Predicting Performance and Cost of Serverless Computing Functions with SAAF

Robert Cordingly
School of Engineering and Technology
University of Washington
Tacoma WA USA
rcording@uw.edu

Wen Shu
School of Engineering and Technology
University of Washington
Tacoma WA USA
shuwen12@uw.edu

Wes J. Lloyd
School of Engineering and Technology
University of Washington
Tacoma WA USA
wlloyd@uw.edu

***Abstract*— Next generation software built for the cloud recently has embraced serverless computing platforms that use temporary infrastructure to host microservices offering building blocks for resilient, loosely coupled systems that are scalable, easy to manage, and extend. Serverless architectures enable decomposing software into independent components packaged and run using isolated containers or microVMs. This decomposition approach enables application hosting using very fine-grained cloud infrastructure enabling cost savings as deployments are billed granularly for resource use. Adoption of serverless platforms promise reduced hosting costs while achieving high availability, fault tolerance, and dynamic elasticity. These benefits are offset by pricing obfuscation, as performance variance from CPU heterogeneity, multitenancy, and provisioning variation obscure the true cost of hosting applications with serverless platforms. Where determining hosting costs for traditional VM-based application deployments simply involves accounting for the number of VMs and their uptime, predicting hosting costs for serverless applications can be far more complex. To address these challenges, we introduce the Serverless Application Analytics Framework (SAAF), a tool that allows profiling FaaS workload performance, resource utilization, and infrastructure to enable accurate performance predictions. We apply Linux CPU time accounting principles and multiple regression to estimate FaaS function runtime. We predict runtime using a series of increasingly variant compute bound workloads that execute across heterogeneous CPUs, different memory settings, and to alternate FaaS platforms evaluating our approach for 77 different scenarios. We found that the mean absolute percentage error of our runtime predictions for these scenarios was just ~3.49% resulting in an average cost error of $6.46 for 1-million FaaS function workloads averaging $150.45 in price.**

Keywords*— *Serverless Computing, Function-as-a-Service, Performance Evaluation, Performance Modeling, Resource Contention, Multitenancy

I. Introduction

Serverless computing recently has emerged as a compelling approach for hosting applications in the cloud [1][2][3]. Serverless computing platforms promise autonomous fine-grained scaling of computational resources, high availability (24/7), fault tolerance, and billing only for actual compute time while requiring minimal setup and configuration. To realize these capabilities, serverless platforms leverage ephemeral infrastructure such as MicroVMs or application containers. The serverless architectural paradigm shift ultimately promises better server utilization as cloud providers can more easily consolidate user workloads to occupy available capacity, while deallocating unused servers, to ultimately save energy [4] [5]. Rearchitecting applications for the serverless model promises reduced hosting costs as fine-grained resources are provisioned on demand and charges reflect only actual compute time.

Function-as-a-Service (FaaS) platforms leverage serverless infrastructure to deploy, host, and scale resources on demand for individual functions known as “microservices” [6] [7] [8]. With FaaS platforms, applications are decomposed and hosted using collections of independent microservices differing from application hosting with Infrastructure-as-a-Service (IaaS) or Platform-as-a-Service (PaaS) cloud platforms. On FaaS platforms, temporary infrastructure containing user code plus dependent libraries are created and managed to provide granular infrastructure for each service [9]. Cloud providers must create, destroy, and load balance service requests across available server resources. Users are billed based on the total number of service invocations, runtime, and memory utilization to the nearest tenth of a second. Serverless platforms have arisen to support highly scalable, event-driven applications consisting of short-running, stateless functions triggered by events generated from middleware, sensors, microservices, or users [10]. Use cases include: multimedia processing, data processing pipelines, IoT data collection, chatbots, short batch jobs/scheduled tasks, REST APIs, mobile backends, and continuous integration pipelines [7].

Serverless computing with its many advantages possesses several important challenges. Unlike IaaS clouds, where cost accounting is as simple as tracking the number of VM instances and their uptime, serverless billing models are multi-dimensional. Software deployments consist of many microservices which must be individually tracked [11]. FaaS platforms exhibit performance variance that directly translates to cost variance. Functions execute over heterogeneous CPUs that host a variable number of co-located function instances causing resource contention. FaaS applications are decomposed into many functions that are hosted and scaled separately. The aggregation, or decomposition of application code into a varying number of FaaS functions can directly impact the composite size and cost of cloud infrastructure. **FaaS platform complexities including multi-dimensional billing models, heterogeneous CPUs, variable function tenancy, and microservice composition, leads to considerable pricing obfuscation for application hosting.**

XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

FaaS platforms presently lack tool support to estimate the costs of hosting applications. Current cloud pricing calculators from public cloud providers (e.g. AWS and Azure), and commercial tools (e.g. Intel Cloud Finder, RankCloudz, Cloudorado) primarily provide IaaS compute and storage cost estimates based on average performance [12][13][14]. Recently, FaaS calculators have appeared, but they are limited to generating cost estimates based on average runtime and memory size [15][16][17]. These calculators do not consider how FaaS function runtime scales relative to the memory reservation size, a feature coupled to CPU power on several FaaS platforms [18][19].

To address pricing obfuscation of FaaS platforms, in this paper, we offer a novel approach combining Linux CPU time accounting and multiple regression to provide highly accurate FaaS function runtime predictions. Equipped with performance predictions, FaaS workload costs can be estimated by applying the platform's pricing policy. Our approach involves profiling CPU metrics of multiple FaaS function deployments (e.g. AWS Lambda with 256, 512, 1024 MB to Intel Xeon E5-2680v2, E5-2676v3, E5-2686v4). We build regression models that predict how CPU metrics (e.g. CPU user mode time, CPU kernel mode time) scale across alternate function deployments with different CPUs and memory settings, and even to different cloud providers. By applying Linux CPU time accounting principles we can then estimate FaaS function runtime on any CPUs (e.g. Intel Xeon E5-2686v4), with any memory size (e.g. 1024 MB), on any cloud (e.g. IBM Cloud Functions). We note that cloud providers readily mix multiple CPU types to host FaaS functions. This CPU heterogeneity increases performance variance while decreasing performance model accuracy which we address in this paper. We evaluate our approach with compute bound functions for 77 different scenarios including deployments to alternate CPUs (36 cases), with alternate memory settings (27 cases), and to alternate platforms (14 cases). We found workload cost can be estimated with ~3.49% mean absolute percentage error (MAPE) by applying FaaS platform pricing policies, resulting in \$6.46 cost error, against an average workload price of \$150.45 for 1-million function call workloads. Our approach can help a developer predict FaaS workload costs to make informed deployment decisions. These advancements can enable developers to better evaluate deployment and design alternatives, while understanding cost implications to achieve more efficient serverless software implementations.

A. *Research Questions*

This paper investigates the following research questions:

RQ-1: (Performance Variance) What factors are responsible for performance variance on Function-as-a-Service (FaaS) platforms? How much do these factors contribute to performance variance?

RQ-2: (FaaS Runtime Prediction) When leveraging Linux CPU time accounting principles and regression modeling, what is the accuracy of FaaS function runtime predictions for deployments with different memory settings and different CPUs?

RQ-3: (Assessing Workload Predictability) How effective are system metrics, for example the number of page faults and context switches, at evaluating reliability of performance predictions?

B. *Research Contributions*

This paper provides the following research contributions:

1. We introduce the Serverless Application Analytics Framework (SAAF), a reusable programming framework that supports characterization of performance, resource utilization, and infrastructure metrics for software deployments to FaaS platforms (AWS Lambda, Azure Functions, Google Cloud Functions, and IBM Cloud Functions) in popular languages (Java, Python, and Node.js).
2. We detail performance variance of CPU-bound functions on AWS Lambda and IBM Cloud Functions. We characterize performance variance from heterogeneous CPUs, and function multitenancy across different memory sizes. **(RQ-1)**
3. We evaluate our FaaS function runtime prediction approach that combines Linux CPU time accounting and multiple regression for deployments across alternate CPUs, memory reservation sizes, and platforms. We evaluate our predictions to determine root mean squared error (RMSE) and MAPE, while identifying factors that impact accuracy using successive compute-bound workloads each introducing more non-determinism. We evaluate our approach for compute-bound functions for 77 different scenarios producing runtime predictions for: alternate CPU types (36), alternate memory settings (27), and alternate platforms (14). (**RQ-2, RQ-3**)

II. Background and Related Work

The challenge of performance prediction on serverless platforms, including the need to address performance variance resulting from hardware heterogeneity is identified in [20]. The authors identify how pay-as-you-go pricing models, and the complexity of serverless application deployments, leads to the key pitfall: ***"Serverless computing can have unpredictable costs"***. In contrast to application hosting with VMs, serverless platforms complicate budgeting as organizations must predict service utilization to estimate hosting costs. Performance variance of serverless workloads and accuracy of runtime predictions is invariably linked. We review related work on cloud performance variance, performance modeling, and performance evaluation of serverless platforms highlighting relationships to our research goals.

A. *Performance Variance of Cloud Systems*

In the public cloud, key factors often responsible for producing performance variance include *hardware heterogeneity*, *provisioning variation*, and *resource contention*. Ou and Farley identified the existence of heterogeneous CPUs that host identically labeled VM types on Amazon EC2, leading to IaaS cloud performance variance [21][22]. Rehman et al. identified the problem of "provisioning variation" in IaaS clouds in [23]. Provisioning variation is the random nature of

VM placements that generates varying multitenancy across physical servers producing performance variance from resource contention. Schad et al. showed the unpredictability of Amazon EC2 VM performance resulting from provisioning variation and resource contention from VM multitenancy in [24]. Ayodele et al. and Lloyd et al. demonstrated how resource contention from multi-tenant VMs can be identified using the cpuSteal metric in [25] [26].

On serverless FaaS platforms Jonas et al. identified heterogeneous CPUs and noted their potential to complicate performance modeling in [20]. Wang et al. identified heterogeneous VM types on FaaS platforms from AWS, Azure, and Google in [5]. They observed 4 CPU types and 5 VM configurations (AWS Lambda), 3 CPU types x 3 VM configurations (Azure functions), and 4 CPU types (Google Cloud Functions). Their efforts did not evaluate the extent of performance variance possible from heterogeneous CPUs.

Previous research has identified how provisioning variation results in varying degrees of multitenancy on FaaS platforms [4] [5] [27]. We identified how the number of function "tenants" on VMs, called "function instances" by Wang, increased when scaling up the number of concurrent requests on AWS Lambda [4]. Conversely, increasing function memory reduced the number of tenants on a VM. Wang observed that function instance placement across VMs on AWS Lambda used greedy placement, where concurrent requests are packed onto individual VMs until available memory (3328MB) is exhausted. Multiple functions from a single user account were found to share VMs, but VMs did not appear to be shared with other users. On Azure, the maximum observed tenancy of function executions did not exceed 8, while up to 4 user accounts shared VMs. While these efforts identified the multitenancy, they did not evaluate performance implications from resource contention.

B. Performance Modeling of Cloud Systems

On IaaS clouds, domain specific approaches have been developed to model workload performance by incorporating specific metadata regarding the tasks [28][29][30][31]. Recently, offline and online machine learning approaches have been applied to model runtime of multi-stage, batch-oriented, scientific workflows. Using task metadata and resource utilization metrics as features provided accuracy improvements [32][33][34].

Other efforts at IaaS cloud performance and cost modeling have focused on cost-aware VM scheduling to support infrastructure management for VM placement [35][36][37][38]. Efforts to save costs by leveraging reduced-priced cloud VMs available through auction based pricing mechanisms, such as Amazon EC2 spot instances, have spurred considerable research [39][40][41]. In summary, existing approaches provide runtime predictions for batch-oriented workloads that execute across homogeneous cloud VMs. Other efforts focus on performance modeling for resource management, to optimize use of auction based VMs, or to help select an appropriate VM type. We are unaware of previous research that has focused on performance and cost modeling of serverless computing workloads.

C. Performance and Cost Evaluation of Serverless Platforms

Prior research on serverless platforms has focused on evaluating performance of FaaS platforms for hosting a variety of workloads. Several efforts have investigated performance implications for hosting scientific computing workflows [42][43][44][45]. Other efforts have evaluated FaaS performance for machine learning inferencing [46][47], NLP inferencing [48], and even neural network training [49]. To support cost comparison of serverless computing vs. IaaS cloud, Boza et al. developed CloudCal, a tool to estimate hosting costs for service-oriented workloads on IaaS (reserved), IaaS (On Demand), and FaaS platforms [50]. CloudCal determines the minimum number of VMs to maintain a specified average request latency to compare hosting costs to FaaS deployments. FaaS resources, however, were assumed to provide identical performance as IaaS VMs when functions were allocated 128 MB RAM. Wang et al. identified AWS Lambda performance at 128 MB as only ~1/10th of 1-core VM performance in [5] suggesting potential inaccuracies with CloudCal. Other efforts have conducted case studies to compare costs for hosting specific application workloads on IaaS vs. FaaS [27][51], and FaaS vs. PaaS [52]. We extend previous efforts by characterizing performance variance of workloads across FaaS platforms, and demonstrating our novel Linux time accounting approach to predict FaaS workload runtime and cost.

III. Methodology

In this section, we detail tools and techniques used to investigate our research questions (**RQ-1**, **RQ-2, RQ-3**). Section III.A describes the SAAF, the framework used to profile our serverless workloads, and section III.B describes FaaS Runner, a tool used to automate profiling experiments. Section III.C details our experimental workloads, and section III.D describes our approach to leverage Linux CPU time accounting principles to generate runtime predictions for FaaS workloads deployed with different configurations, or to alternate platforms.

A. The Serverless Application Analytics Framework (SAAF)

To support profiling FaaS software deployments we have developed the Serverless Application Analytics Framework [53]. SAAF supports characterization of performance, resource utilization, and infrastructure for FaaS workloads deployed to AWS Lambda, Google Cloud Functions, Azure Functions, and the IBM Cloud Functions commercial FaaS platforms [21][22][61][62]. SAAF supports characterization of workloads written in Java, Python, Node.js, Go, and with AWS Lambda custom runtimes. Programmers include the SAAF library and a few lines of code to enable profiling. SAAF collects metrics from the Linux `/proc` filesystem and appends them to the JSON payload returned by the function instance. Metrics are then processed by FaaS Runner (see section B) our custom client application for further analysis. Table I shows a selected set of key metrics collected by SAAF.

Commercial FaaS platforms (e.g. AWS Lambda, IBM Cloud Functions) expose or hide different metadata about the underlying Linux environments used to host functions. **In this paper, we focus on AWS Lambda and IBM Cloud Functions as both platforms offer production level support of Java.** On

Azure Functions, Java runs in a Windows environment causing Linux time accounting metrics used by our runtime prediction approach, described in section IV.B, to be unavailable. Google Cloud Functions does not presently support Java. SAAF's approach to data collection is applicable to any FaaS platform that exposes Linux CPU time accounting metrics.

TABLE I. RUNTIME, RESOURCE UTILIZATION, AND CONFIGURATION METRICS COLLECTED BY SAAF WITHIN A FUNCTION INSTANCE. 'Δ' INDICATES RAW AND DELTA VERSIONS ARE PROVIDED

SAAF Metric	Description	Source
instanceID	Cloud provider's unique ID for function runtime environment. On AWS Lambda this is the CloudWatch log stream ID.	environment variable
ΔconSwitches	Number of context switches	/proc/vmstat
ΔcpuIdle	CPU idle time in ms	/proc/stat
ΔcpuIOWait	CPU time waiting for I/O to complete	/proc/stat
ΔcpuIrq	CPU time servicing HW interrupts	/proc/stat
ΔcpuKrn	CPU time in kernel mode in ms	/proc/stat
cpuModel	CPU model number	/proc/cpuinfo
ΔcpuNice	CPU time executing prioritized processes	/proc/stat
ΔcpuSoftIrq	CPU time servicing soft interrupts	/proc/stat
cpuType	FaaS function function instance CPU type	/proc/cpuinfo
ΔcpuUsr	CPU time in user mode in ms	/proc/stat
saafRuntime	Overhead time in (ms) of SAAF metric collection	calculated by SAAF
freeMemory	FaaS environment free memory in MB	/proc/vmstat
latency	Difference between runtime measured by FaaS Runner and SAAF runtime metric	calculated by client
mjrPgFaults	VM major pagefaults for function instance	/proc/vmstat
newcontainer	0=function executes in new function instance 1=function executes recycled instance	calulcated by SAAF
Δpagefaults	VM pagefaults for function instance	/proc/vmstat
runtime	Server side total FaaS function runtime in ms	calculated by SAAF
totalMemory	FaaS environment total memory in MB	/proc/vmstat
userRuntime	Runtime of function minus SAAF time (ms)	calculated by SAAF
Δvmcpusteal	CPU ticks lost to other VMs or the hypervisor	/proc/stat
vmID	Unique id for the VM hosting this function	/proc/cgroup, /sys/hyprvsr

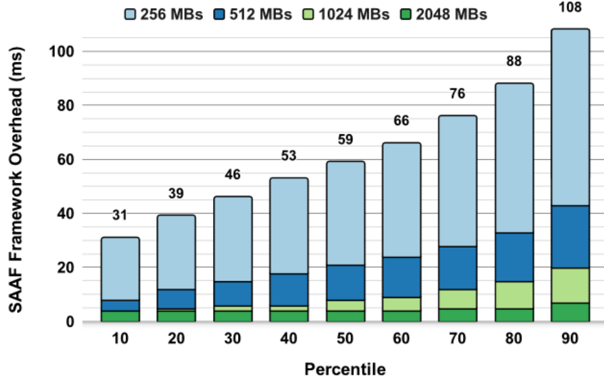

Fig. 1. SAAF profiling overhead percentiles (ms) at different memory settings on AWS Lambda

To determine function tenancy and potential resource contention, SAAF supports uniquely identifying VMs that host one or more function instances by implementing platform specific mechanisms. IBM Cloud Functions runs Xen 4.7 allowing the unique XEN hypervisor ID that is available from `/sys/hypervisor/uuid` [55] to be used as a method of VM identification. VMs can be uniquely identified on AWS Lambda with the sandbox-root ID in `/proc/$$/cgroup` [5].

Granularity of SAAF metric collection can controlled to specify which metrics to collect: CPU, memory, function instance, Linux, and platform metrics. We profiled the overhead of collecting metrics on AWS Lambda using a function only containing SAAF at 256MB, 512MB, 1024MB, and 2048MB and show the overhead in (ms) by percentile in Figure 1. AWS Lambda couples CPU timeshare with function memory allocation, reducing performance. Only for functions at 256MB, when collecting all metrics, did SAAF overhead exceed 100 ms, the billing unit of AWS Lambda in 10% of cases.

B. FaaS Runner

FaaS Runner provides a client-side application used in conjunction with SAAF. FaaS Runner supports automating profiling experiments across many different function configurations, while compiling results into a report that aggregates data for quick analysis. FaaS Runner combines the performance, resource utilization, and configuration metrics from many concurrent sessions enabling observations not possible when profiling individual FaaS functions calls. FaaS Runner is written in Python 3.6 and uses separate threads to host up to 1,000 individual, concurrent function invocations. Repeatable experiment configurations are defined using JSON files. Users define a set of input JSON payloads to distribute among function invocations, the number of concurrent or sequential calls to make, when to reconfigure FaaS function memory settings, and how to display the results. FaaS Runner groups results by CPU type, the virtual machine hosting function instances, or any other attributes defined in an experiment file. By categorizing results, FaaS Runner supports inferring the number of function instances sharing the same CPU type, VM, or any other unique attribute. This enables performance comparisons based on function tenancy, the number of function instances that share a host (VM).

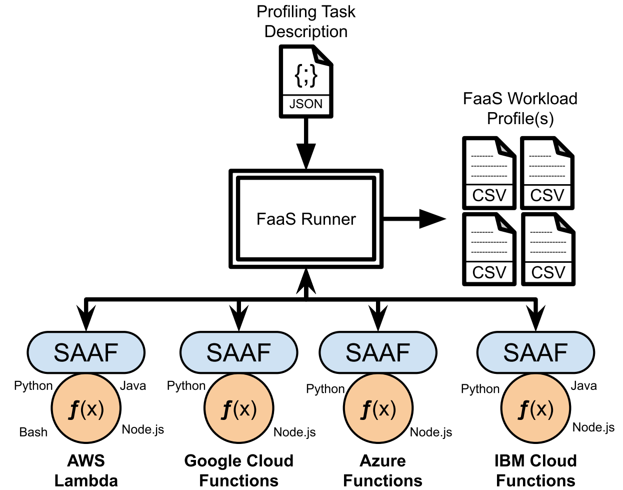

Fig. 2. Workload profiling with FaaS Runner and SAAF

C. Experimental Workloads

To evaluate our Linux time accounting and regression performance prediction approach, we developed a compute-bound function known as the "Calcs Service" (https://github.com/wlloyduw/CalcsService). This microservice produces workloads where a variable number of calculations are performed using the formula (`a × b ÷ c`) with operands stored

in separate large arrays on the heap. For each calculation, a random index is chosen into the arrays to store random numbers for use as operands. This is in contrast to multiplying operands stored using local primitive integers. To vary the degree of memory stress, the array size is adjusted from 1 to 1,000,000 elements. The calcs function was used in our workloads to perform a number of calculations between 30,000,000 to 60,000,000 to provide a variety of function runtimes to support training performance models. To ensure deterministic behavior, we used the same random seed for random array indexing to produce identical array access sequences for every execution. We also used the same random seed to generate identical "random" operand values. A child thread was introduced to create a multi-threaded workload where the child thread performs ½ the number of calcs to finish before the parent. The second thread adds CPU contention while the parent thread dictates the function's runtime. To evaluate performance predictions we did not use existing CPU benchmark applications because their binary executables may not always fit in the FaaS package space, and deploying binaries results in FaaS functions essentially being wrappers.

We profiled the alternate function configurations (e.g. CPU, memory, platform) described in table II and III to evaluate prediction accuracy where subsequent workloads introduce additional memory stress. Regression models are trained to convert individual CPU metrics from scenario-to-scenario using profiling data obtained from representative workloads. Developing a *one-size fits all generic model* to derive runtime predictions for any FaaS workload using generalized training data, similar to [56] for IaaS clouds, was not our objective for this paper.

TABLE II. EXPERIMENTAL WORKLOAD ALIAS AND DEFINITIONS

Name	Definition
NMT1	Fixed # of Calcs, **N**o **M**emory Stress, **1** **T**hread, concurrent calls
MT1	Fixed # of Calcs, **M**emory Stress, **1** **T**hread, concurrent calls
NMT2-seq	Fixed # of Calcs, **N**o **M**emory stress, **2** **T**hreads, **Seq**uential calls
NMT2	Fixed # of Calcs, **N**o **M**emory Stress, **2** **T**hreads, concurrent calls
MT2	Fixed # of Calcs, **M**emory Stress, **2** **T**hreads, concurrent calls
SCNMT1	**S**caling **C**alcs, **N**o **M**emory Stress, **1** **T**hread, concurrent calls
SCMT1	**S**caling **C**alcs, **M**emory Stress, **1** **T**hread, concurrent calls
SCNMT2	**S**caling **C**alcs, **N**o **M**emory Stress, **2** **T**hreads, concurrent calls
SCMT2	**S**caling **C**alcs, **M**emory Stress, **2** **T**hreads, concurrent calls
SCSMT2	**S**caling **C**alcs, **S**caling **M**emory Stress, **2** **T**hreads, concurr. calls

TABLE III. EXPERIMENTAL WORKLOAD CONFIGURATIONS

Workload Name	Calcs	Memory Stress	Threads	Tenancy
NMT1	40 million	No	1	n
MT1	40 million	array=1 million	1	n
NMT2-seq	40 million	No	2	1
NMT2	40 million	No	2	n
MT2	40 million	array=1 million	2	n
SCNMT1	30→60m step 3m	No	1	n
SCMT1	30→60m step 3m	array=1 million	1	n
SCNMT2	30→60m step 3m	No	2	n
SCMT2	30→60m step 3m	array=1 million	2	n
SCSMT2	30→60m step 3m	1→1m, step 100k	2	n

The Calcs Service supports generating FaaS workloads described in tables II and III. We profiled the total number of page faults running the two-thread calcs service (SCSMT2) on AWS Lambda at 256MB and observed 15.8x more average page faults with array sizes of 1,000,000 vs. 1, and 13.2x at 2048MB. More page faults occurred at lower memory settings because of higher function tenancy on VMs. This confirmed our memory stress approach successfully generates memory contention. Memory stress also significantly reduced performance. When comparing NMT1 and MT1 workloads, runtimes increased by (3.90x, 3.88x, 3.55x, 2.24x) for 256MB, 512MB, 1024MB, and 2048MB respectively. Memory stress also increased performance variance. The Coefficient of Variation (CV), defined as the standard deviation divided by the mean, provides a normalized comparison of performance variance. CV increased from (11.9→29.7%, 9.6→25.8%, 9.2→21.7%, 5.5→24.8%) with maximum memory stress for 256MB, 512MB, 1024MB, and 2048MB respectively.

D. Runtime Predictions with Linux Time Accounting

In this paper, we adapt our IaaS cloud performance modeling techniques leveraging Linux CPU time accounting for FaaS platforms [57]. Our approach is in contrast to traditional performance modeling approaches described in section II.B that leverage application metadata or resource utilization metrics as features to train models that directly predict runtime. Linux provides CPU time accounting by providing metrics that detail time spent in different CPU states measured in centiseconds (cs). Summing these metrics and dividing by the number of CPU cores provides the wall clock time of any profiled workload as in the formula:

$$\text{Workload}_{\text{time}} = \frac{cpuUsr + \text{cpuKrn} + \text{cpuIdle} + \text{cpuIOWait} + \text{cpuIntSrvc} + \text{cpuSftIntSrvc} + \text{cpuNice} + \text{cpuSteal}}{\#\ of\ CPU_{cores}}$$

In contrast to training models to predict runtime, we train individual models to predict individual CPU metrics for FaaS deployments with different configurations (e.g. memory, CPUs, or to alternate platforms). We then solve for workload runtime using the formula. In this paper, we focus on evaluating this approach for CPU-bound workloads. For these workloads, the majority of the variance is explained by CPU user mode time (cpuUsr) and CPU idle time (cpuIdle). We trained regression models to estimate how individual CPU metrics scale across different FaaS deployments to different CPUs, with different memory sizes, etc. Having models for specific CPUs allows accurate workload runtime and cost predictions for FaaS functions that run across heterogeneous CPUs. Table V provides data detailing an example of CPU heterogeneity on commercial FaaS platforms.

For FaaS functions with different memory reservation sizes, we observed on platforms that scale CPU power with memory (e.g. AWS Lambda and Google Cloud Functions), that cpuIdle time scales inversely with memory size. The workload's cpuUsr time remains approximately the same. In effect, the required cpuUsr time to complete the workload does not change, but changing the FaaS function memory alters the CPU timeshare for function execution, and this is reflected by cpuIdle.

To produce FaaS workload runtime predictions we profiled our workloads using a base configuration having a fixed CPU and memory setting (e.g. 256MB CPU E5-2680v2). We

generated regression models to convert cpuUsr and cpuIdle to a variety of target platforms. Deltas of cpuUsr, cpuIdle, CPU context switches, and page faults were used as independent variables. We did not incorporate application specific independent variables to ensure our approach is workload agnostic. Multiple regression models were trained and evaluated using RStudio [58]. We then applied Linux time accounting principles to predict function runtime for 77 different target configurations (e.g. 256MB CPU E5-2676v3 as one example).

Table IV describes our experiment source and target platform configurations. Numbers in parentheses indicate the maximum observed number of co-located function instances on the VM at different memory configurations. To refer to different CPUs we use aliases (e.g. a1, a2, i1, i2, etc.), described in Table V. To address function multitenancy in our models and normalize predictions, we filtered runs that did not exhibit maximum tenancy. Due to greedy function placement, the vast majority of concurrent function invocations were observed to exhibit maximum tenancy on AWS Lambda.

TABLE IV. RUNTIME PREDICTION SOURCE AND TARGET PLATFORM CONFIGURATIONS INCLUDING INTEL XEON E5 CPU, MEMORY, AND FUNCTION TENANCY PER VM IN PARENTHESES

Source Platform	Target Platform(s)
CPU Configurations:	
AWS 256MB 2680v2 (13)	AWS 2676v3:256(13), 512(6), 1024(3), 2048MB(1)
AWS 256MB 2680v2 (13)	AWS 2686v4:256(13), 512(6), 1024(3), 2048MB(1)
AWS 256MB 2676v3 (13)	AWS 2686v4:256(13), 512(6), 1024(3), 2048MB(1)
Memory Configurations:	
AWS 256MB 2680v2 (13)	AWS 512MB (6) v2, 1024MB (3) v2, 2048MB (1) v2
AWS 256MB 2676v3 (13)	AWS 512MB (6) v3, 1024MB (3) v3, 2048MB (1) v3
AWS 256MB 2686v4 (13)	AWS 512MB (6) v4, 1024MB (3) v4, 2048MB (1) v4
IBM Configurations:	
AWS 2048MB 2680v2 (1)	IBM 2048MB (4): 2683v3, 2683v4, 2650v4, 2690v4

TABLE V. OBSERVED RATIOS OF CPU TYPES ON AWS LAMBDA AND IBM CLOUD FUNCTIONS FAAS PLATFORMS

Platform	Intel Xeon CPU	VM	Alias	%
AWS	E5-2680v2 @ 2.8 GHz, 10 core	c3	**a1**	67.5
AWS	E5-2676v3 @ 2.4 GHz, 12 core	m4	**a2**	19.9
AWS	E5-2686v4 @ 2.3 GHz, 18 core	r4	**a3**	12.5
IBM	E5-2683v3 @ 2.0 GHz, 14 core	unseen	**i1**	18.4
IBM	E5-2683v4 @ 2.1 GHz, 16 core	bl2/b1/m1	**i2**	66.1
IBM	E5-2650v4 @ 2.2 GHz, 12 core	u1	**i3**	3.8
IBM	E5-2690v4 @ 2.6 GHz, 14 core	c1	**i4**	7.2
IBM	Gold 6140 @ 2.3 GHz, 18 core	unseen	**i5**	4.5

IV. EXPERIMENTAL RESULTS

To evaluate our research questions, we profiled the workloads described in Tables II & III on source and target platforms described in Table IV. We deployed AWS Lambda functions in the Virginia region. We pinned all Lambda functions to execute within in the same availability zone (e.g. us-east-1b) using a VPC to reduce hardware heterogeneity and increase the likelihood that experiments run with identical conditions. IBM Cloud Functions were deployed to the us-south Dallas datacenter. We identify the prevalence of heterogeneous CPUs found on both AWS Lambda-VPC (us-east-1b, 350 runs) and IBM Cloud Functions (south, 1000 runs) as observed in August/September 2019 in Table V. Our statistics illustrate one example of possible CPU variance on commercial FaaS platforms.

A. Performance Variance of FaaS Platforms

To quantify performance variance for **RQ-1**, we leveraged our calcs service with the NMT2-seq and NMT2 workloads. These workloads were designed to minimize non-deterministic performance behavior. Functions performed a static number of calculations, using the same operand values, without memory stress. Our goal was to quantify performance variance on AWS Lambda and IBM Cloud Functions.

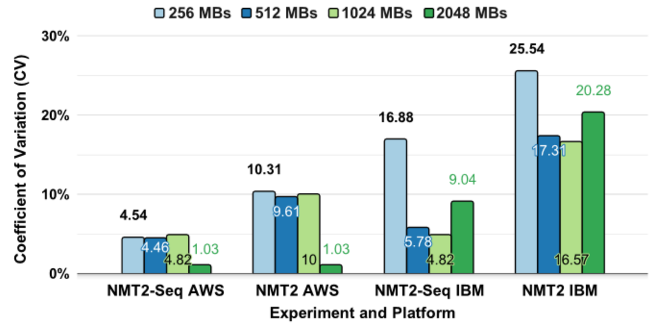

Fig. 3. FaaS workload performance variance resulting from heterogeneity in CPU type and the number of co-located function executions on host VMs based on the experiment, platform, and memory reservation size

We compared function runtime for NMT2-seq workloads performing 40,000,000 calcs (`a × b ÷ c`) using the same random seed to select operands. We ran this workload sequentially to force single tenant function execution to enable measuring performance across isolated VMs dedicated to individual requests on both platforms. We measured performance at 256, 512, 1024, and 2048MB and group results by CPU type. **CV on AWS Lambda was 4.54% (256MB), 4.46% (512MB), 4.82% (1024MB), and 1.03% (2048MB) for function executions across heterogeneous CPUs** as shown in Figure 3. Grouping runtime by CPU type reduced performance variance to only 0.64%, 0.42%, and 0.45% CV for the a1, a2, and a3 CPUs. **On IBM CV was 16.88% (256MB), 5.78% (512MB), 4.82% (1024MB), and 9.04% (2048MB) for function executions across heterogeneous CPUs.** Characterizing runtime by CPU type on IBM did not substantially improve CV: 10.79% (i2 CPU) and 4.07% (i3 CPU). Performance on AWS Lambda a3 CPUs outperformed a1 CPUs by 14.6% (256MB), 16.1% (512MB), and 16.4% (1024MB), while a3 was about 3.6% faster than a2. Despite the faster clock speed of the a1 CPU, it only outperformed a2 and a4 with maximum memory (3008MB), where the a1 CPU produced runtimes 19.8% less than a2, and 16.5% less than a3. This behavior reflects better single core performance with the a1 CPU, and better multi-core performance with the a2 and a3 CPUs which coincides with the evolution of Intel Xeon processors from v2→v3→v4. **CV increased as a result of CPU heterogeneity ~7.4x on AWS Lambda, but only ~1.2x on IBM Cloud Functions.** Single tenant performance variance for function execution with identical CPUs on IBM was more than 10x that of AWS Lambda as most of the NMT2-seq workload variance on IBM appeared to be not related to CPU heterogeneity. We explain key differences with IBM's FaaS platform at the end of section IV.B.

We next compared performance of the multithreaded NMT2 workload to investigate performance variance for multitenant function executions on FaaS platforms. Here many, but not all functions execute with identical tenancy due to greedy function placement on AWS, e.g. 256MB (commonly 13 tenants), 512MB (6 tenants), 1024MB (3 tenants), and 2048MB & 3008MB (1 tenant) for all CPUs. Here CV increased to 10.31% (256MB), 9.61% (512MB), and 10% (1024MB) across all CPUs as shown in Figure 3. On IBM, CV values from multitenancy were 24.54% (256MB), 17.31% (512MB), 16.57% (1024MB), and 20.28% (2048MB), nearly 2x more than AWS Lambda. **Moving from single tenant to multitenant function executions, CV increased ~2.7x on AWS Lambda, and ~2.5x on IBM Cloud Functions.** Given that Lambda employs greedy function placement across VMs, most executions occur at the same tenancy level. IBM Cloud Functions had higher CV for the single tenant NMT2-seq workload producing a lower increase with multitenancy (NMT2). We illustrate estimated hosting costs for 1,000,000 function calls in Figure 4, demonstrating how CPU heterogeneity translates to price volatility. The "combined" column projects the total cost based on observed CPU ratios. **FaaS CPU heterogeneity results in a lottery, where lucky users reap lower hosting costs.**

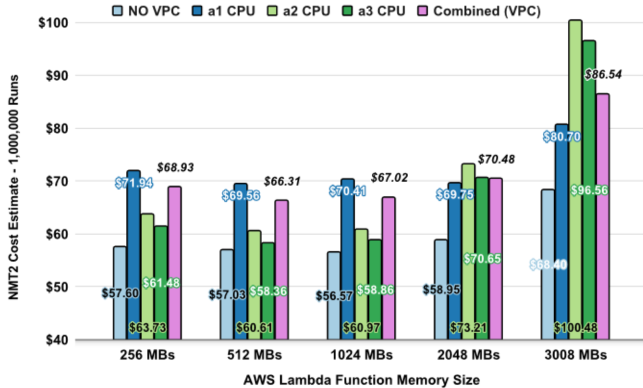

Fig. 4. Lambda hosting cost variation, NMT2 with CPU heterogeneity

B. FaaS Runtime Prediction

To investigate **RQ-2**, we estimate workload runtime with Linux CPU time accounting, by training individual regression models to predict specific CPU metrics (e.g. cpuUsr, cpuIdle) for FaaS deployments with alternate configurations (e.g. CPUs, memory, commercial FaaS platform). We convert a workload's resource utilization profile from one configuration to another and apply Linux CPU time accounting to generate runtime predictions. On FaaS platforms, resource utilization metrics obtained by SAAF originate from containers (IBM) or MicroVMs (AWS) and are generally isolated to report resource utilization of individual function executions. Linear regression and multiple regression can convert individual CPU metrics with high accuracy. We demonstrate linear regression of cpuUsr and cpuIdle from AWS Lambda in Figure 5. We captured CPU resource utilization metrics for single tenant, single thread invocations of our calcs service at different memory settings (256MB, 512MB, and 2048MB) on two different CPUs (Intel Xeon E5-2680v2 @ 2.8 GHz and E5-2686v4 @ 2.3 GHz). We scaled calculations from 80 to 120 million stepping by 400,000 without memory stress. Linear regression of cpuIdle time between the 256MB and 512MB deployments had a coefficient of determination of R^2=.988. Linear regression of cpuUsr time between the E5-2680v2 and E5-2686v4 CPUs at 2048MB had R^2=.974. The high predictability of individual CPU metrics enables high accuracy with our Linux time accounting approach to predict runtime.

We trained cpuUsr and cpuIdle models for source and target platforms described in Table IV and applied Linux time accounting to generate runtime predictions. The FaaS Runner automated data collection. Samples matching the desired source and targets were filtered using R scripts. We investigated the accuracy of our approach using successive workloads each introducing additional performance variance (SCNMT2→ SCMT2→SCSMT2). The CV for workload runtime across all CPU and memory configurations was 87.8%, 104.8%, and 114.9% for the respective workloads. Our objective is that each successive workload introduces more performance variance providing a greater challenge for performance prediction.

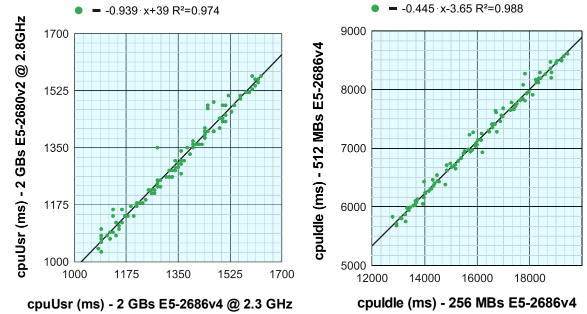

Fig. 5. cpuIdle & cpuUsr linear regression AWS Lambda

SCNMT2 provides a CPU-bound workload with a scaled number of random calculations from 30 to 60 million, leveraging 2 threads without memory stress. SCMT2 performs the same calculations, but adds fixed memory stress using large arrays for math operands as described in section III.C. Finally, SCSMT2 scales the array size from 1 to 1 million in steps of 100,000 to produce 10 different memory stress scenarios. As Lambda uses greedy function placement across VMs, most functions execute with the same VM function tenancy. For a workload of 100 concurrent function executions, approximately 91%, 96%, and 99% had identical tenancy of 256MB (13 tenants/VM), 512MB (6 tenants/VM), and 1024MB (3 tenants/VM). To simplify modeling, we used profiling data from function executions with maximum tenancy.

In total, we consider 77 different workload/configuration scenarios generating runtime predictions for: alternate CPU types (36), alternate memory settings (27), and alternate platforms (14). For all evaluations we associate source observations with actual target observations to establish ground truth by pairing samples in the same order function responses were returned by the FaaS platforms. This differs from sorting and pairing observations by runtime. An alternative is to obtain ground truth using Z-score normalization to compute a target value by projecting the source observation into the target distribution. This approach allows surrogate workloads to be substituted in performance models by eliminating the need to pair actual source and target observations, and we plan to adopt

this approach henceforth. Detailed prediction statistics for all workload/configuration scenarios, including average runtime of workloads, CV, RMSE, MAPE, mean absolute error (MAE), and degrees of freedom *are available online at: [59].*

Figures 6 and 7 depict the average % error of our runtime predictions for each workload and configuration. Degrees of freedom varied across tests because of the variable infrastructure received when executing FaaS workloads. For example, when executing a workload, we may randomly receive 70 a1 CPUs in one trial, and 47 a1 CPUs in another, resulting in different quantities of training data for different configurations. **The average error for SCNMT2 runtime predictions on different CPUs was just 0.51%, and with different memory settings 0.59%**. Predictions from the a1 to a3 CPU at 1024MBs had the highest average error at 1.53% and MAE of 62ms, less than the smallest FaaS billing increment.

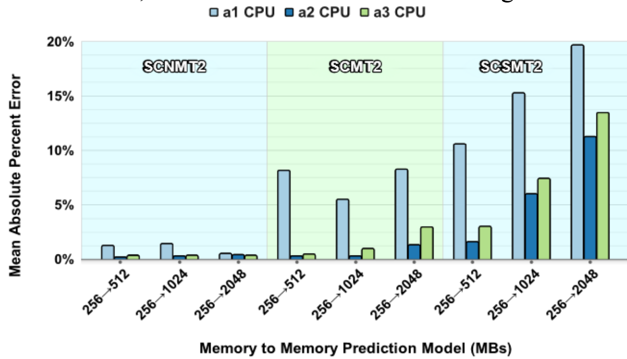

Fig. 6. Mean absolute percent error (MAPE) of memory to memory FaaS runtime prediction models.

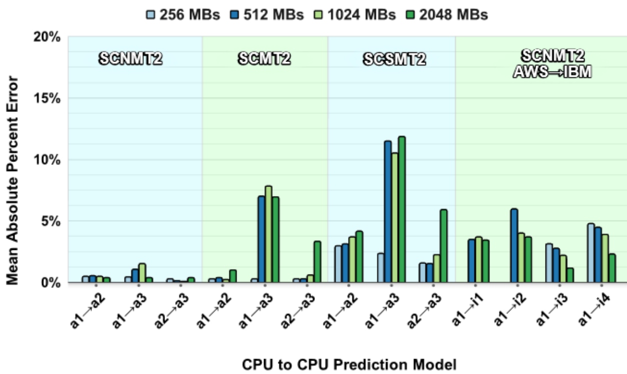

Fig. 7. Mean absolute percent error (MAPE) of CPU to CPU FaaS runtime prediction models.

For SCMT2 workloads that add *memory stress*, **the average error for runtime predictions for all configurations to different CPUs was 2.52%.** Adding memory stress increased runtime prediction errors ~5x. **The average error for SCMT2 runtime predictions to different memory settings was 3.83%,** an increase of ~6.5x over the SCNMT2 workload without memory stress. Predictions between the a1 and a3 to 512MB, 1024MB, and 2048MB had the highest MAPE at 7.29% producing MAE of 1.81s, 929ms, and 485ms for respective memory values. All other SCMT2 CPU predictions had far less error averaging just 0.96% MAPE. One million function invocations of our SCMT2 workload at 2048MB memory cost approximately $232.81. To put our runtime predictions into perspective for the SCMT2 workload, our worst case runtime error for a1→a3 CPU at 2048MB results in overestimating cost by $16.18, compared to average cost error of just $5.15 for all SCMT2 runtime predictions.

SCSMT2 workloads introduce *variable memory stress* resulting in an **average error for SCSMT2 runtime predictions to different CPUs of 5.10%.** With increasing memory stress, runtime predictions for SCSMT2 had about 2x more error than SCMT2 workloads. SCSMT2 has a higher CV than SCMT2 and SCNMT2. Additional memory stress made the SCSMT2 workload non-deterministic, increasing performance variance leading to more difficult runtime predictions, and less accuracy. We project our performance prediction error for the SCSMT2 runtime predictions to different CPUs with 1,000,000 functions calls in Figure 8. **The average error for SCSMT2 runtime predictions to different memory settings was 10.48%.** Test data for predicting SCSMT2 workload runtime to different memory settings also had the highest CV at 36%. Correspondingly, we observed a decrease in R^2 for our regression models from ~ .98 to .85.

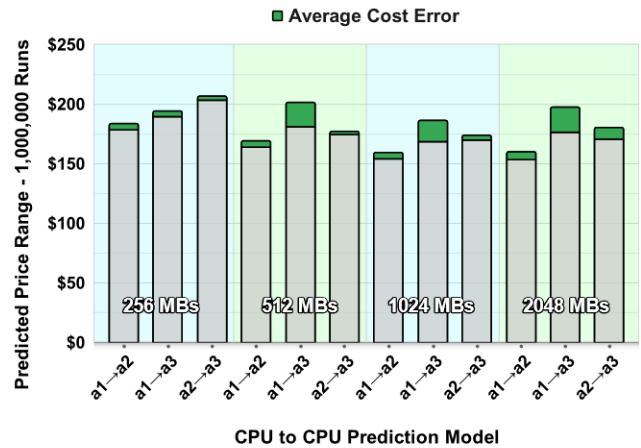

Fig. 8. Percent error of cost predictions for SCSMT2 derived from FaaS runtime prediction models

We evaluated our runtime predictions for SCNMT2 workloads without memory stress deployed on IBM Cloud Functions [55]. **The average error for IBM SCNMT2 runtime predictions to four different CPUs was 3.55%.** The occurrence rates for obtaining different IBM CPUs is described in Table V. Our prediction error equated to an average of 1.09s, 717ms, 287ms, and 120ms with 256, 512, 1024, and 2048MB. Average cost error of one million function invocations on IBM was $4.24 vs. an average workload cost of $119.38.

We observed that IBM shares VMs differently than AWS Lambda. Where AWS Lambda explicitly couples CPU timeshare to the memory reservation size, IBM does not adapt the CPU timeshare based on memory reservation size. IBM allows co-located function instances to compete for available CPU time on the VM. This allows users to obtain the best possible performance based on available resources, resulting in much higher performance variance. IBM Cloud Functions differed from AWS Lambda in that host VMs had 4 vCPUs and 16GB of RAM each. IBM function tenancy on each VM maxed out at: 32x256MB, 16x512MB, 8x1024MB, and 4x2048MB. This is in contrast to AWS Lambda max tenancy of: 13x256MB, 6x512MB, 3x1024MB, and 1x2048MB. On IBM, single tenant

executions of our NMT2 calcs service at 256MB required just 2091ms, where with full multitenancy performance slowed to 26,816ms, a slowdown of 12.82x. We observed performance degradation from multitenancy of 5.86x, 3.42x, and 1.69x at 512, 1024, and 2048MB. As configured, we estimate IBM Cloud Functions to be 63% more expensive than AWS Lambda to execute one million calcs functions with maximum VM function tenancy resulting from high concurrency. If functions execute sequentially however, IBM Cloud Functions completes the workload for just 13.4% the price of Lambda ($8.89) at 256MB. **The same workload on IBM can cost anywhere from $8.89 to $113.97 at 256MB depending on the tenancy of function executions across VMs driven by the concurrency of client requests.** On IBM, users benefit when FaaS workloads execute with low concurrency, and pay more when demand spikes. This provides an excellent example of pricing obfuscation on serverless platforms.

Table VI summarizes results of our model evaluations where each row summarizes all workload predictions to different target configurations (e.g. CPU, memory, or platform) we tested. Supported by SAAF, our models were trained to specifically account for CPU heterogeneity and function multi-tenancy to improve overall accuracy. **Across all scenarios, we calculated a MAPE of 3.49%, equaling a cost error of $6.46 for 1,000,000 function workloads costing an average of $150.45.**

TABLE VI. RUNTIME PREDICTION MODEL EVALUATION SUMMARY

Workload Prediction Type	Number of Models	Workload CV	MAPE	Average Cost Error	Average Workload Cost
SCNMT2 – CPU	12	21%	0.51%	$0.36	$70.27
SCMT2 – CPU	12	23%	2.52%	$5.15	$204.23
SCSMT2 – CPU	12	32%	5.10%	$8.86	$173.64
SCNMT2 – Memory	9	20%	0.59%	$0.45	$76.30
SCMT2 – Memory	9	22%	3.83%	$9.07	$236.88
SCSMT2 – Memory	9	36%	10.5%	$19.99	$190.80
SCNMT2 - IBM	14	18%	3.55%	$4.24	$119.38
Overall Average	***77 (sum)***	***--***	***3.49%***	***$6.46***	***$150.45***

C. Assessing Workload Predictions

To investigate heuristics for **RQ-3** we assessed statistical correlations between resource utilization metrics and absolute error of our runtime predictions. Our objective is to identify heuristics for different types of workloads that employ metric thresholds to signal when runtime predictions are likely to be error prone. We evaluated Pearson correlation coefficients between the absolute error of our runtime predictions and resource utilization metrics for our SCSMT2 workload tests. We evaluated correlations for configurations with minimum (CPU:a2→a3 1024MB, memory:256→1024 a3), median (CPU:a2→a3 512MB, memory:256→512 a1), and maximum (CPU:a1→a3 512MB, 256→2048, a1) prediction error. We ignored metric correlations that were expected to correlate with runtime: cpuUsr, cpuIdle, conswitches, # calcs, and array size.

For CPU predictions, a significant positive correlation was between cpuSteal and prediction absolute error (max: r=.18 p<.0001 df=506, median: r=.25 p<.05 df=70, min: r= n.s.). CpuSteal ticks are registered when a VM is ready to execute, but the physical CPU is busy servicing work from other co-located VMs sharing the physical host, or from the hypervisor itself [27]. CpuSteal introduces performance variance as workloads underperform for no apparent reason when the CPU is "stolen" by another VM. For the 12 SCSMT2 CPU and memory configurations in Figure 7, 4 had a statistically significant correlation between prediction error and cpuSteal.

For memory predictions, a significant negative correlation was between freeMemory and prediction absolute error (max: r=-.154 p<.01 df=324, median: r=-.19 p<.0001 df=506, min: r=-.25 p<.001 df=177). When VMs had less free memory, our predictions tended to be less accurate. Runtime predictions from 256→2048MB produced on average 4.3x more error than 256→512MB or 256→1024MB predictions. VM freeMemory also decreased with function memory size: at 256MB VMs had approximately ½ the freeMemory of VMs at 2048MB. We suspect that lower VM freeMemory results from co-located function instances. In conclusion, 9 memory configurations of SCSMT2 depicted in Figure, 5 had a statistically significant correlation between prediction error and VM freeMemory.

V. CONCLUSIONS

In this paper, we demonstrated how the Serverless Application Analytics Framework (SAAF) supported by the FaaS Runner tool can profile performance, resource utilization, and infrastructure of concurrent FaaS workloads. To dispel pricing obfuscation of serverless platforms, we leveraged Linux CPU time accounting principles and multiple regression to generate accurate FaaS function runtime predictions. FaaS hosting costs were then estimated by applying platform specific pricing policies. Research findings include: **RQ-1:** We characterized performance variance for identical CPU bound workloads on AWS Lambda and measured a 0.5% coefficient of variance (CV) for single tenant runs on identical CPUs. CV increased ~7.4x as a result of CPU heterogeneity, and another ~2.7x from multitenancy when function instances ran concurrently on the same host. **RQ-2:** Leveraging Linux time accounting, we predicted FaaS workload runtime across different CPUs, with different memory settings, and to different platforms using successive workloads that introduce additional performance variance. Mean absolute percentage error for predictions of 77 scenarios was 3.49%, equating to an average cost error of $6.46 against an average cost of $150.45 for one million function workloads. Prediction error for workloads without memory stress was approximately ~0.5%, with fixed memory stress ~3%, with variable memory stress ~7%, and for deployments to IBM Cloud Functions ~3.5%. **RQ-3:** CpuSteal was found to correlate with prediction error, while host VM freeMemory had a negative correlation.

ACKNOWLEDGMENTS

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

REFERENCES

[1] M. Yan, P. Castro, P. Cheng, and V. Ishakian, "Building a chatbot with serverless computing," in *Proceedings of the 1st International Workshop on Mashups of Things and APIs*, 2016, p. 5.

[2] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Serverless computation with openlambda," in *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*, 2016.

[3] I. Baldini *et al.*, "Serverless Computing: Current Trends and Open Problems," in *Research Advances in Cloud Computing*, 2017.

[4] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara, "Serverless computing: An investigation of factors influencing microservice performance," in *Proceedings - 2018 IEEE International Conference on Cloud Engineering, IC2E 2018*, 2018.

[5] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking Behind the Curtains of Serverless Platforms," *2018 USENIX Annu. Tech. Conf.*

(*USENIX ATC 18*), 2018.
[6] A. Sill, "The Design and Architecture of Microservices," *IEEE Cloud Comput.*, 2016.
[7] "Openwhisk common use cases." [Online]. Available: https://console.bluemix.net/docs/openwhisk/openwhisk_use_cases.html#openwhisk_common_use_cases.
[8] "Fn Project – The Container Native Serverless Framework." [Online]. Available: https://fnproject.io/.
[9] E. Oakes, L. Yang, K. Houck, T. Harter, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Pipsqueak: Lean Lambdas with Large Libraries," in *Proceedings - IEEE 37th International Conference on Distributed Computing Systems Workshops, ICDCSW 2017*, 2017.
[10] I. Baldini *et al.*, "The serverless trilemma: Function composition for serverless computing," in *Onward! 2017 - Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, co-located with SPLASH 2017*, 2017.
[11] A. Eivy, "Be Wary of the Economics of 'Serverless' Cloud Computing," *IEEE Cloud Comput.*, 2017.
[12] M. Fazio, A. Celesti, R. Ranjan, C. Liu, L. Chen, and M. Villari, "Open Issues in Scheduling Microservices in the Cloud," *IEEE Cloud Comput.*, 2016.
[13] M. Eisa, M. Younas, K. Basu, and H. Zhu, "Trends and directions in cloud service selection," in *Proceedings - 2016 IEEE Symposium on Service-Oriented System Engineering, SOSE 2016*, 2016.
[14] M. Eisa, M. Younas, and K. Basu, "Analysis and representation of QoS attributes in cloud service selection," in *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, 2018.
[15] "AWS Lambda Pricing Calculator." [Online]. Available: https://s3.amazonaws.com/lambda-tools/pricing-calculator.html.
[16] "Serverless Cost Calculator." [Online]. Available: http://serverlesscalc.com/.
[17] "[20] Servers.LOL – Serverless Cost Calculator for AWS Lambda – IOPipe." [Online]. Available: https://servers.lol/.
[18] "AWS Lambda - Serverless Compute." [Online]. Available: https://aws.amazon.com/lambda/.
[19] "Cloud Functions - Event-driven Serverless Computing." [Online]. Available: https://cloud.google.com/functions/.
[20] E. Jonas *et al.*, "Cloud programming simplified: a berkeley view on serverless computing," *arXiv Prepr. arXiv1902.03383*, 2019.
[21] Z. Ou *et al.*, "Is the Same Instance Type Created Equal? Exploiting Heterogeneity of Public Clouds," *IEEE Trans. Cloud Comput.*, vol. 1, pp. 201–214, 2013.
[22] B. Farley, A. Juels, V. Varadarajan, T. Ristenpart, K. D. Bowers, and M. M. Swift, "More for your money: Exploiting Performance Heterogeneity in Public Clouds," in *Proceedings of the Third ACM Symposium on Cloud Computing - SoCC '12*, 2012, pp. 1–14.
[23] M. S. Rehman and M. F. Sakr, "Initial findings for provisioning variation in cloud computing," in *Proceedings - 2nd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2010*, 2010, pp. 473–479.
[24] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: observing, analyzing, and reducing variance," *Proc. VLDB Endow.*, vol. 3, pp. 460–471, 2010.
[25] A. O. Ayodele, J. Rao, and T. E. Boult, "Performance Measurement and Interference Profiling in Multi-tenant Clouds," in *Proceedings - 2015 IEEE 8th International Conference on Cloud Computing, CLOUD 2015*, 2015.
[26] W. Lloyd, S. Pallickara, O. David, M. Arabi, and K. Rojas, "Mitigating resource contention and heterogeneity in public clouds for scientific modeling services," in *Proceedings - 2017 IEEE International Conference on Cloud Engineering, IC2E 2017*, 2017.
[27] W. Lloyd, M. Vu, B. Zhang, O. David, and G. Leavesley, "Improving application migration to serverless computing platforms: Latency mitigation with keep-Alive workloads," in *Proceedings - 11th IEEE/ACM International Conference on Utility and Cloud Computing Companion, UCC Companion 2018*, 2019.
[28] K. Wang and M. M. H. Khan, "Performance prediction for apache spark platform," in *Proceedings - 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security and 2015 IEEE 12th International Conference on Embedded Software and Systems, H*, 2015.
[29] J. White, M. Matalka, W. F. Fricke, and S. Angiuoli, "Cunningham: a BLAST Runtime Estimator," *Nat. Preced.*, 2011.
[30] A. Ganapathi *et al.*, "Predicting multiple metrics for queries: Better decisions enabled by machine learning," in *Proceedings - International Conference on Data Engineering*, 2009.
[31] A. Ganapathi, Y. Chen, A. Fox, R. Katz, and D. Patterson, "Statistics-driven workload modeling for the cloud," in *Proceedings - International Conference on Data Engineering*, 2010.
[32] M. Hafizhuddin Hilman, M. A. Rodriguez, and R. Buyya, "Task runtime prediction in scientific workflows using an online incremental learning approach," in *Proceedings - 11th IEEE/ACM International Conference on Utility and Cloud Computing, UCC 2018*, 2019.
[33] R. F. Da Silva, G. Juve, M. Rynge, E. Deelman, and M. Livny, "Online Task Resource Consumption Prediction for Scientific Workflows," in *Parallel Processing Letters*, 2015.
[34] T. P. Pham, J. J. Durillo, and T. Fahringer, "Predicting Workflow Task Execution Time in the Cloud using A Two-Stage Machine Learning Approach," *IEEE Transactions on Cloud Computing*, 2017.
[35] R. Ghosh, F. Longo, V. K. Naik, and K. S. Trivedi, "Modeling and performance analysis of large scale IaaS clouds," *Futur. Gener. Comput. Syst.*, 2013.
[36] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *Proceedings - International Conference on Distributed Computing Systems*, 2011, pp. 559–570.
[37] J. L. L. Simarro, R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, "Dynamic placement of virtual machines for cost optimization in multi-cloud environments," in *Proceedings of the 2011 International Conference on High Performance Computing and Simulation, HPCS 2011*, 2011, pp. 1–7.
[38] D. Villegas, A. Antoniou, S. M. Sadjadi, and A. Iosup, "An analysis of provisioning and allocation policies for infrastructure-as-a-service clouds," in *Proceedings - 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2012*, 2012, pp. 612–619.
[39] S. Yi, D. Kondo, and A. Andrzejak, "Reducing costs of spot instances via checkpointing in the Amazon Elastic Compute Cloud," in *Proceedings - 2010 IEEE 3rd International Conference on Cloud Computing, CLOUD 2010*, 2010, pp. 236–243.
[40] A. Andrzejak, D. Kondo, and S. Yi, "Decision Model for Cloud Computing under SLA Constraints," in *IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2010, pp. 257–266.
[41] Q. Zhang, Q. Zhu, and R. Boutaba, "Dynamic Resource Allocation for Spot Markets in Cloud Computing Environments," *2011 Fourth IEEE Int. Conf. Util. Cloud Comput.*, pp. 178–185, 2011.
[42] J. Spillner, C. Mateos, and D. A. Monge, "Faaster, better, cheaper: the prospect of serverless scientific computing and HPC," in *Communications in Computer and Information Science*, 2018.
[43] M. Malawski, A. Gajek, A. Zima, B. Balis, and K. Figiela, "Serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions," *Future Generation Computer Systems*, 2017.
[44] Q. Jiang, Y. C. Lee, and A. Y. Zomaya, "Serverless execution of scientific workflows," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2017.
[45] M. Malawski, K. Figiela, A. Gajek, and A. Zima, "Benchmarking heterogeneous cloud functions," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018.
[46] V. Ishakian, V. Muthusamy, and A. Slominski, "Serving deep learning models in a serverless platform," in *Proceedings - 2018 IEEE International Conference on Cloud Engineering, IC2E 2018*, 2018.
[47] A. Bhattacharjee, A. D. Chhokra, Z. Kang, H. Sun, A. Gokhale, and G. Karsai, "BARISTA: Efficient and Scalable Serverless Serving System for Deep Learning Prediction Services," in *2019 IEEE International Conference on Cloud Engineering (IC2E)*, 2019, pp. 23–33.
[48] M. Fotouhi, D. Chen, and W. J. Lloyd, "Function-as-a-Service Application Service Composition: Implications for a Natural Language Processing Application," in *Proceedings of the 5th International Workshop on Serverless Computing*, 2019, pp. 49–54.
[49] L. Feng, P. Kudva, D. Da Silva, and J. Hu, "Exploring Serverless Computing for Neural Network Training," in *IEEE International Conference on Cloud Computing, CLOUD*, 2018.
[50] E. F. Boza, C. L. Abad, M. Villavicencio, S. Quimba, and J. A. Plaza, "Reserved, on demand or serverless: Model-based simulations for cloud budget planning," in *2017 IEEE 2nd Ecuador Technical Chapters Meeting, ETCM 2017*, 2018.
[51] M. Villamizar *et al.*, "Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures," in *Proceedings - 2016 16th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2016*, 2016.
[52] L. F. A. Jr, F. S. Ferraz, R. F. A. P. Oliveira, and S. M. L. Galdino, "Function-as-a-Service X Platform-as-a-Service : Towards a Comparative Study on FaaS and PaaS," *Twelfth Int. Conf. Softw. Eng. Adv. Funct.*, 2017.
[53] "SAAF: Serverless Application Analytics Framework." [Online]. Available: https://github.com/wlloyduw/SAAF.
[54] "Azure Functions - Develop Faster with Serverless Compute." [Online]. Available: https://azure.microsoft.com/en-us/services/functions/.
[55] "IBM Cloud Functions." [Online]. Available: https://cloud.ibm.com/functions/.
[56] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, B. Smith, and R. H. Katz, "Selecting the best VM across multiple public clouds: A data-driven performance modeling approach," in *SoCC 2017 - Proceedings of the 2017 Symposium on Cloud Computing*, 2017.
[57] W. J. Lloyd *et al.*, "Demystifying the Clouds: Harnessing Resource Utilization Models for Cost Effective Infrastructure Alternatives," *IEEE Trans. Cloud Comput.*, vol. 5, no. 4, pp. 667–680, 2015.
[58] "Open source and enterprise-ready professional software for data science - RStudio." [Online]. Available: https://www.rstudio.com/.
[59] "The Serverless Application Analytics Framework: Performance Modeling." [Online]. Available: https://github.com/wlloyduw/SAAF/blob/master/perfmodel.pdf.