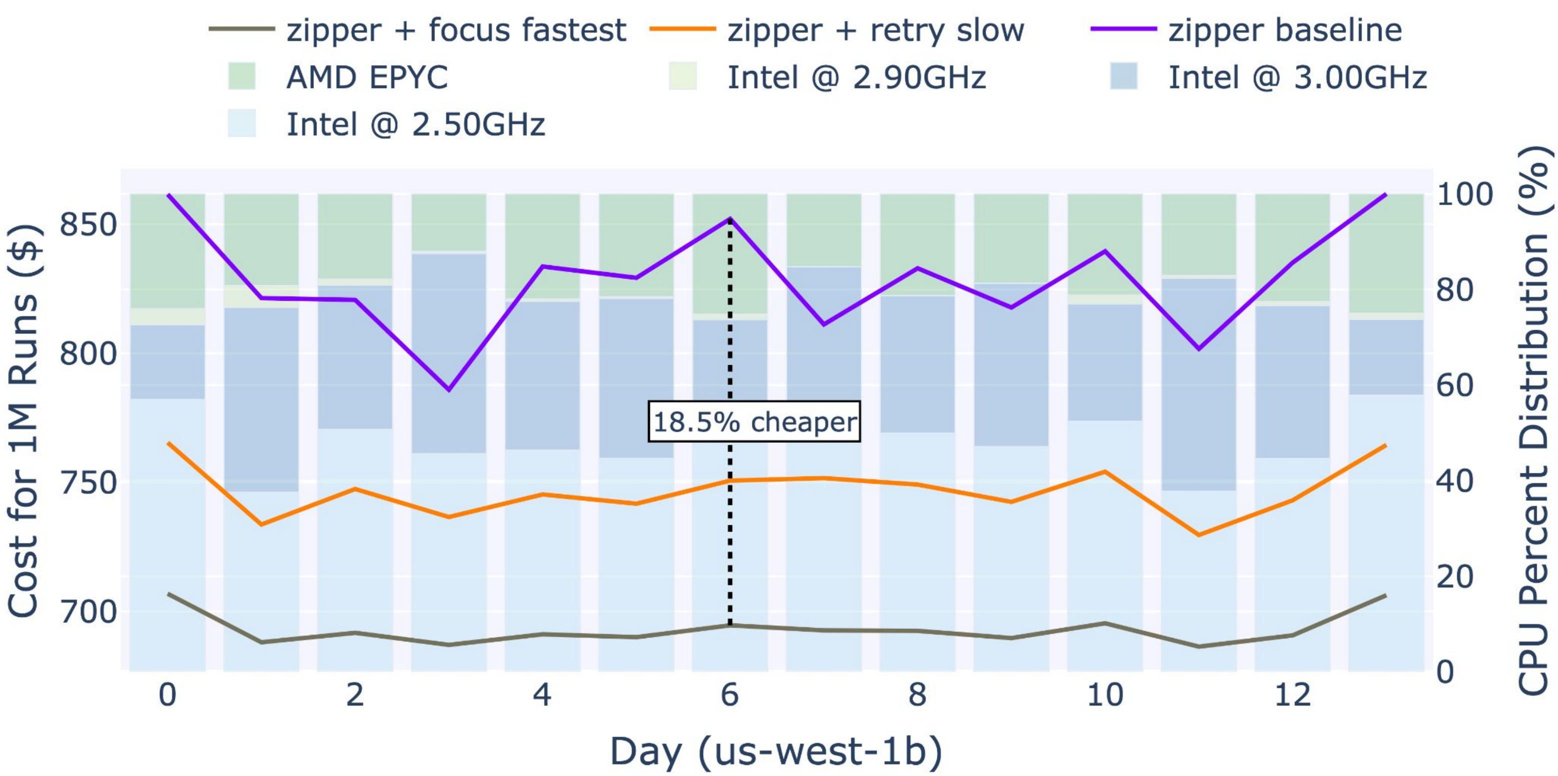
1. The 3.0 GHz was 5% to 15% **faster**
2. The 2.9 GHz was 15% to 30% slower
3. The EPYC processor was the worst, up to 50% slower

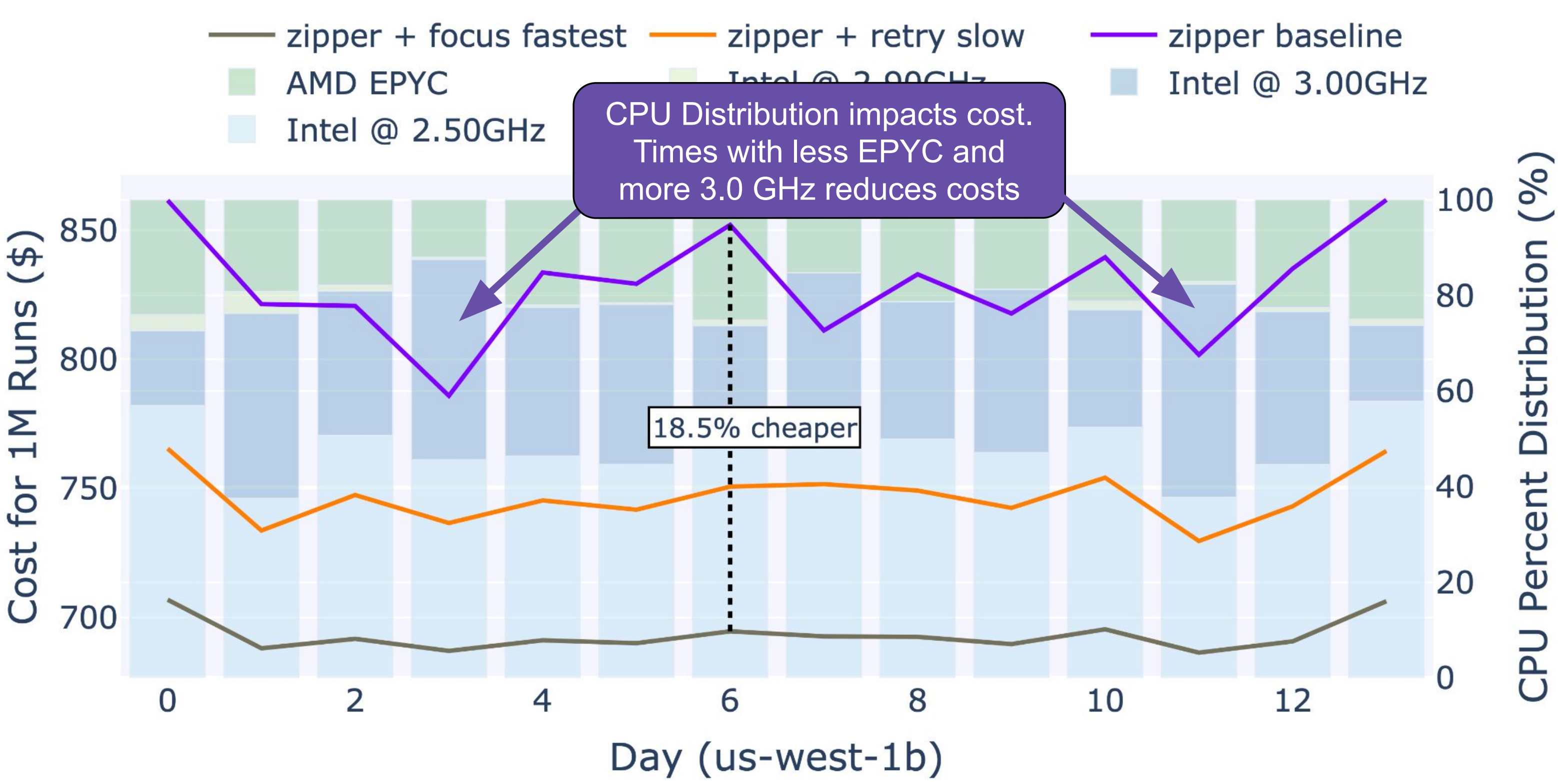
Processor Ranks:

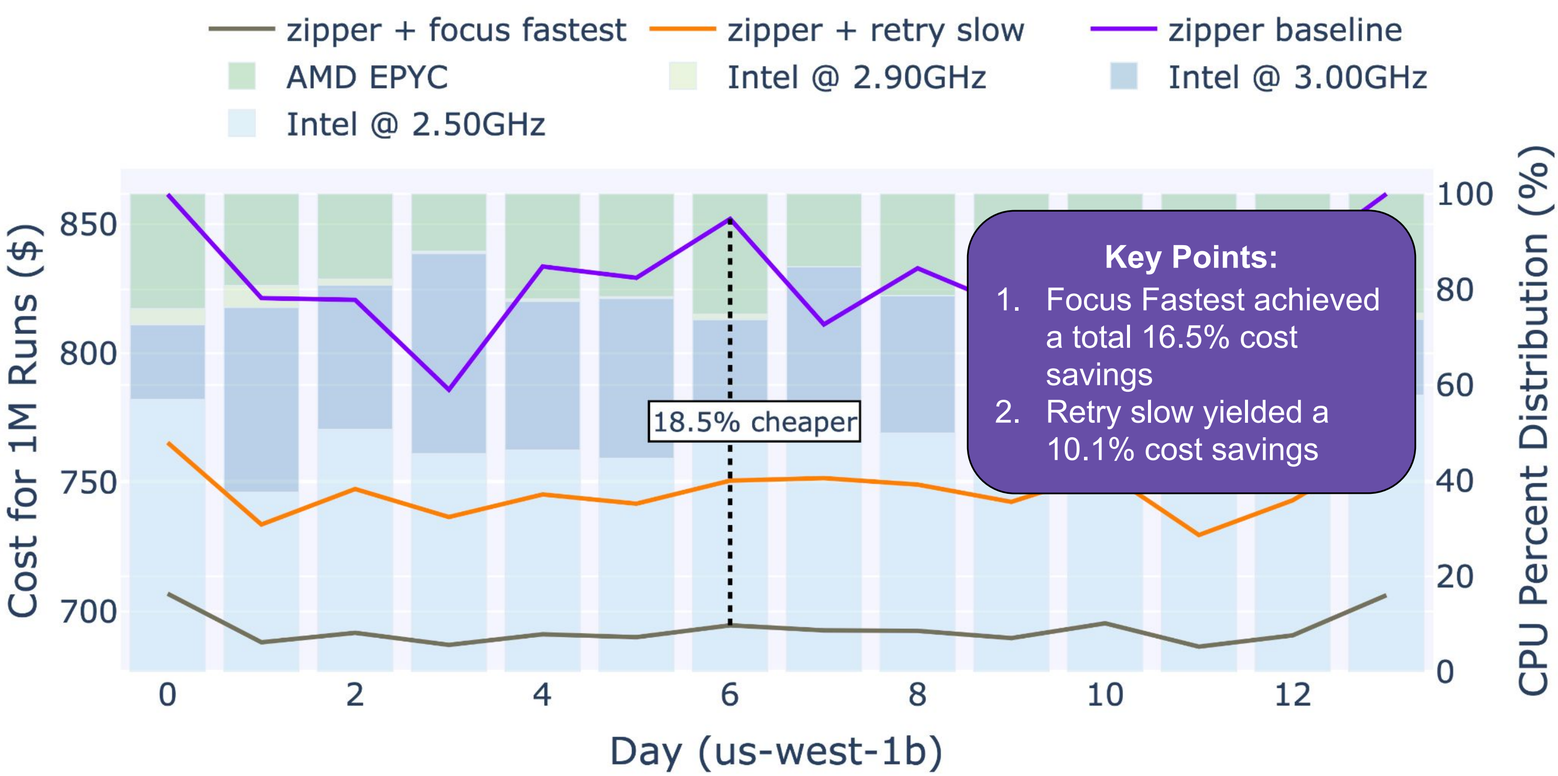
1. Intel 3.0 GHz processor
2. Intel 2.5 GHz processor
3. Intel 2.9 GHz processor
4. AMD EPYC processor

- Runtime relative to the most common Intel 2.5 GHz processor observed on AWS Lambda

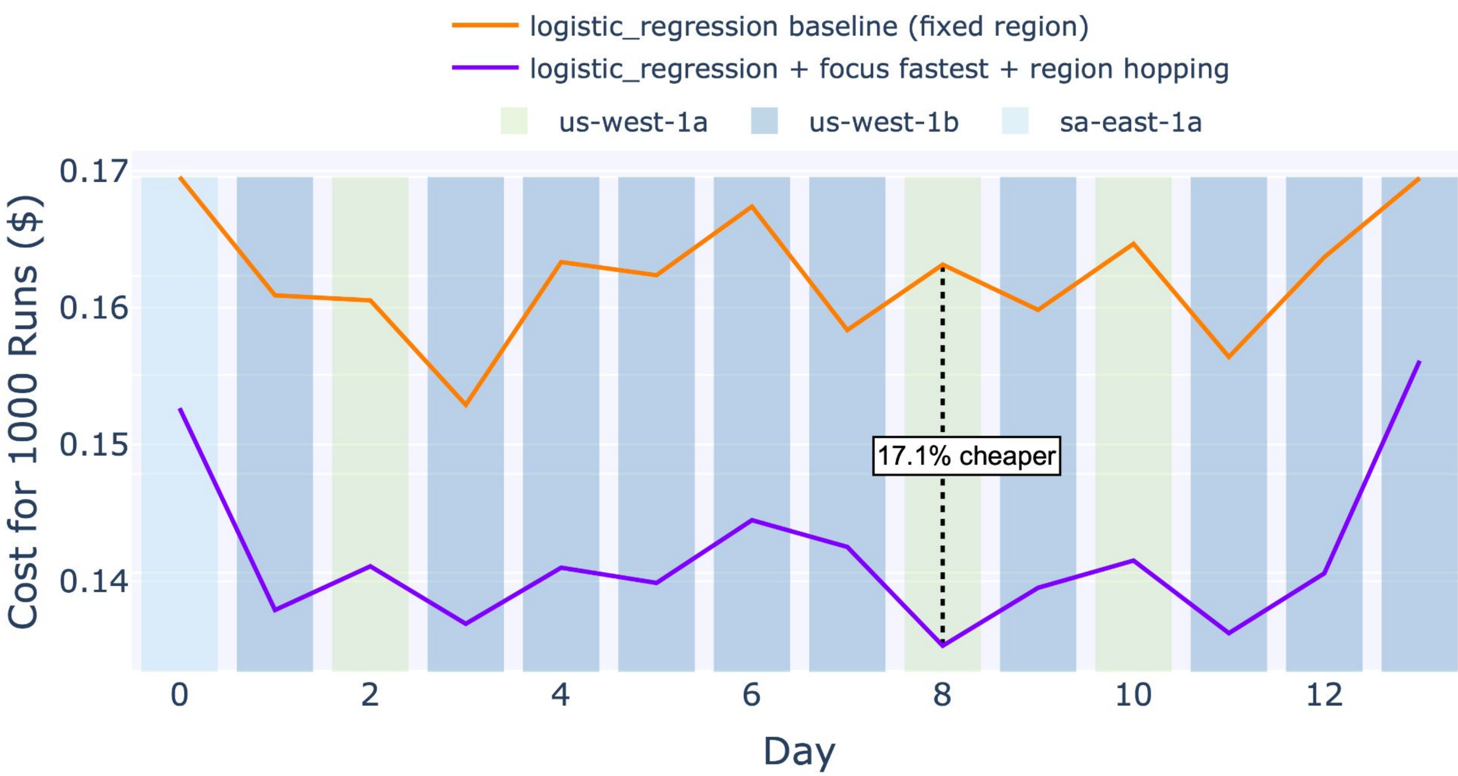
Retry Methods


zipper + focus fastest
zipper + retry slow
zipper baseline
AMD EPYC
Intel @ 2.90GHz
Intel @ 3.00GHz
Intel @ 2.50GHz
18.5% cheaper
Cost for 1M Runs ($)
CPU Percent Distribution (%)
Day (us-west-1b)

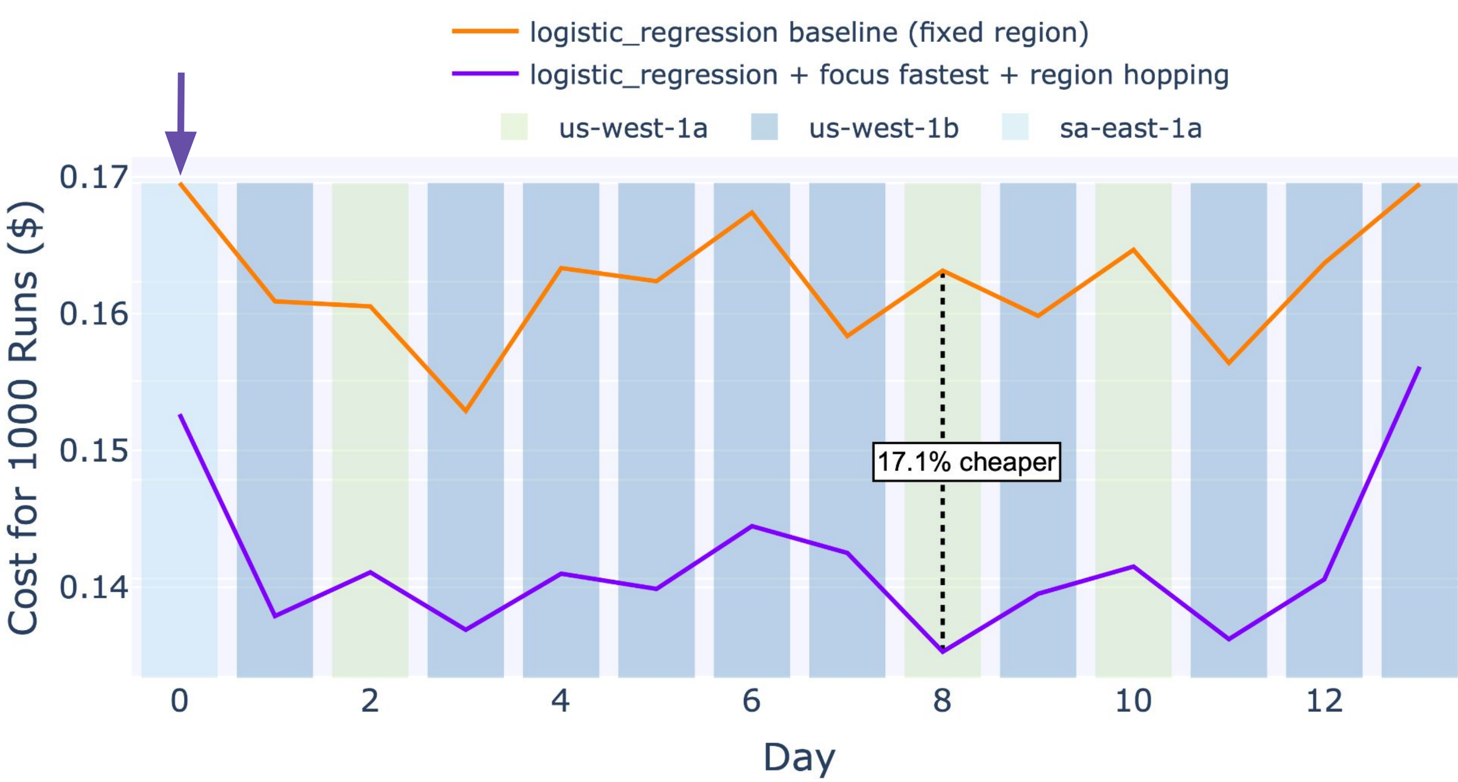

zipper + focus fastest
zipper + retry slow
zipper baseline
AMD EPYC
Intel @ 3.00GHz
Intel @ 2.50GHz
CPU Distribution impacts cost.
Times with less EPYC and
more 3.0 GHz reduces costs
18.5% cheaper
Cost for 1M Runs ($)
CPU Percent Distribution (%)
Day (us-west-1b)



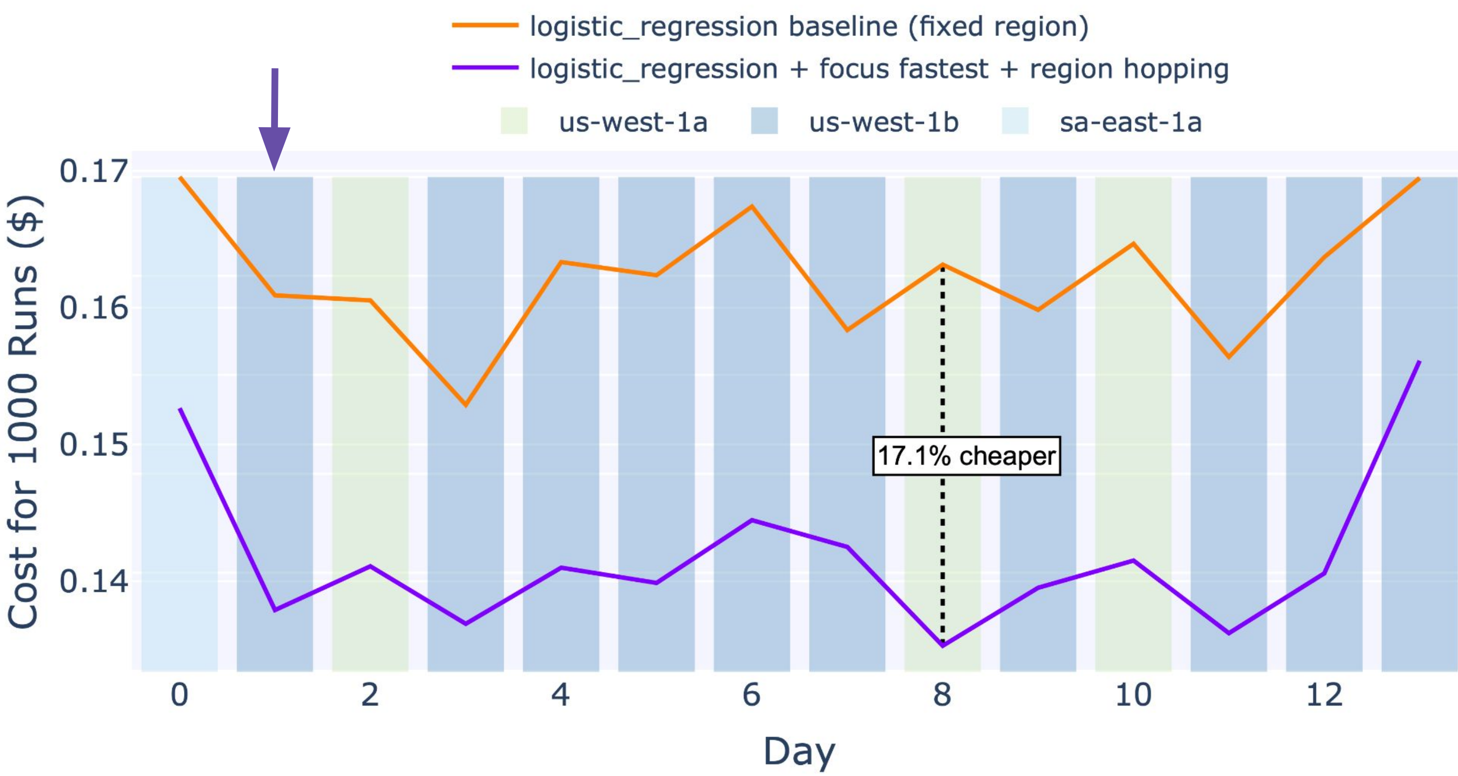
Region Hopping



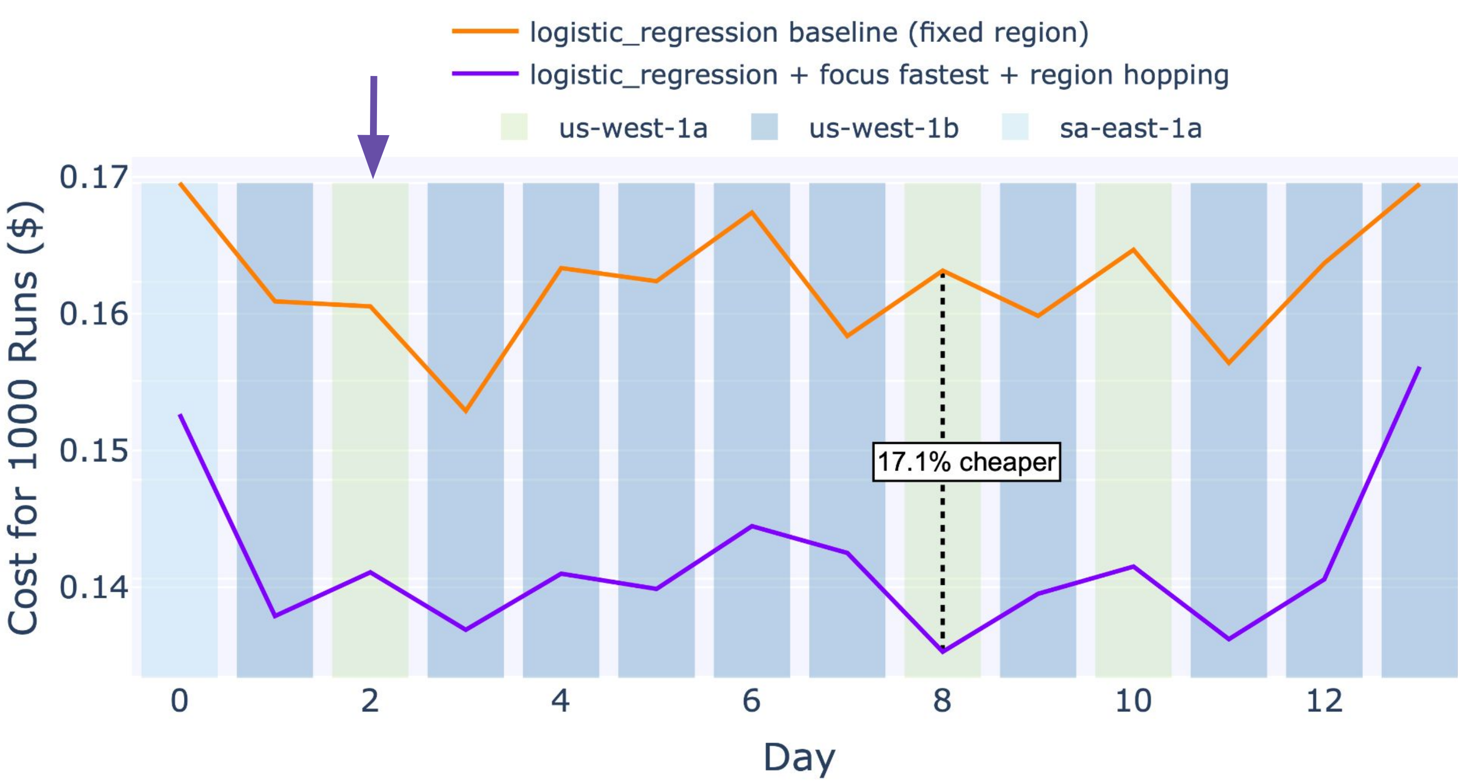
- Baseline fixed region of us-west-1b



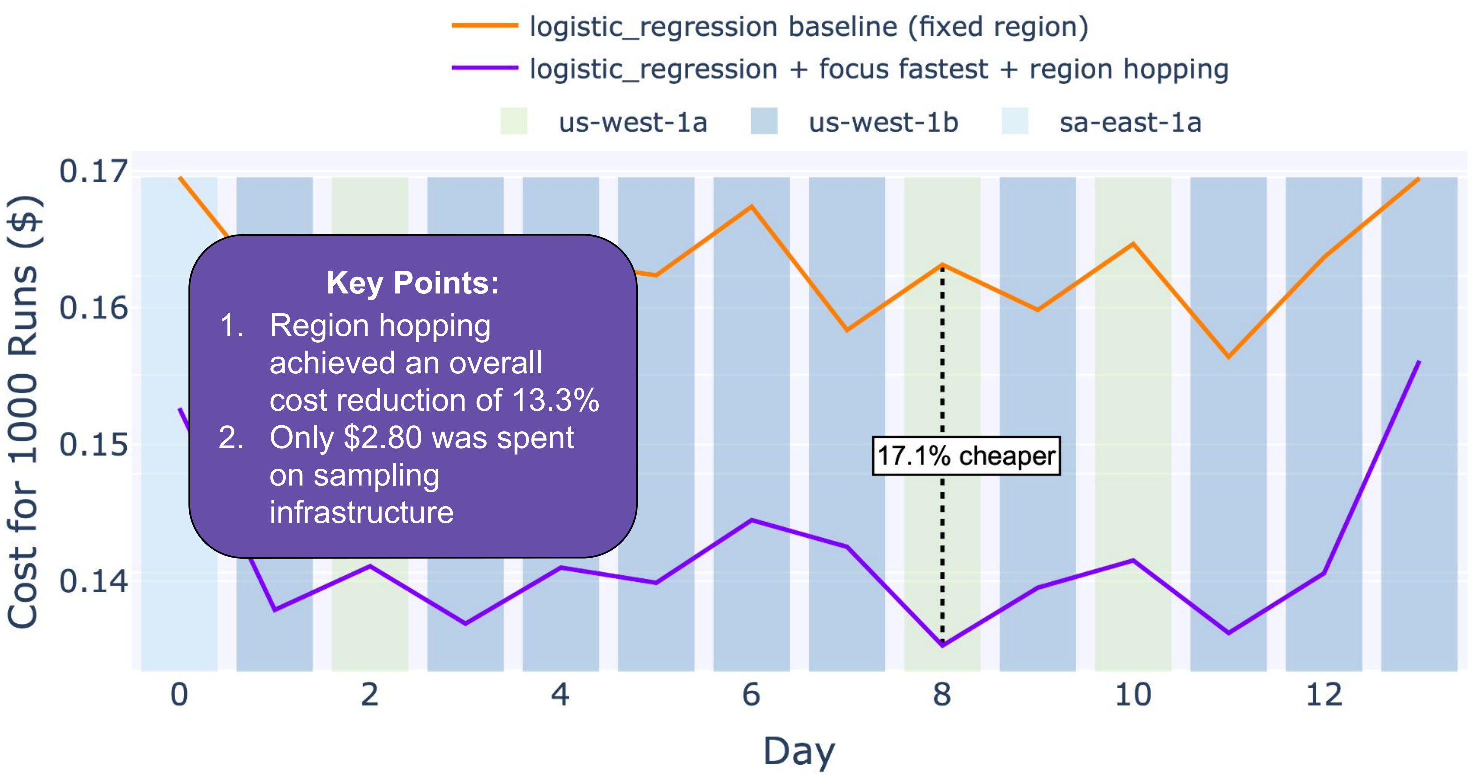
- Baseline fixed region of us-west-1b



- Baseline fixed region of us-west-1b



- Baseline fixed region of us-west-1b

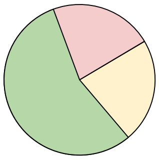


- Baseline fixed region of us-west-1b

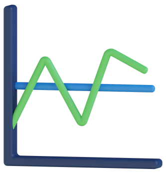
Conclusions

Key Results

- **RQ-1:** We were able to observe diverse CPU characterizations across 41 serverless regions
- **RQ-1:** Our sampling technique appears to be able to saturate entire availability zones creating accurate hardware characterizations



- **RQ-2:** Our sampling technique is able to characterize an availability zone for only $0.04 with 95% accuracy
- **RQ-3:** The **Retry Slow** method resulted in a consistent 10.1% reduction in runtime and cost



- **RQ-3: Focus Fastest** had up to 18.5% lower costs, averaging 16.5% across all functions
- **RQ-3:** Compared to a baseline of running in us-west-1b, the **Region Hopping** hybrid approach averaged a cost savings of 10.0% up to 18.2%

Future Work

Future Work and Limitations

- The retry methods sacrifice responsiveness for cost savings
- Building CPU characterizations for many regions is expensive
- In the future, a complete Serverless Sky Computing platform could build the CPU characterizations as workloads run, eliminating the need for dedicated polling
- Instead of relying on individual characterizations, they could be updated as new functions execute

Thank You!

Any Questions?

This research has been supported by the of Washington Carwein-Andrews Distinguished Fellow Fund and AWS Educate Cloud Credits.