Function Memory Optimization for Heterogeneous Serverless Platforms with CPU Time Accounting

Robert Cordingly, Sonia Xu, Wes Lloyd

School of Engineering and Technology
University of Washington Tacoma
10th IEEE International Conference on Cloud Engineering
IC2E 2022

Outline

- <u>Background and Motivation</u>
- Research Questions
- CPU Time Accounting Memory Selection (CPU-TAMS)
  - CPU-TAMS on AWS Lambda
  - IBM Cloud Functions
  - DigitalOcean Functions
  - Google Cloud Functions
- Experiments and Results
- Conclusions

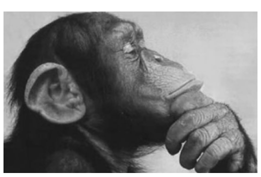
IBM Cloud Functions Apache OpenWhisk

Why 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

But there are still challenges…



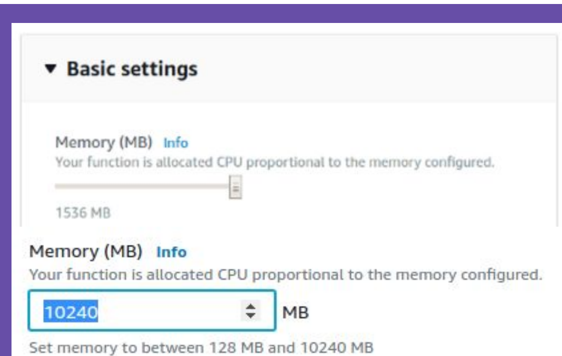
Serverless Function Memory Reservation Size?

AWS Lambda memory reserved for functions

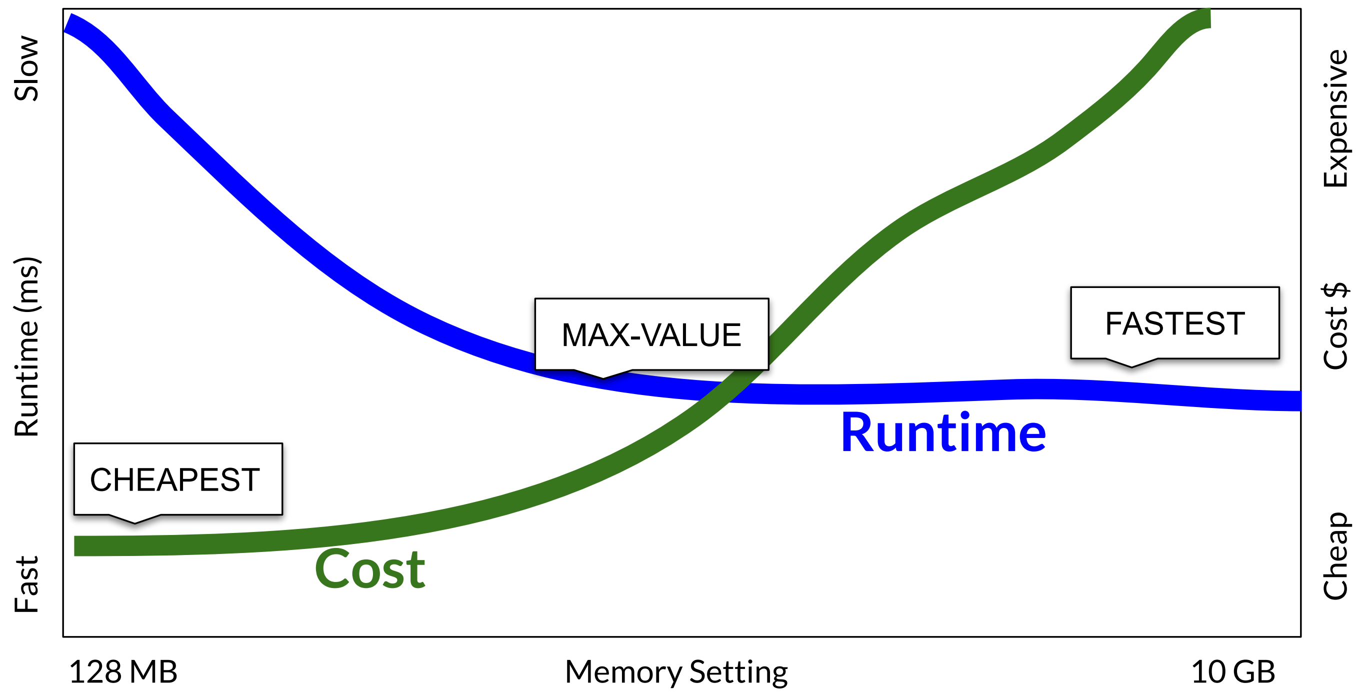
UI provides textbox to set function's memory *(previously a slider bar)*

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

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



But how much memory do functions require?


Slow
Runtime (ms)
Fast
MAX-VALUE
FASTEST
CHEAPEST
Runtime
Cost
Expensive
Cost $
Cheap
128 MB
Memory Setting
10 GB

Outline

Research Questions

- RQ-1: (**FaaS Resource Scaling**) How are resources, such as CPU, disk I/O, or network utilization, scaled with FaaS function memory reservation size?
- RQ-2 (**FaaS Memory Prediction**) How accurately can we predict FaaS function memory reservation size to achieve MAX-VALUE?

Outline

- Background and Motivation
- Research Questions
- CPU Time Accounting Memory Selection (CPU-TAMS)
    - CPU-TAMS on AWS Lambda
    - IBM Cloud Functions
    - DigitalOcean Functions
    - Google Cloud Functions
- Experiments and Results
- Conclusions

Selection Goals

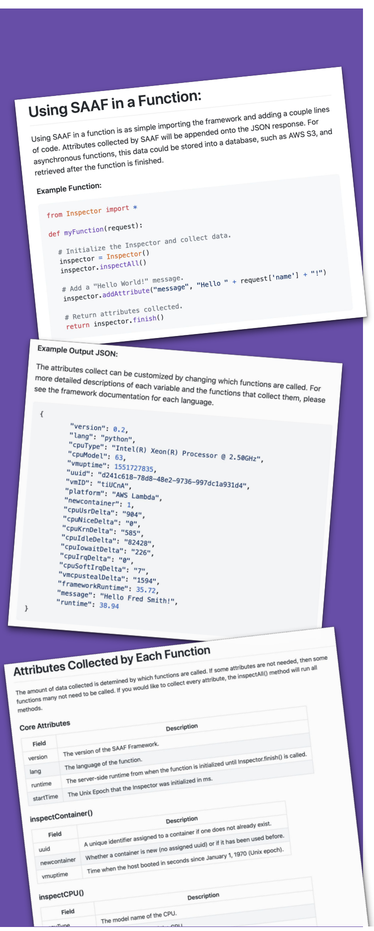
We investigated 3 selection goals. Each selection technique focuses on finding memory settings with a specific goal. CPU-TAMS focusing on finding MAX-VALUE memory settings.

Objective	Description
CHEAPEST	Lowest hosting cost with no regard to runtime.
FASTEST	Lowest runtime with no regard to cost.
MAX-VALUE	Maximizes the ratio between cost and performance. Offering both high performance and reduced cost.

$$cost = memory_{GB} * runtime_{S} * 0.00016667_{\$}$$

(AWS Lambda Pricing Policy [1])

$$value = -runtime_{S} * cost_{\$}$$



Supporting Tools - SAAF

We utilize the Serverless Application Analytics Framework to collect CPU Time Accounting metrics from serverless functions.

The function's operating system keeps track of how much time the CPU spends processing in different modes.

We utilize the CPU Time metics for our **CPU Time Accounting Memory Selection (CPU-TAMS)** method.

CPU Time Accounting

In previous work, we use CPU Time metrics to predict the runtime of serverless functions.

This can be done using the equation:

$$Runtime = \frac{cpuUsr + cpuKrn + cpuIdle + cpuIOWait + cpuIntSrvc + cpuSoftIntSrvc}{\#\ of\ vCPUs}$$

We can adapt this equation to calculate the number of utilized CPUs, by removing idle time and solving for the # of vCPUs:

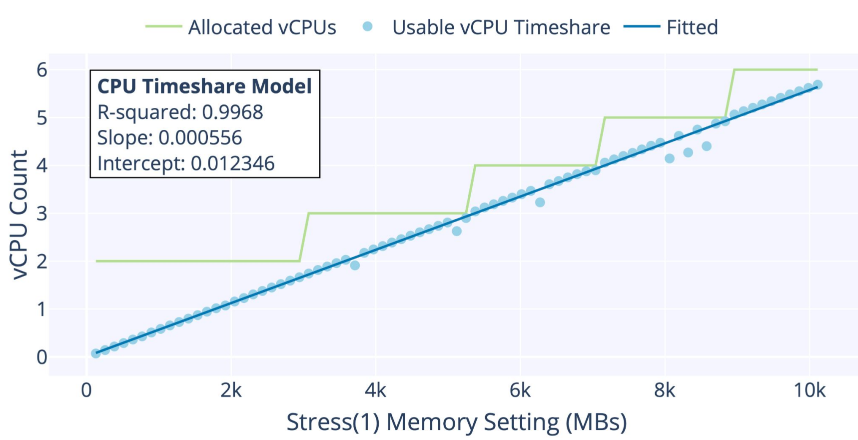
$$Utilized\ vCPUs_{pred} = \frac{cpuUsr + cpuKrn + cpuIOWait + cpuIntSrvc + cpuSoftIntSrvc}{Runtime_{obs}}$$

These two equations form the foundation for CPU-Time Accounting Memory Selection (CPU-TAMS).

CPU Time Accounting Memory Selection

By using a vCPU-to-memory model we can map the number of utilized vCPUs to a specific memory setting.

This memory setting should allocate an appropriate amount of infrastructure to the function to provide the fastest performance at the lowest price, usually achieving MAX-VALUE.



Algorithm 1 CPU Time Accounting Memory Selection

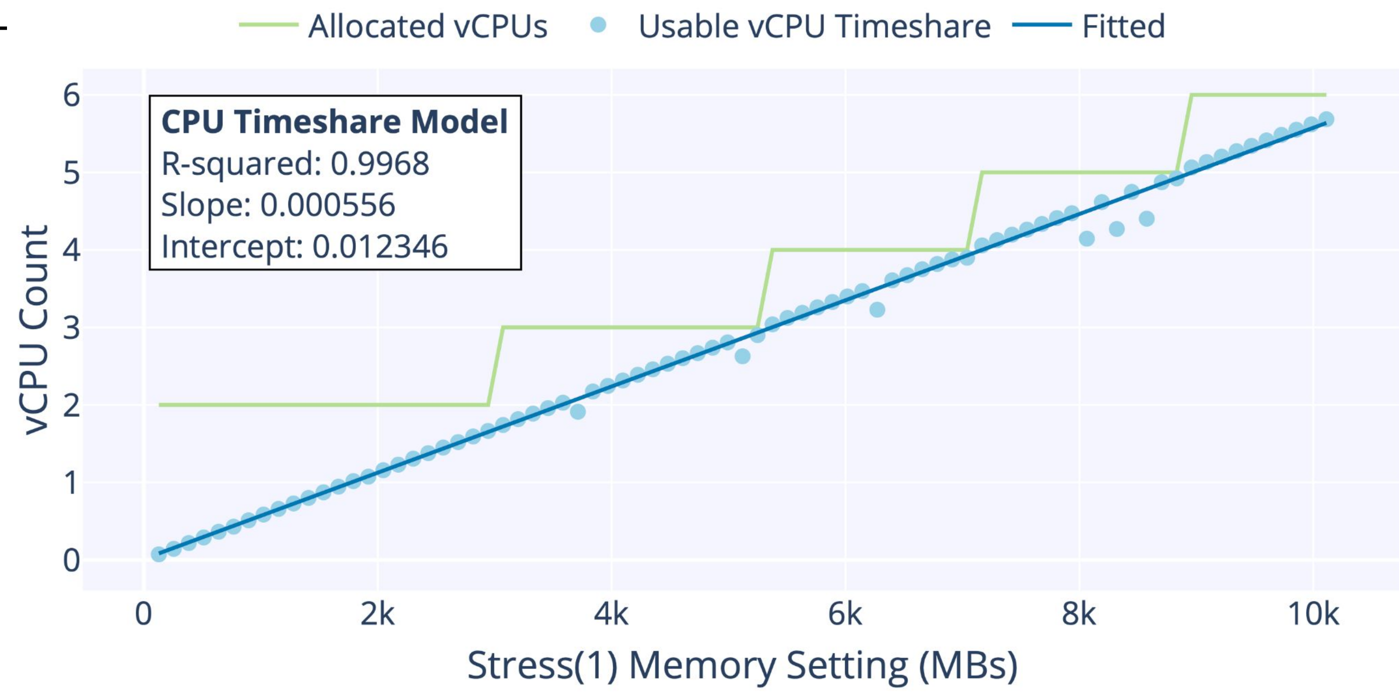
Require: Configure function to use max FaaS platform mem.

1. Profile workload to collect CPU metrics.

2. Calculate utilized vCPUs using equation (2)

3. Solve for memory using equation (3) derived from the linear regression model in Figure 1.

4. Adjust function memory to recommendation or the required memory reported in logs (whichever is higher).



Observed utilized vCPUs at each memory setting on AWS Lambda using Stress(1)

Baseline Selection Methods

We compared our CPU-TAMS approach to 3 rules of thumb, the AWS Compute Optimizer, and 4 search methods.

Name	Type	Description
MIN	RoT	Select the minimum memory required for a function
MID	RoT	Select a mid-range setting between MIN and MAX
MAX	RoT	Select the maximum available memory setting
CPU-TAMS	M	Utilize CPU Time Accounting metrics to calculate the average number of utilized vCPUs for a workload. Use vCPU model to predict value setting.
AWS-CO	M	The AWS Compute Optimizer. A tool by AWS that recommends memory settings. Requires >50 runs below 1792MB to make a recommendation.
Linear Search (LS)	S	Run a workload at many different memory settings, iterating linearly, and stopping when the settings meets a target goal.
Binary Search (BS)	S	Search through memory settings by iterating using a binary search algorithm. Test settings and progressively cut the memory setting range in half.
Gradient Descent (GD)	S	Search through memory settings by iterating using a gradient descent algorithm. Test settings and progressively move toward a value memory setting.
Brute Force	S	Run a workload at every available memory setting iterating with a desired step size.

(**RoT**: Rule of Thumb, **M**: Model, **S**: Search Method)

Outline

- Background and Motivation
- Research Questions
- CPU Time Accounting
- Memory Selection (CPU-TAMS)
  - CPU-TAMS on AWS Lambda
  - IBM Cloud Functions
  - DigitalOcean Functions
  - Google Cloud Functions
- Experiments and Results
- Conclusions

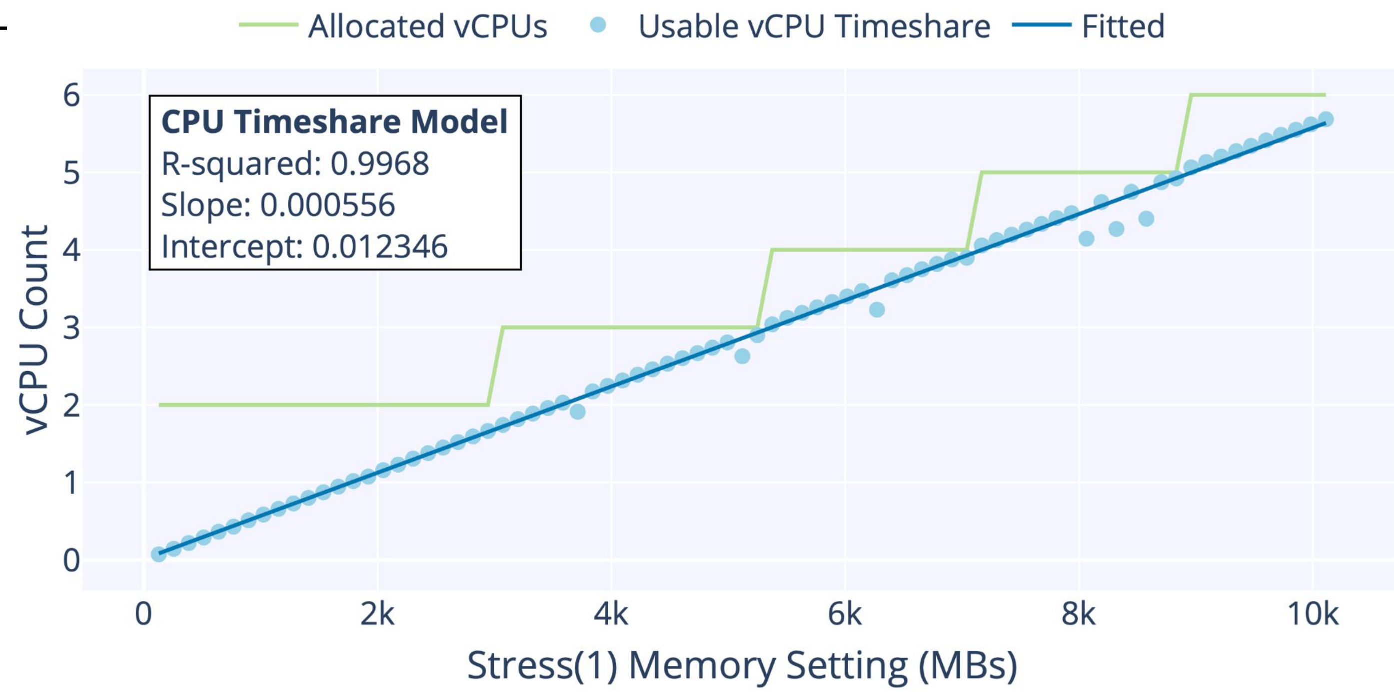
CPU-TAMS on AWS Lambda

AWS Lambda scales performance with memory setting by increasing available vCPU timeshare, linearly scaling vCPU allocation across the entire range of memory settings.

AWS Lambda offers partial vCPU allocations and CPU time accounting metrics are observable by SAAF.

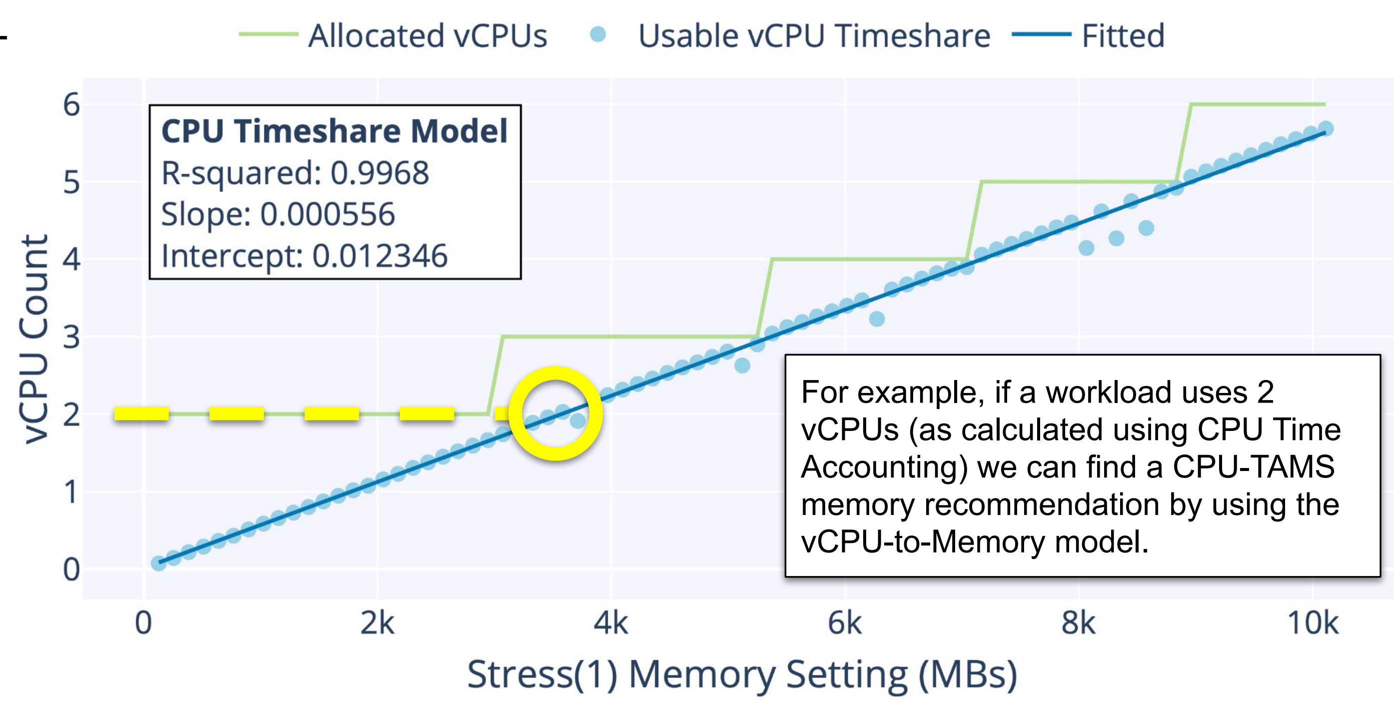
We constructed a vCPU-to-memory model on AWS Lambda by running a multi-threaded CPU bound function (e.g. Stress(1)) across the range of memory settings and measuring the available vCPU timeshare.

—



Fitted line shows the vCPU-to-Memory model for AWS Lambda.

—



Fitted line shows the vCPU-to-Memory model for AWS Lambda.

Outline

- Background and Motivation
- Research Questions
- CPU Time Accounting Memory Selection (CPU-TAMS)
  - CPU-TAMS on AWS Lambda
  - IBM Cloud Functions
  - DigitalOcean Functions
  - Google Cloud Functions
- Experiments and Results
- Conclusions

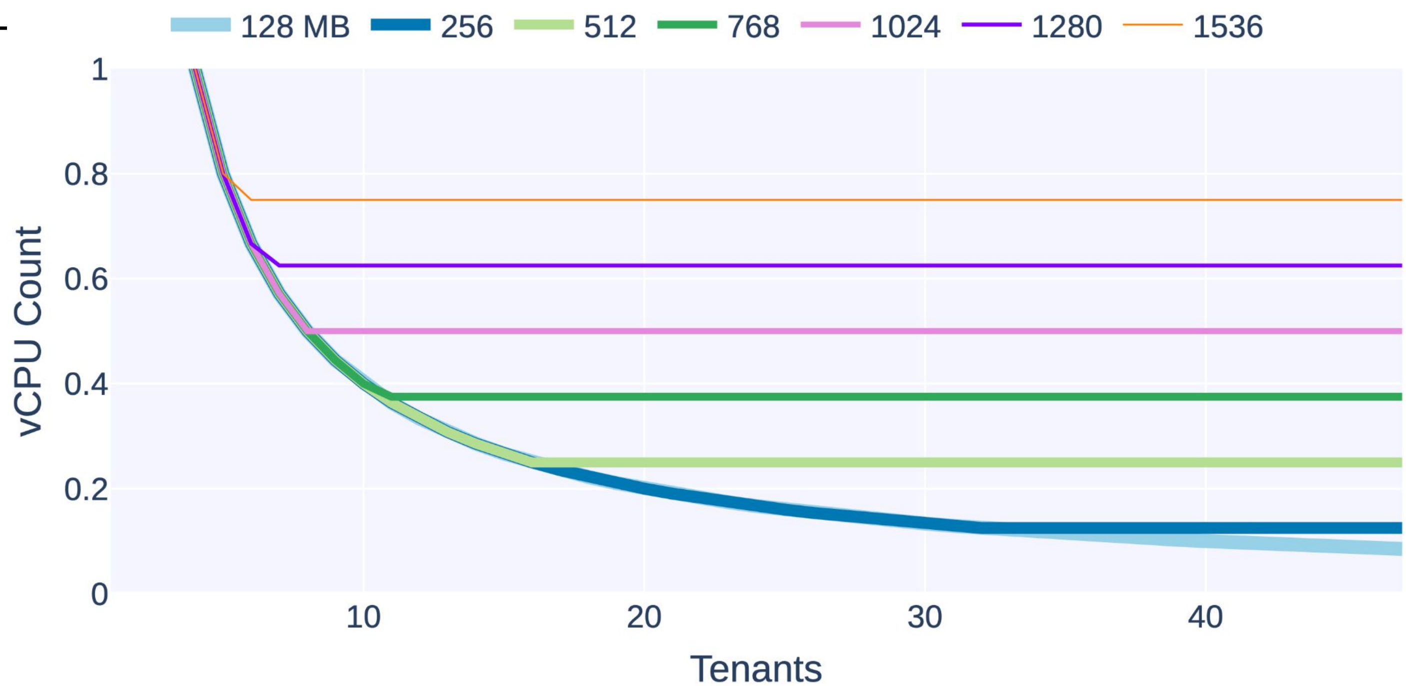
IBM Cloud Functions Apache OpenWhisk

CPU-TAMS on IBM Cloud Functions

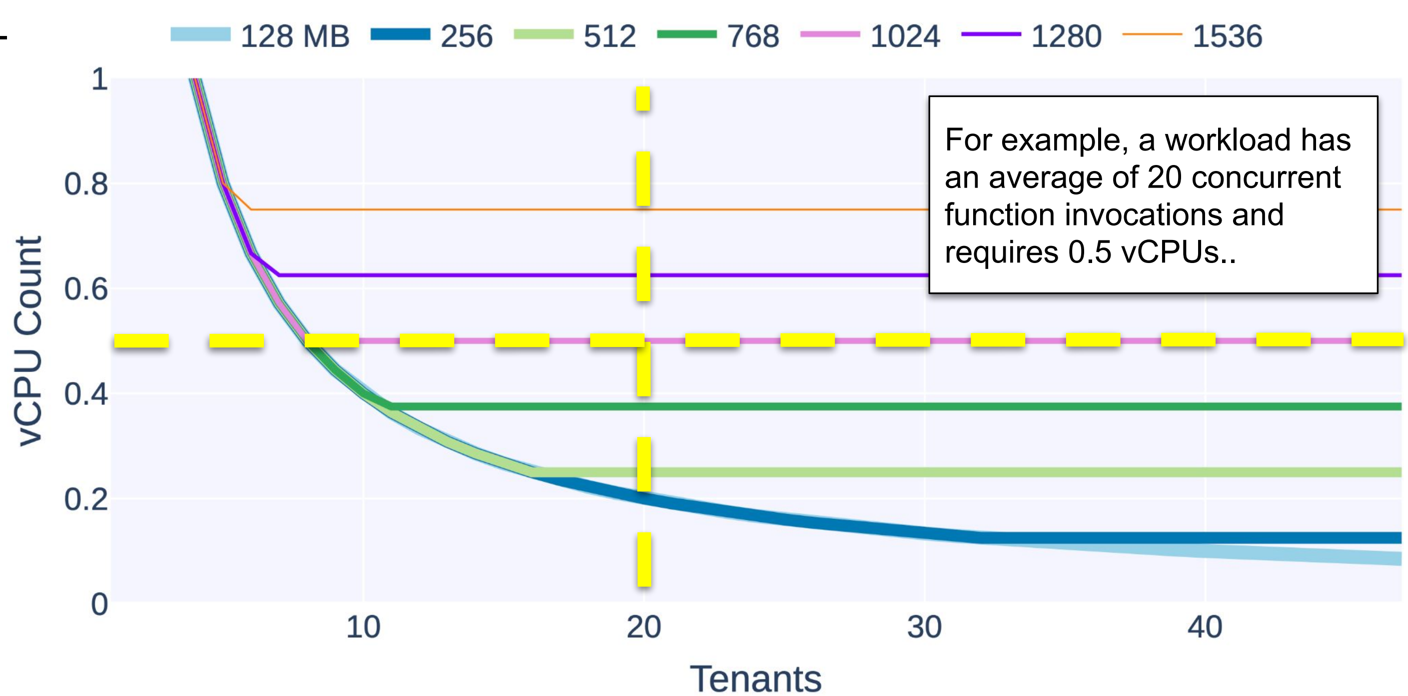
IBM Cloud Functions scales performance with function memory by reducing the number of tenants that share host VMs.

Functions are left to fight for resources, resulting in function memory settings having no impact on performance for sequentially called functions. High memory settings only improve performance for heavily concurrent workloads.

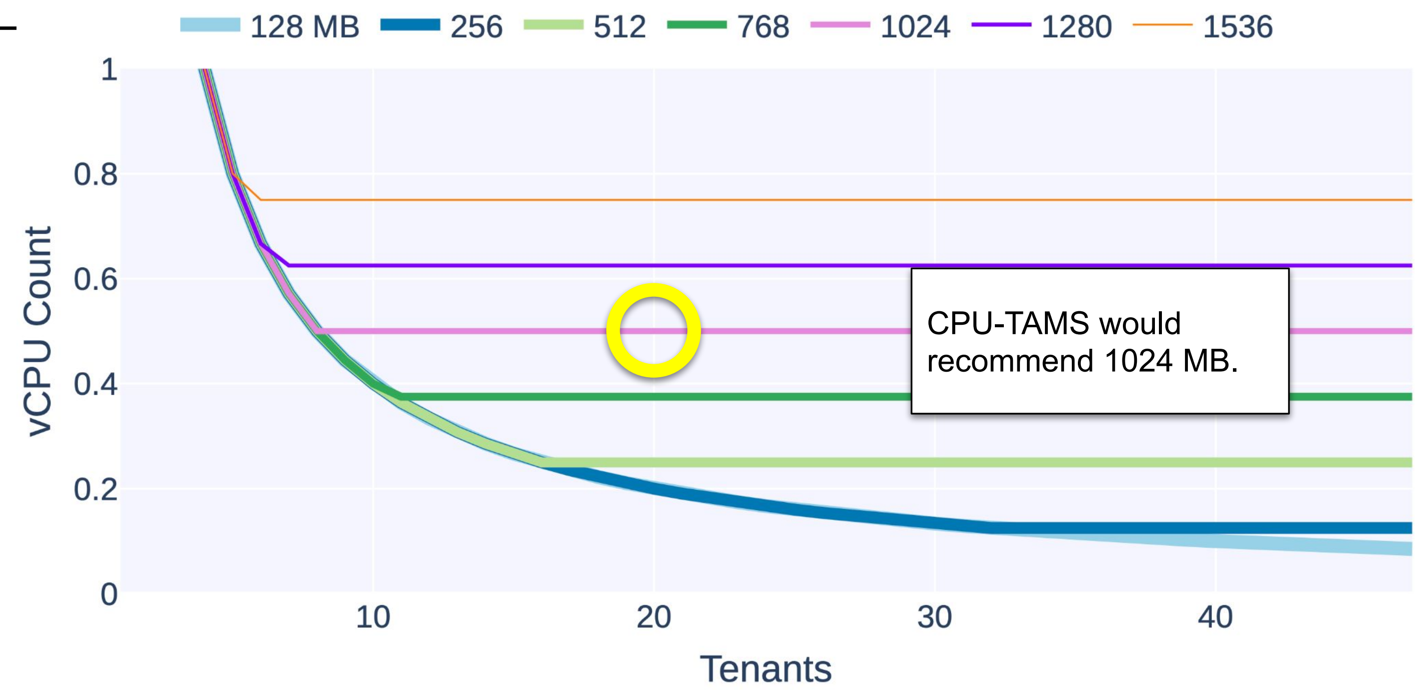
This leads to a vCPU-to-memory model with an additional dimension…



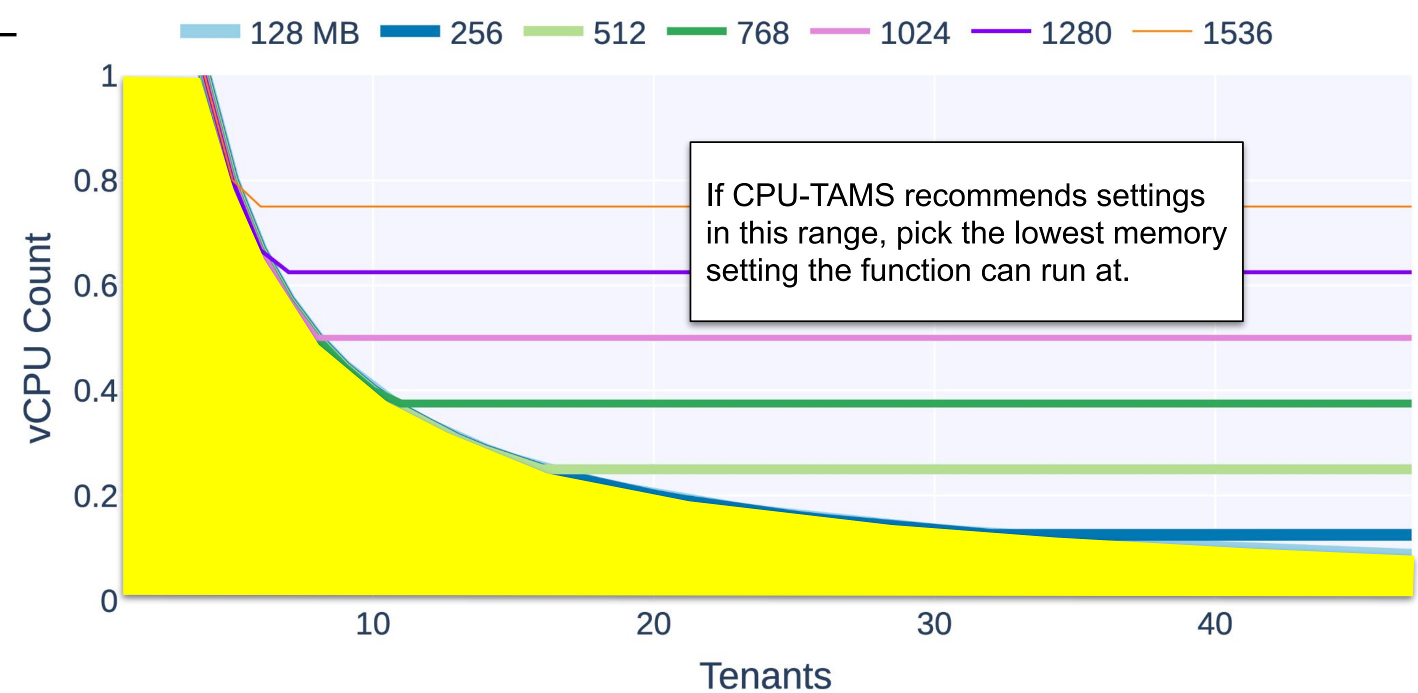
IBM Cloud Functions vCPU-to-Memory Model



IBM Cloud Functions vCPU-to-Memory Model



IBM Cloud Functions vCPU-to-Memory Model



IBM Cloud Functions vCPU-to-Memory Model

Outline

DigitalOcean

CPU-TAMS on DigitalOcean Functions

Both IBM Cloud Functions and DigitalOcean Functions use OpenWhisk for their backend. This results in both platforms scaling performance by limiting the number of functions that share infrastructure with a few key differences:

IBM Cloud Functions	**DigitalOcean Functions**
RAM: 128-2048 MB	RAM: 128-1024 MB
Host vCPUs: 4	Host vCPUs: 8
CPU Metrics: Observable	CPU Metrics: Not Available

CPU-TAMS on DigitalOcean Functions

DigitalOcean functions appears to use the same vCPU-to-Memory model as IBM Cloud Functions, although with a smaller range of memory settings.

Both IBM Cloud Functions and DigitalOcean functions do not allocate functions over 1 vCPU when called concurrently. This results in many functions benefiting from selecting the maximum memory setting.

Outline

- Background and Motivation
- Research Questions
- CPU Time Accounting Memory Selection (CPU-TAMS)
  - CPU-TAMS on AWS Lambda
  - IBM Cloud Functions
  - DigitalOcean Functions
  - Google Cloud Functions
- Experiments and Results
- Conclusions

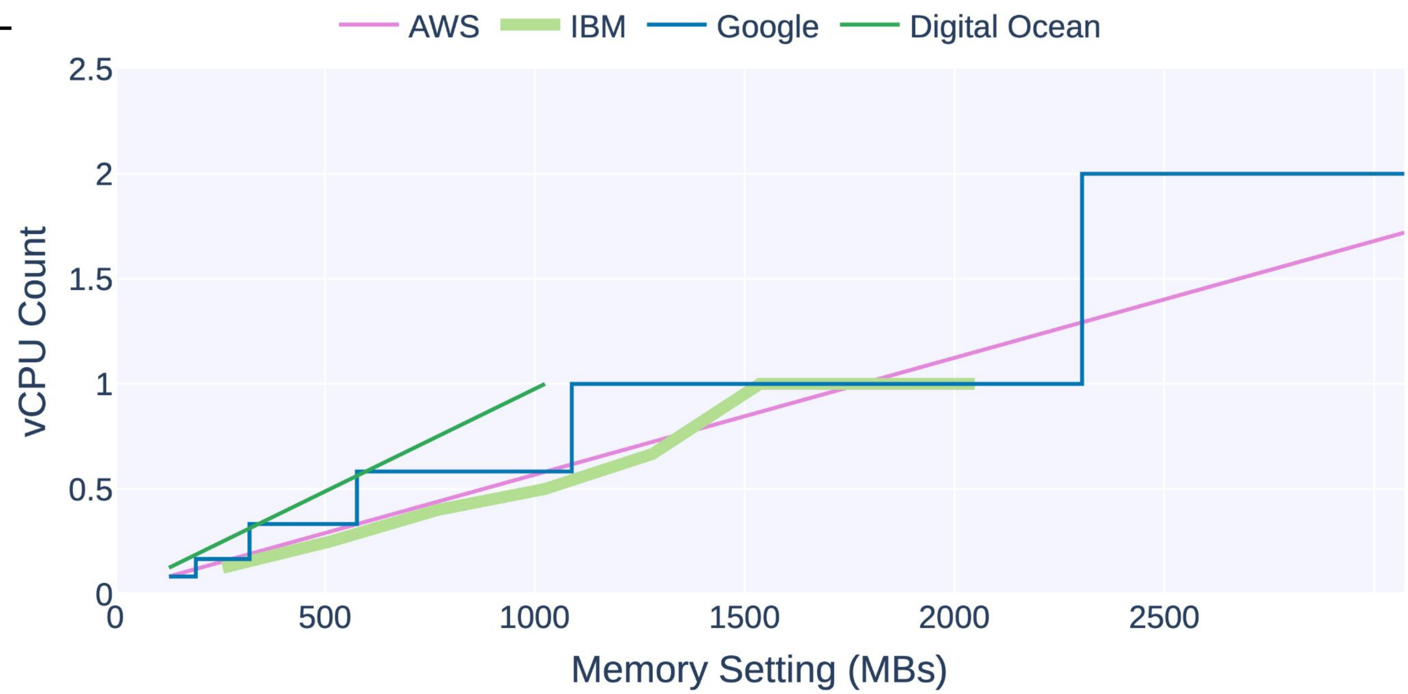
CPU-TAMS on Google Cloud Functions

Creating the vCPU-to-Memory model is incredibly easy on Google Cloud Functions.

Unlike all of the other platforms, GCF reports in the logs the exact number of vCPUs allocated to a function at each memory setting.

Although, GCF does not scale performance linearly, but uses a tiered approach where multiple memory settings will have the same number of vCPUs.

Platform Comparison



vCPU-to-Memory model for each platform. AWS Lambda and GCF extend to higher memory settings.

Platform Comparison

Each FaaS platform is different. We developed vCPU-to-Memory models for AWS Lambda, IBM Cloud Functions, DigitalOcean Functions, and Google Cloud Functions.

We also investigated Azure Cloud Functions and OpenFaaS. These platforms do not scale performance with a memory setting so CPU-TAMS is not applicable.

FaaS Platform	Memory (MB)	Scales w/ Memory	vCPU Cores	CPU Metrics
AWS Lambda	128-10240	CPU Timeshare	2-6	Available
IBM CF	128-2048	Max Tenancy	4*	Available
DigitalOcean	128-1024	Max Tenancy	8*	N/A
Google CF	128-16384	CPU Timeshare	1-5	N/A
Azure Functions	1536	N/A	2	Available
OpenFaaS	Any	N/A	Any	Available

* Up to with tenancy of 1

Outline

- Background and Motivation
- Research Questions
- CPU Time Accounting Memory Selection (CPU-TAMS)
  - CPU-TAMS on AWS Lambda
  - IBM Cloud Functions
  - DigitalOcean Functions
  - Google Cloud Functions
- Experiments and Results
- Conclusions

—

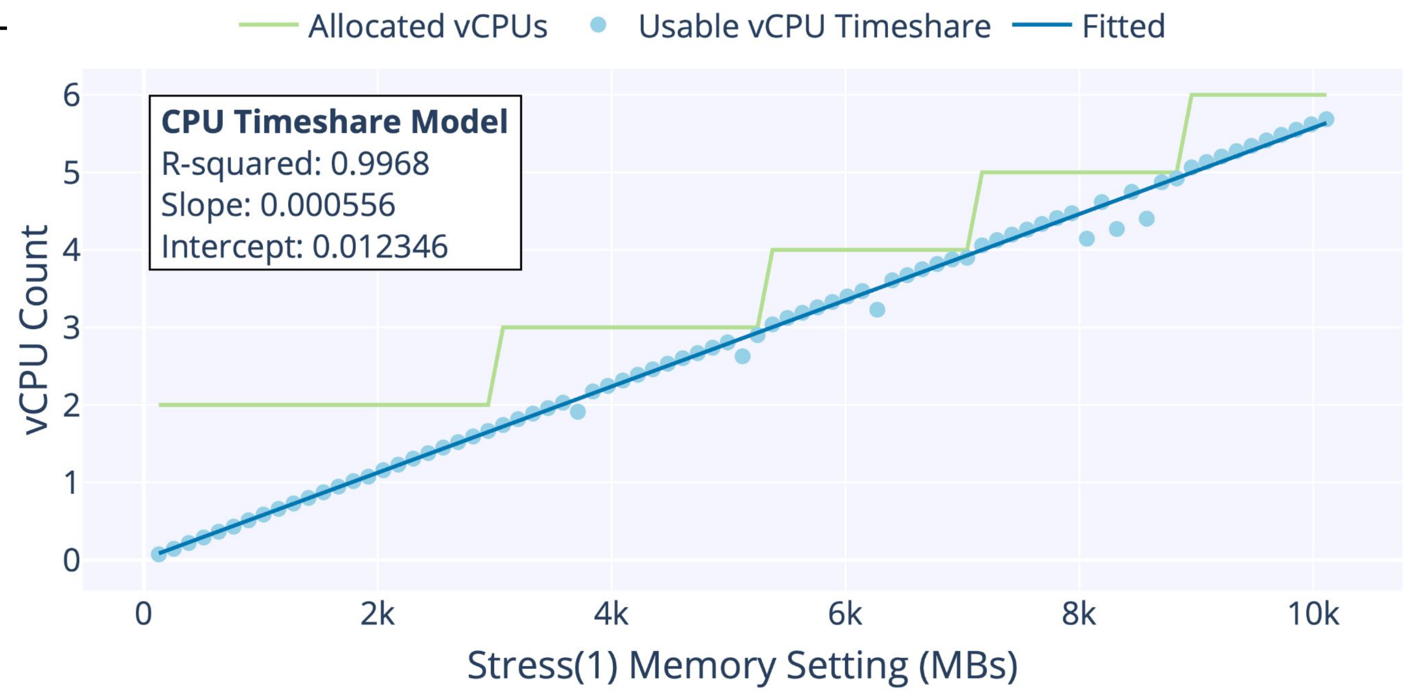
Functions

We used 14 functions across all of our experiments.

Some functions are only compatible with certain platforms.

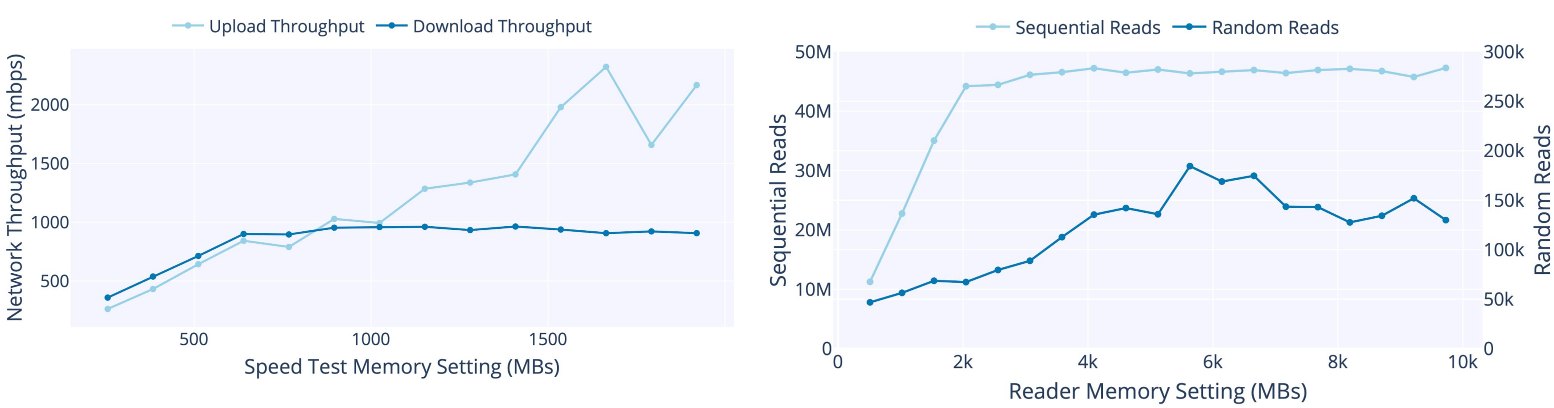
Function	Clouds	vCPU	Description
Sysbench	AI	n	Linux Benchmark used to generate prime numbers.
MST	AGID	1	Generates a graph and calculates the min spanning tree.
BFS	AGID	1	Generates a graph and processes a breadth first search.
Page Rank	AGID	1.2	Generates a graph and processes page rank of each node.
Writer	AGID	1	Generates text and repeatedly writes it to disk and deletes.
Compress	AGID	1	Generates files and compresses them into a zip file.
Resize	A	1	Pulls an image from S3, resizes it and saves it back to S3.
DNA	A	0.9	Pulls DNA sequence from S3 and creates visualization data.
TLQ	A	N/A	4 transform-load-query data pipelines 4 (Java/Python/Go/Node.js).
Speed Test	A	N/A	Network speed test created by Ookla.
Random Reader	A	1	Container that includes large files which are randomly read.
Calcs	AGID	n	Executes random math operations.
Stress(1)	AI	n	Linux tool used to generate CPU stress.
Sleep	AGID	0	Sleeps for a specified duration.

Clouds: **A**WS, **G**CF, **I**BM, **D**OF
Unshaded: CPU-TAMS Evaluation Functions, Shaded: Profiling Functions



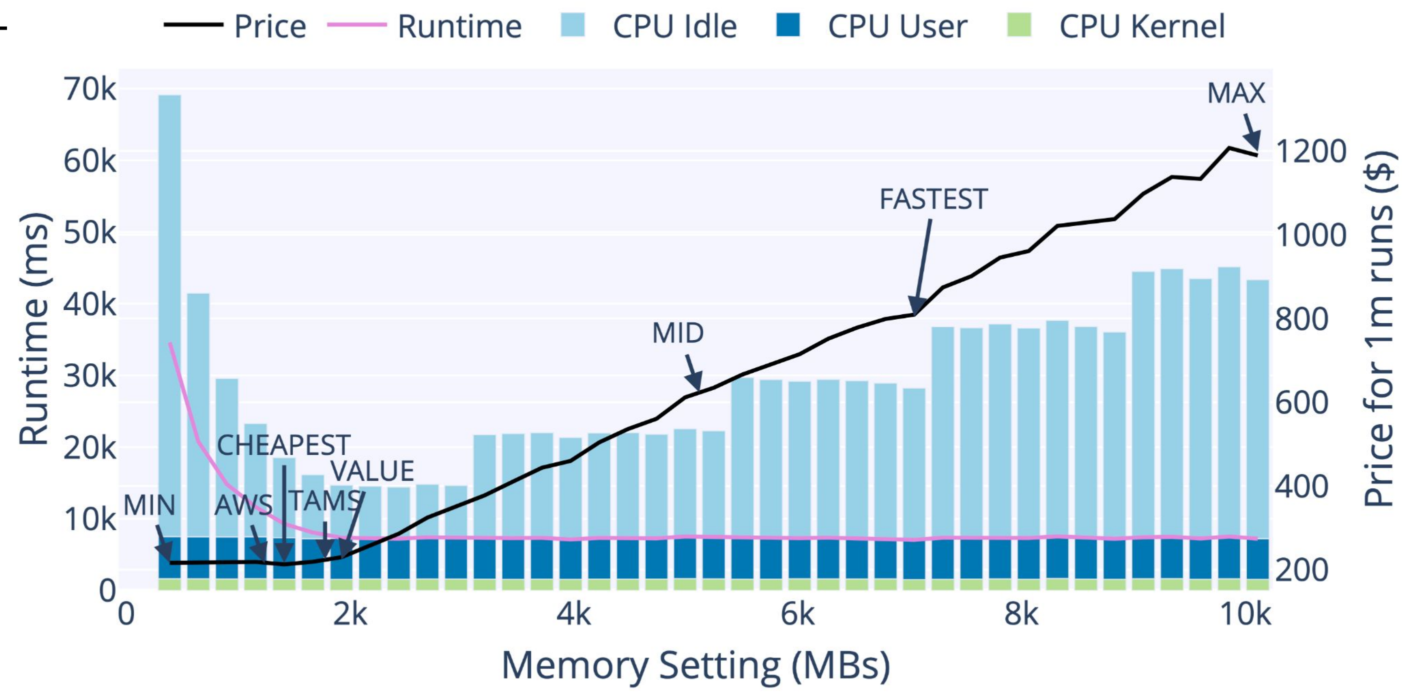
Observed utilized vCPUs at each memory setting on AWS Lambda using Stress(1)

RQ - 1 (FaaS Resource Scaling) Results: CPU Timeshare Scaling



Network I/O and /tmp read performance scaling on AWS Lambda

RQ - 1 (FaaS Resource Scaling) Results: Network and Storage Performance Scaling



Runtime and cost comparison of memory setting selections for Breadth First Search (BFS) Function.

RQ - 2 (FaaS Memory Prediction) Results: AWS Lambda

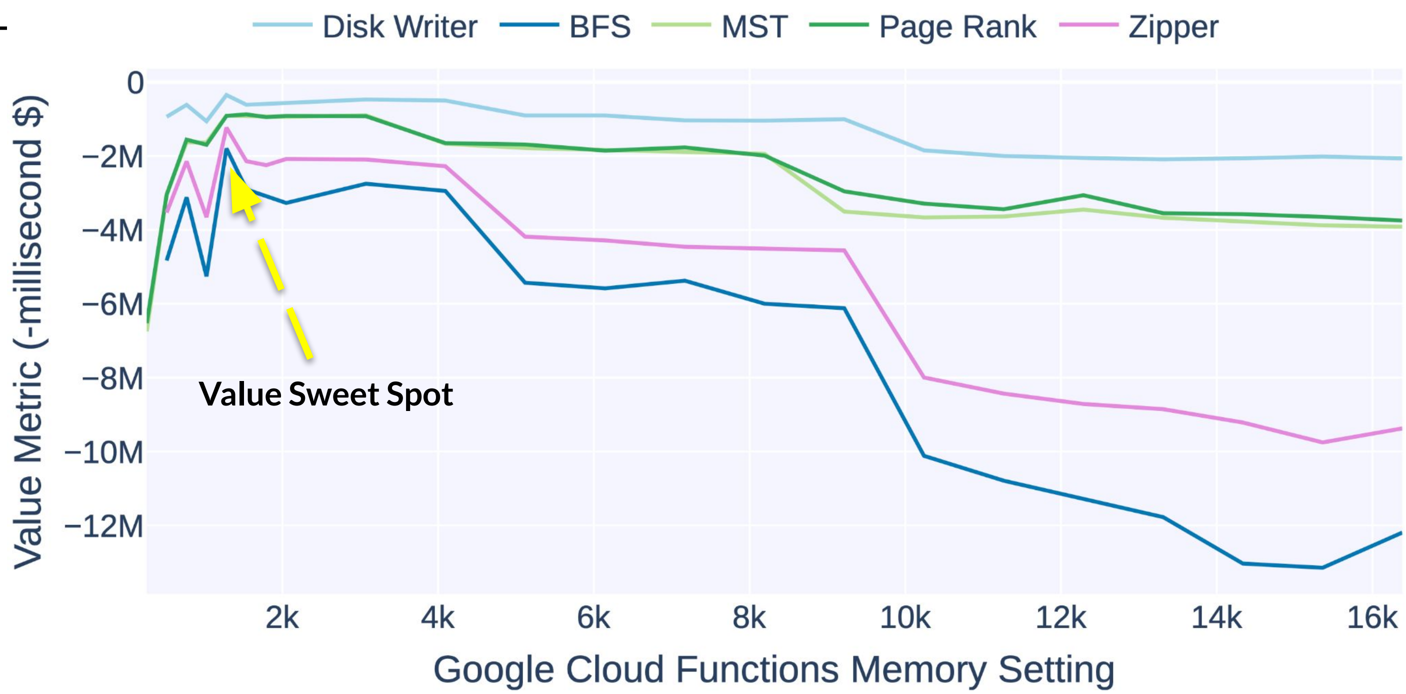
Function	Cheapest* Price Δ%	MIN Price Δ%	AWS-CO Price Δ%	CPU-TAMS Price Δ%	MID Price Δ%	MAX Price Δ%	Fastest* Price Δ%
Writer	−10	−7	−9	−2	160	410	160
Zip	−8	−5	−7	−6	150	406	232
Resize	−9	−7	−9	−7	142	406	142
DNA	−21	−15	−21	−9	165	406	0
PR	−6	−3	−6	11	144	368	325
MST	−3	4	−3	0	185	467	341
BFS	−7	−6	−5	−5	167	419	253
Sysbench	−8	−7	−8	−2	−5	0	0
Average	−9	−5.75	−8.5	−2.5	138.5	360.25	181.625

Function	Cheapest* Runtime Δ%	MIN Runtime Δ%	AWS-CO Runtime Δ%	CPU-TAMS Runtime Δ%	MID Runtime Δ%	MAX Runtime Δ%	Fastest* Runtime Δ%
Writer	23	367	50	13	−6	−4	−6
Zip	26	375	56	8	−4	−4	−6
Resize	172	1287	51	8	−7	−4	−7
DNA	50	384	50	19	7	7	0
PR	57	1355	57	−2	−11	−12	−13
MST	41	1247	41	0	−5	−7	−12
BFS	26	371	58	10	2	−2	−4
Sysbench	383	7296	711	6	92	0	0
Average	97.25	1585.25	134.25	7.75	8.5	−3.25	−6

Selection method average percent error compared to brute force discovered MAX-VALUE memory setting.

RQ - 2 (FaaS Memory Prediction) Results: AWS Lambda

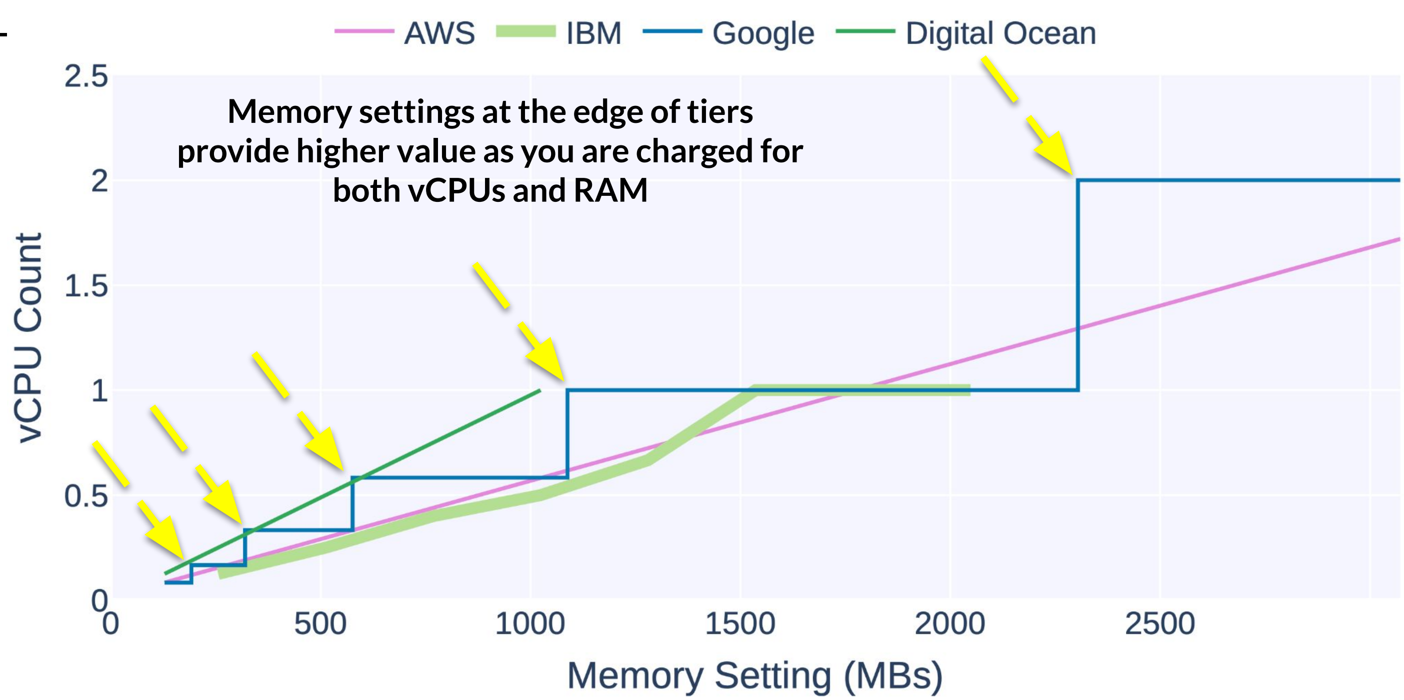
—



Function value comparison on Google Cloud Functions

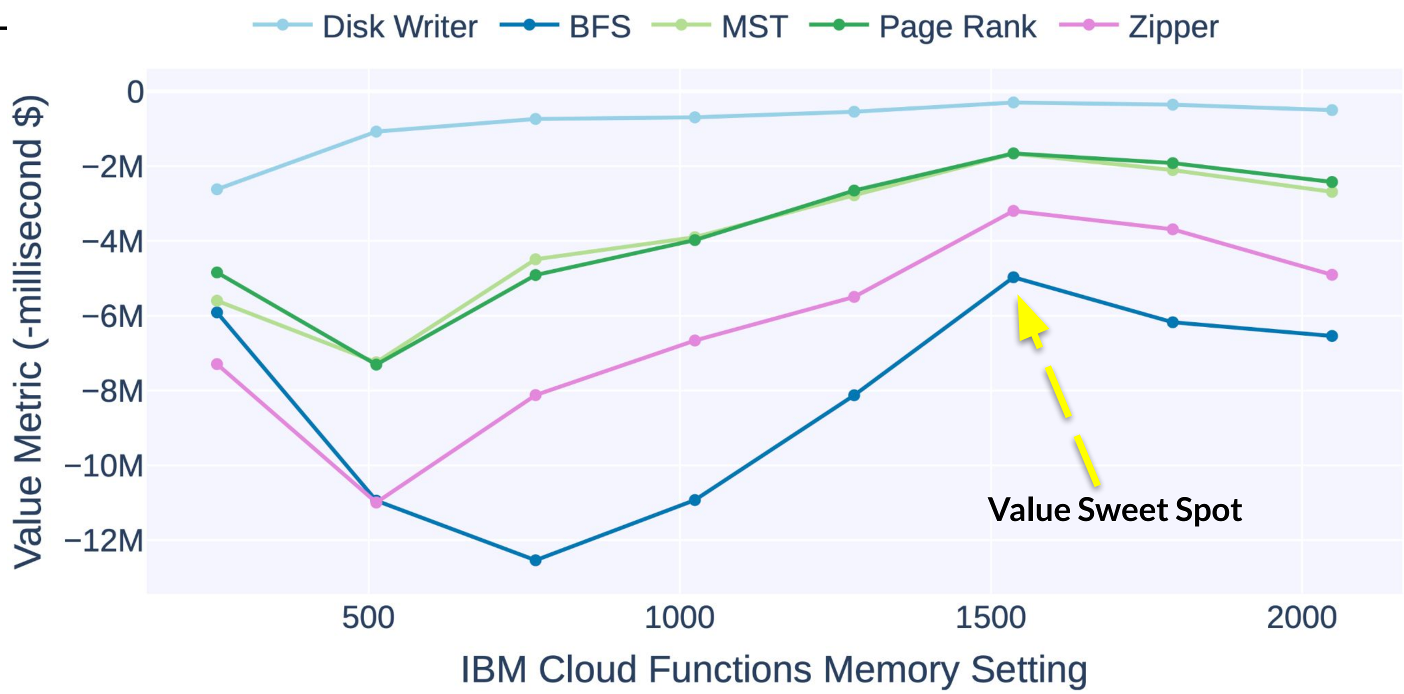
RQ - 2 (FaaS Memory Prediction) Results: Google Cloud Functions

—



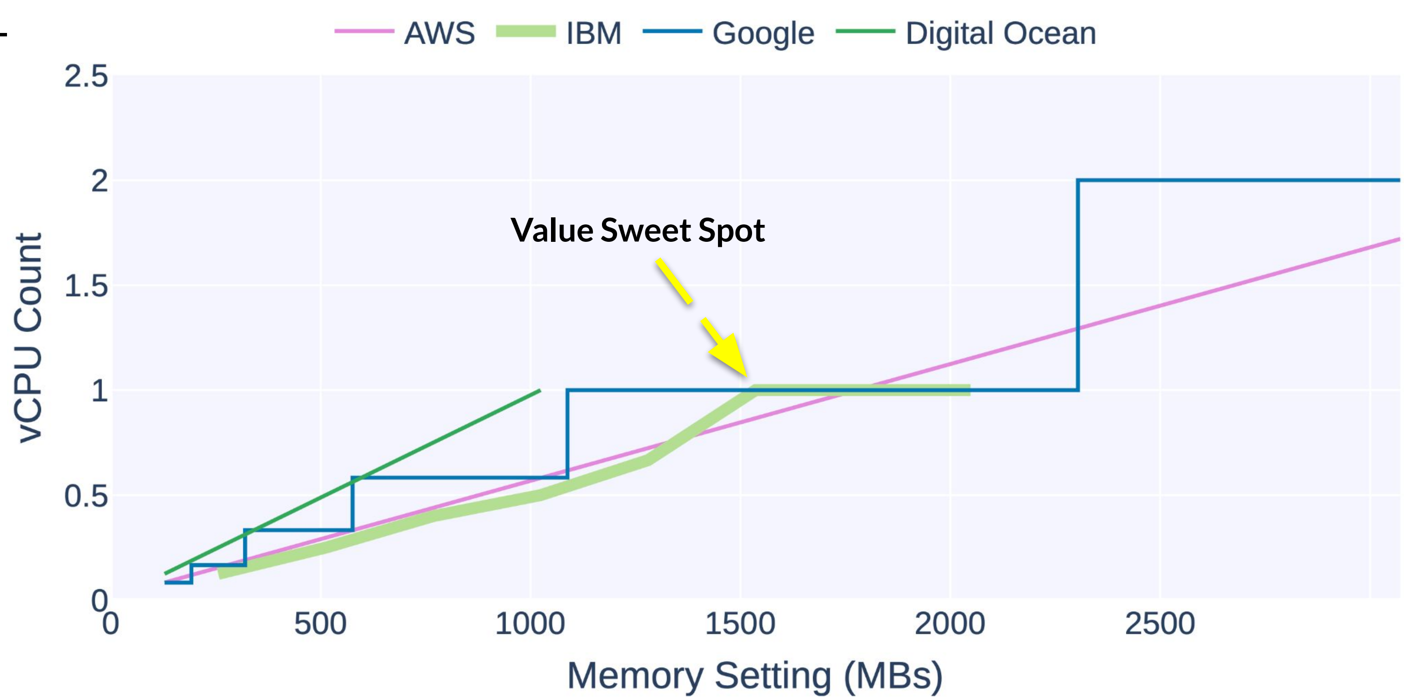
vCPU-to-Memory model for each platform. AWS Lambda and GCF extend to higher memory settings.

RQ - 2 (FaaS Memory Prediction) Results: Google Cloud Functions



Function value comparison on IBM Cloud Functions

RQ - 2 (FaaS Memory Prediction) Results: IBM Cloud Functions



vCPU-to-Memory model for each platform. AWS Lambda and GCF extend to higher memory settings.

RQ - 2 (FaaS Memory Prediction) Results: IBM Cloud Functions

Outline

- Background and Motivation
- Research Questions
- CPU Time Accounting Memory Selection (CPU-TAMS)
  - CPU-TAMS on AWS Lambda
  - IBM Cloud Functions
  - DigitalOcean Functions
  - Google Cloud Functions
- Experiments and Results
- Conclusions

Conclusions RQ-1 (FaaS Resource Scaling)

We found unique observations about each platform's resource scaling:

- AWS Lambda scaled vCPU, disk, and networking performance with memory setting.
- IBM and DigitalOcean scale performance by reducing the number of instances sharing host VMs.
  - IBM showed a distinct 'sweet spot' memory setting where performance was much higher than the rest.
- Google Cloud Function utilizes a tiered approach for vCPU allocation rather than linear like AWS.

Conclusions RQ-2 (FaaS Memory Prediction)

CPU-TAMS was able to find MAX-VALUE memory settings with only 5% cost, and 8% runtime mean absolute percent error compared to brute force discovered MAX-VALUE on AWS Lambda.

On all other platforms, CPU-TAMS was able to find the MAX-VALUE memory setting with no error by leveraging distinct characteristics of each platform's vCPU-to-memory scaling policy.

Our efforts demonstrate that a one-size-fits-all approach to find optimal FaaS function memory configurations for every platform is not possible as accounting for platform heterogeneity is required.

Thank You!

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

Questions?

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