Characterizing X86 and ARM Serverless Performance Variation: A Natural Language Processing Case Study

Danielle Lambion*
University of Washington
Tacoma, Washington, USA
dlambion@uw.edu

Robert Schmitz*
University of Washington
Tacoma, Washington, USA
rgs1@uw.edu

Robert Cordingly
University of Washington
Tacoma, Washington, USA
rcording@uw.edu

Navid Heydari
University of Washington
Tacoma, Washington, USA
navidh2@uw.edu

Wes Lloyd
University of Washington
Tacoma, Washington, USA
wlloyd@uw.edu

ABSTRACT

In this paper, we leverage a Natural Language Processing (NLP) pipeline for topic modeling consisting of three functions for data preprocessing, model training, and inferencing to analyze serverless platform performance variation. Specifically, we investigated performance using x86_64 and ARM64 processors over a 24-hour day starting at midnight local time on four cloud regions across three continents on AWS Lambda. We identified public cloud resource contention by leveraging the CPU steal metric, and examined relationships to NLP pipeline runtime. Intel x86_64 Xeon processors at the same clock rate as ARM64 processors (Graviton 2) were more than 23% faster for model training, but ARM64 processors were faster for data preprocessing and inferencing. Use of the Intel x86_64 architecture for the NLP pipeline was up to 33.4% more expensive than ARM64 as a result of incentivized pricing from the cloud provider and slower pipeline runtime due to greater resource contention for Intel processors.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**; • **General and reference → Performance**.

KEYWORDS

Function-as-a-Service, Topic Modeling, Performance Variation, Resource Contention, Serverless Computing

ACM Reference Format:
Danielle Lambion, Robert Schmitz, Robert Cordingly, Navid Heydari, and Wes Lloyd. 2022. Characterizing X86 and ARM Serverless Performance Variation: A Natural Language Processing Case Study . In *Companion of the 2022 ACM/SPEC International Conference on Performance Engineering (ICPE '22 Companion), April 9–13, 2022, Bejing, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3491204.3543506

1 INTRODUCTION

Serverless Function-as-a-Service (FaaS) platforms offer an attractive cloud platform for hosting computational workloads integrating support for desirable features including high availability, fault tolerance, and automatic scaling. As the popularity of these platforms has grown, cloud providers have continued to refine and enhance their features to address open problems and support additional capabilities [10].

Serverless FaaS platforms abstract many of the configuration and management details from end users to simplify their use. This abstraction, however, comes with the cost of reduced observability necessary to conduct root-cause analysis to troubleshoot performance issues and explain sources of performance variation. For example, on the AWS Lambda serverless FaaS platform, the underlying processor type and number of shared tenants sharing a physical host is not disclosed. Processor architecture and multitenancy, where multiple users or workloads share the same host causing resource contention, have been shown as key factors responsible for performance variation of software deployments in the cloud [4, 9, 16, 17].

In this paper, we investigate the use of two new features offered by AWS Lambda to examine the performance variation for hosting a natural language processing (NLP) pipeline. First we investigate support for multiple CPU architectures on AWS Lambda. Developers can now choose to deploy functions to Intel Xeon x86_64 or Graviton2 ARM64 processors. Secondly, we harness the ability to deploy functions with container images up to 10GB to more easily package and deploy large applications. Leveraging our NLP pipeline as a use case, we investigated performance variation for a 24-hour day across four cloud regions and two CPU architectures using experiments coordinated to run from midnight to midnight in each local timezone.

While the number of tenants sharing cloud servers is often obscured, the cpuSteal metric, available from the Linux procfs has been shown to indicate resource contention which typically correlates with workload performance degradation [4, 7, 14]. CpuSteal ticks are registered when a virtual machine (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. In this paper, we examine performance over 24 hours across multiple regions to investigate relationships between cpuSteal and serverless FaaS function performance.

*Both authors contributed equally to this research.

ICPE '22 Companion, April 9–13, 2022, Bejing, China

ACM ISBN 978-1-4503-9159-7/22/04.
https://doi.org/10.1145/3491204.3543506

1.1 Research Questions

This paper investigates the following research questions:

RQ-1: (Architecture Performance) What are the performance and cost implications of adopting the ARM64 CPU architecture vs. x86_64 Intel for running a multi-step NLP pipeline on a commercial serverless FaaS platform?

RQ-2: (Performance Variation) What performance variation results from the use of alternative cloud regions over a 24-hour day where the state of resource contention is likely to change to host a multi-step NLP pipeline on a commercial serverless FaaS platform?

1.2 Contributions

This paper provides the following research contributions:

(1) We investigate performance of ARM64 vs x86_64 architectures for hosting an NLP pipeline on a commercial FaaS platform. Our results indicate that current performance improvements observed on ARM64 processors are the result of lower resource contention for ARM64 serverless infrastructure.
(2) We investigate performance trends for a 24-hour workday for hosting an NLP pipeline across four cloud regions in three continents. Severless functions executing during regular business hours were shown to have slower runtimes than those executing off hours leading to slightly higher costs.
(3) Leveraging an NLP case study, we examine the utility of the cpuSteal metric to detect resource contention on a commercial FaaS platform. CpuSteal is shown to correlate with NLP pipeline runtime across four cloud regions over a 24-hour day.

2 BACKGROUND

On serverless FaaS platforms Jonas et al. identified heterogeneous CPUs and noted their potential to complicate performance modeling in [10]. Wang et al. identified heterogeneous VM types on FaaS platforms from AWS, Azure, and Google in [23]. 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). In [4], distinct ratios of heterogeneous CPU types were identified and then used to create performance models capable of accounting for performance variation caused by these heterogeneous CPUs on AWS Lambda and IBM Cloud Functions. These efforts, however, did not evaluate performance implications of alternate CPU architectures, i.e. ARM64 vs. x86_64.

Previous research has identified how provisioning variation results in varying degrees of function tenancy and consequently resource contention on FaaS platforms [4, 15, 23]. We identified how the number of function "tenants" on hosts, called "function instances" by Wang, increased when scaling up the number of concurrent requests on AWS Lambda [15]. Conversely, increasing function memory reduced the number of tenants on a host. Wang observed that function instance placement across hosts on AWS Lambda used greedy placement, where concurrent requests are packed onto individual hosts until available memory is exhausted. Multiple functions from a single user account were found to share hosts, but hosts did not appear to be shared with other users before the adoption of the Firecracker MicroVM on the platform in 2019 [1]. On Azure Functions, the maximum observed tenancy of function executions was reported to 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 or the use of the cpuSteal metric to do so.

Schad in 2010 contributed an extensive evaluation of performance variation of IaaS and object storage platforms on Amazon EC2 examining instance startup, CPU, memory, disk I/O, and network I/O performance over multiple months [20]. These early results were before various advancements that significantly reduced performance overhead of virtual machines [8]. Later Leitner and Cito measured mean relative standard deviations for CPU, disk I/O, and memory benchmarks on VMs deployed to Amazon, Google, Azure, and IBM clouds for 72 hour periods identifying performance variation from CPU heterogeneity [13]. Uta and Obaseki followed by emulating network bandwidth variability results from Ballani et al. to investigate performance implications for six real world big data workloads [2, 22]. Their work used a private IaaS cloud to evaluate performance implications of network bandwidth limitations that may occur due to resource contention.

Ginzburg and Freedman investigated diurnal patters in function performance on the AWS Lambda serverless FaaS platform [7]. Their work primarily focused on evaluating performance variation within a single region, us-east-1 (Virginia) over 1-week. They did examine multi-region performance variation by comparing performance of two regions: ap-northeast-2 and us-east-1. Their reported results however were not extensive in that they only identified how a cache benchmark exhibited a 11% end-to-end performance difference, and that us-east-1 had consistently better networking performance. More importantly their work identified a relationship between function performance and cpuSteal reporting an R^2 between 0.4 and 0.6 varying diurnally. The authors demonstrated how better performing function instances could be found and then exploited using short-running performance tests. This approach yielded a cost savings from 2 to 8%. Additionally, the authors suggested that function instances across different cloud regions have potential to be exploited by leveraging favorable diurnal patterns for cost savings.

We extend on Ginzburg's work by examining FaaS performance variation for a multi-function NLP pipeline across four regions in three continents. In particular, we examined the performance implications of the ARM vs Intel CPU architectures diurnally while correlating observations with cpuSteal as a proxy for resource contention. Our results reinforce the utility of cpuSteal as a metric to infer FaaS resource contention while adding insights regarding alternate CPU Architectures and performance across more regions for a robust compute-bound NLP pipeline. Additionally, our case study exercised new Docker container support capabilities and our experiments were performed on AWS Lambda after the maximum function memory size was increased from 3GB to 10GB, a feature which could exacerbate resource allocation and management.

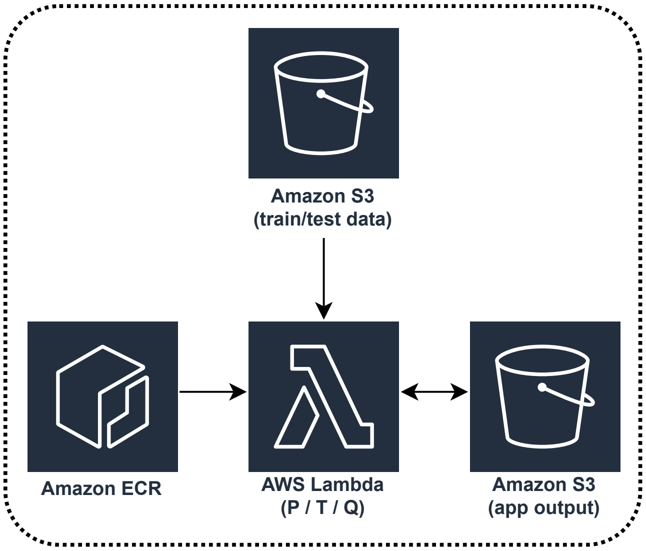

Figure 1: Switchboard style application design using a single AWS Lambda function.

3 METHODOLOGY

3.1 Natural Language Processing Case Study

To investigate FaaS performance variation, we leveraged a Natural Language Processing (NLP) application use case for topic modeling. Our application consisted of three compute-bound functions implemented in Python and deployed to AWS Lambda using the platform's new support (as of Dec 2020) for deploying a container image to augment read-only storage capacity [18]. Our NLP pipeline use case is interesting to study because it provides a compute intensive workflow where every function in the application requires several minutes of processing, enabling us to examine performance metrics over long periods. Our functions were single threaded and largely consumed a single vCPU's entire time share during execution as shown in figure 2. We focused our case study on examining performance variation for a compute bound application as AWS Lambda is primarily considered a compute platform where performance of typical workloads is not disk or network I/O bound. The source code for our application can be found here [21].

Our NLP use case is a topic modeling application for news headlines developed in Python3 using the Natural Language Toolkit (NLTK) and Gensim libraries [3, 19]. We used a dataset of approximately 1.1 million news headlines from Australia's ABC News [12]. This application is composed of three separate functions. Our three functions stored intermediate workflow data in an Amazon Simple Storage Service (S3) bucket. The dataset consisted of a single 63MB CSV file where each sample contains a news article headline and its publication date. News headlines were split into a training and testing dataset and stored in another S3 bucket.

The preprocessing (P) function loaded and prepared data for model training. This step removed stopwords and tokenized strings. The word tokens were then stemmed and lemmatized. A dictionary was created from the processed token words and was used to create a frequency-inverse document frequency (TF-IDF) model. The dictionary and the TF-IDF model from preprocessing were stored in the intermediate S3 bucket and provided as input for function 2.

The training (T) function trained a latent Dirichlet allocation (LDA) topic model using the output from the data preprocessing step. The trained model was cached in the intermediate S3 bucket to be queried by future users by function 3.

The query (Q) function queried the model for topics with new headlines. This step used headlines as strings provided in a CSV file from the testing dataset. These strings were tokenized, lemmatized, and word stemmed. The model generated in the training step was queried with the processed token words from the query headlines. This step outputs the input CSV file with an appended column containing topics obtained from the provided news headlines and a score rating the quality of topic match to the headline.

3.2 Serverless Application Implementation

The application was packaged into a Docker container image along with the Serverless Application Analytics Framework (SAAF) [4] to collect profiling data when deployed to AWS Lambda. A Docker image was used as the underlying package for deploying the code to AWS Lambda due to the limited file size of deploying Lambda functions using zip files (i.e. 50MB compressed and 250MB uncompressed) as Docker container images can be up to 10GB [18]. Docker images were deployed to the AWS Elastic Container Registry (ECR). Compilation of Docker images for both ARM64 and x86_64 was performed on an AWS EC2 spot-instance c5d.large (x86_64) running Ubuntu 20.04 using Docker's buildx CLI add-on along with QEMU for non-native architecture support [11].

Each step in the application pipeline could be called independently by sending JSON data specifying the function to be executed. A single Lambda function deployment was created for each CPU architecture and used to run the NLP pipeline and obtain performance metrics. Each Lambda function was configured to have a 10 minute timeout due to the long processing time needed for each step in the application. The deployed Lambda functions were each allocated 2560MB (2.5GB) of memory as the application's P function required a large amount of memory. An Identity and Access Management (IAM) role was attached to each Lambda function with an Amazon S3 access policy to allow for data retrieval and creation using AWS S3 buckets.

An S3 bucket was created for each Lambda function to store all output files from each function execution. Additional S3 buckets were created for each region to store training and testing data. The S3 buckets and Docker container images were co-located in the same regions as their dependent Lambda functions to minimize resource cost and network latency for each case study.

To minimize function cold starts, we leveraged the switchboard architecture design pattern depicted in Figure 1 [5]. Code for three separate functions was combined into a single Lambda function. A JSON payload parameter was used to determine which code path to execute. By adopting the switchboard design which combines all of our functions into a single deployment package, the initial call to the pipeline's P function initializes the aggregate function's runtime environment, known as a function instance. This function instance is then available and pre-warmed to execute the T function and Q function without cold start initialization. This design will increase the total number of function warm starts to effectively decrease the overall runtime of the application to enable cost reductions due to the sequential nature of the NLP pipeline.

3.3 Experimental Approach

Four regions were selected for our case studies: Asia Pacific: Tokyo (ap-northeast-1), Europe: Frankfort (eu-central-1), US East: Ohio (us-east-2), and US West: Oregon (us-west-2). These regions were selected to provide a variety of timezones across three continents. In each region, we deployed a c6gd.large (arm64) AWS EC2 spot instance which provided 2 vCPUs and 4 GB memory for use as a client to run experiments and collect results.

A (Bash) shell script was used as a wrapper for the AWS CLI to execute each application step in series (i.e. preprocessing, training, query) and to collect the container metrics returned from SAAF with a two second delay between function calls. The shell script removed any output file artifacts from the S3 buckets between runs. A Linux cron job was used to schedule the start of the experiments across all four regions. We measured performance variation for a 24-hour day starting at 12:00 a.m. local time in each region. Profiling metrics collected during testing were measured from the time the Lambda function started executing the FaaS function’s code until the function completed. Our runtime metrics measured server-side runtime of the FaaS functions code therefore excluding network latency between the client and server and any infrastructure initialization that occurs before the FaaS function executes. We note that even though infrastructure initialization time was not captured in the runtime measurement, cold function runtime is slower than warm function runtime due to the unfavorable state of memory caches when first executing unseen code. We performed approximately 112 pipeline executions for each CPU architecture/region pair consisting of 3 function calls. Overall we experienced approximately 3.86% function cold starts while continuously running eight instances of the pipeline for 24 hours.

4 RESULTS

4.1 NLP Pipeline Function Comparison

We profiled each of our NLP pipeline functions with SAAF to evaluate differences in computational requirements using the us-east-2 (Ohio) region. We measured the maximum function memory utilization by testing each function separately by making a cold function call. In addition we profiled runtime, the average number of memory page faults per minute, and the average CPU context switches per minute on x86_64 vs. ARM64 as shown in Table 1. In us-east-2, the ARM64 architecture provided faster runtime for the P and Q functions (7.3% and 8.9%), while x86_64 provided faster runtime for the T function (23.6%). Figure 2 depicts the average values for Linux CPU profiling metrics for each function for x86_64 versus ARM64. Values are averages for observations from us-east-2. None of the functions exhibited significant CPU kernel, I/O wait, or interrupt service time.

Table 1: NLP pipeline function comparison - us-east-2

Function Architecture	P ARM64	P x86_64	T ARM64	T x86_64	Q ARM64	Q x86_64
pageFaults/min	159638	148821	37368	45973	4656	3964
contextSwitch/min	9494	8174	1016	1151	933	982
max memory	1954	1947	1009	1014	377	477
runtime (s)	159.21	170.45	225	182	361	392.3

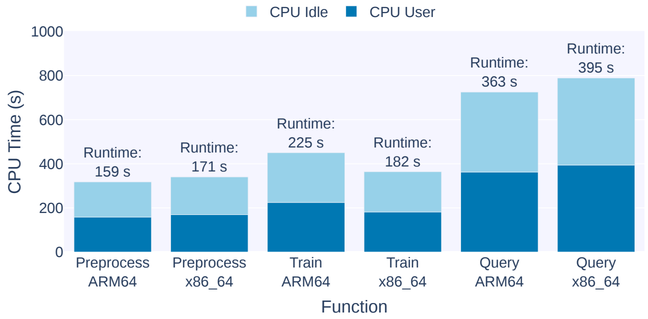

Figure 2: CPU time profile for the NLP pipeline

4.2 Performance Implications of CPU Architecture

To investigate **RQ-1**, we executed our NLP pipeline for 24 hours across four AWS regions and calculated global average runtime metrics as shown in Table 2. While the global minimum x86_64 Intel Xeon (2.5 GHz) performance was 15% faster than ARM, Intel global average runtime was 1.7% slower (p=1.0725e-06). This performance improvement provided by ARM64, however, appears to be the result of lower resource contention for ARM64 CPUs on AWS Lambda. To investigate this resource contention, we leveraged the cpuSteal metric as in [7, 14].

While we presume that resource contention for ARM based resources on AWS Lambda to be quite low, when we inspected ARM64 cpuSteal, only 0 values were reported by the Linux procfs. We hypothesized that cpuSteal may not be reported by Graviton 2 processors. To verify, we launched a c6g ec2 dedicated host based on the Graviton 2 ARM64 processor and placed 2 x c6g.8xlarge 32 vCPUs VMs on it to guarantee VM multi-tenancy. We then ran the stress tool to generate CPU contention by overprovisioning the tool to exercise 64 cores on both VMs for several minutes. The cpuSteal metric in the /proc/stat file remained at 0 for the duration of the test. We believe that cpuSteal may not occur with Graviton 2 processors because unlike Intel Xeons, Graviton 2 does not support hyperthreading, and all CPU cores are physical cores [6]. This suggests that both EC2 and Lambda do not overprovision Graviton 2 processors when placing VMs and functions, a decision that should lead to lower performance variation in the cloud as CPU cores are not simultaneously shared.

Across all regions, we observed that NLP pipeline runtime on ARM CPUs exhibited a Coefficient of Variation (CV) of only 2.47% compared to 6.89% on x86_64 which was 2.79x greater. The runtime spread, the difference between maximum and minimum runtime, was 3.14x greater on Intel than ARM confirming that ARM offered more stable performance. These performance differences led to

Table 2: CPU architecture runtime comparison - all regions

metric	arm64(s)	arm64 (%intel)	x86_64(s)	x86_64 (%arm)
min runtime	692.59	115.64	598.9	86.47
max runtime	799.5	85.40	936.2	117.10
avg runtime	735.07	98.31	747.74	101.72
runtime spread	106.91	31.70	337.3	315.50
stdev runtime	18.15	35.24	51.51	283.80
CV (%)	2.47	35.85	6.89	278.95
cost-10k runs	$245.02	78.64	$311.56	127.15

Table 3: CPU steal across AWS regions for x86_64

metric/region	us-east-2	us-west-2	eu-central-1	ap-northeast-1
avg cpuSteal/min	8.89	18.26	4.24	4.79
% of eu-central-1	209.7	431.6	100.0	113.0
R^2 runtime	0.618	0.379	0.427	0.39
Pearson (r)	0.7861	0.6157	0.6537	0.6249

substantial cost savings running our pipeline using the ARM64 architecture. AWS heavily discounts the use of their ARM CPUs so that Intel CPU time costs 20% more than ARM on Lambda. This discount combined with slightly slower Intel performance, results in x86_64 pipeline executions costing 27.15% more for an estimated 10,000 runs.

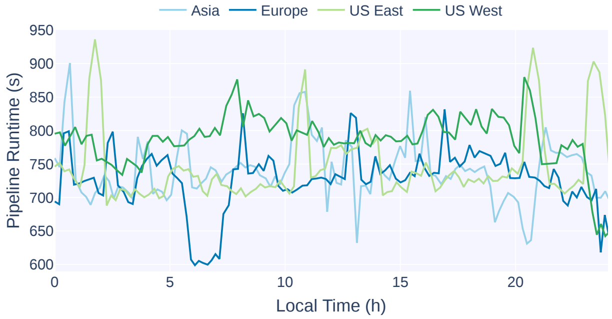

Figure 3: x86_64 CPU architecture runtimes in four regions.

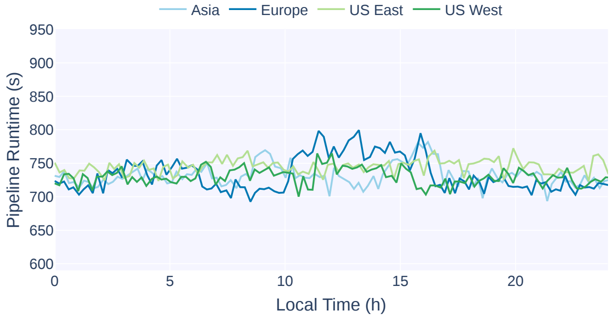

Figure 4: ARM64 CPU architecture runtimes in four regions.

4.3 24-Hour Global Performance Variation

To investigate performance variation of our NLP pipeline in support of **RQ-2**, we executed our pipeline continuously over the course of a regular business day across four regions on three continents on both the x86_64 and ARM64 CPU architectures. Observations were obtained on Wednesday, January 19, 2022 starting at 12:00 a.m. local time in each region. Global 24-hour performance on Intel x86_64 processors is shown in Figure 3 and on ARM64 processors in Figure 4. We have used the same y-axis scale on both figures to highlight the differences between performance variation on x86_64 vs. ARM64.

Over a 24-hour day we found that NLP pipeline performance on x86_64 processors in us-west-2 had significantly slower performance (4.3%) than average Intel performance across all regions (p=1.6-e09). Conversely, NLP pipeline performance on x86_64 processors in eu-central-1 had significantly faster performance (2.9%) than average Intel performance across all regions (p=.00002). For ARM64 processors, NLP pipeline performance on ARM in us-east-2 had significantly slower performance (1.4%) than other regions for ARM (p=1.5e-13). Conversely, NLP pipeline performance on ARM in us-west-2 had significantly faster performance (0.7%) than other regions for ARM (p=.0007). The fastest Intel region, eu-central-1 was on average faster than all of the ARM regions. This suggests that when Intel CPUs had slower runtime the major contributing factor was resource contention. To confirm this, we compared the average cpuSteal ticks per minute in us-west-2 (slowest region) to ticks in eu-central-1 (fastest region). We observed average cpuSteal per minute of 18.26 in us-west-2, but only 4.24 cpuSteal per minute in eu-central-1, which is approximately 4.3x more cpuSteal (slowest vs. fastest region). We summarize cpuSteal statistics in Table 3. We found that cpuSteal predicted pipeline runtime with R^2 from .38 to .62 which corroborates with earlier AWS Lambda measurements [7]. We depict the global linear regression between average cpuSteal per minute with NLP pipeline runtime in Figure 5. The graph shows how longer runtimes on the right side of the graph corresponded with higher cpuSteal. Figure 6 depicts how cpuSteal varied over a full 24-hour day across AWS regions. While us-east-2 had larger spikes indicating more severe resource contention, us-west-2 had consistently higher cpuSteal. Comparatively, cpuSteal in eu-central-1 and ap-northeast-1 is relatively low.

The average performance difference between the us-west-2 (slowest) and eu-central-1 (fastest) regions using the x86_64 architecture resulted in increased costs of 7.37%, similar to a ’sales tax’ rate. We contrast the difference in performance between x86_64 and ARM64 in the us-west-2 region in Figure 7. In this region, x86_64 had the slowest observed Intel performance, and ARM64 had the fastest observed Graviton 2 performance of all regions tested. The average pipeline runtime differential was approximately 50 seconds or about 6.86%. Given that AWS charges approximately 25% more for Lambda functions run on Intel processors, this produced our largest cost differential of 33.36%, where 100,000 pipeline executions would cost approximately $974 more on Intel than on ARM64.

Finally, we used a 2-hour sliding window and calculated global average NLP pipeline performance for both CPU architectures. We display these 2-hour averages normalized to global average ARM pipeline runtime in Figure 8. We observed that NLP pipeline

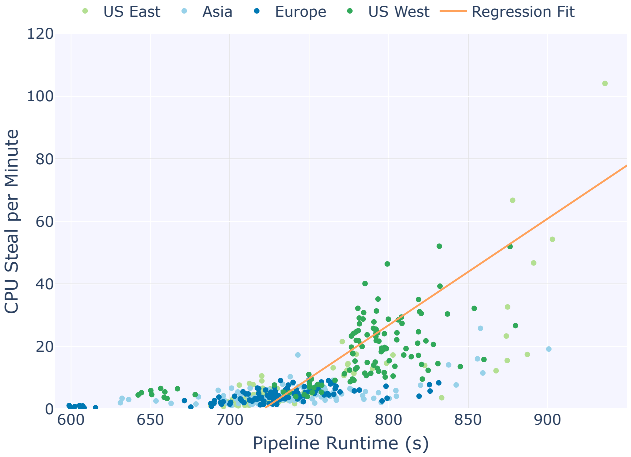

Figure 5: x86_64 CPU steal linear regression with runtime

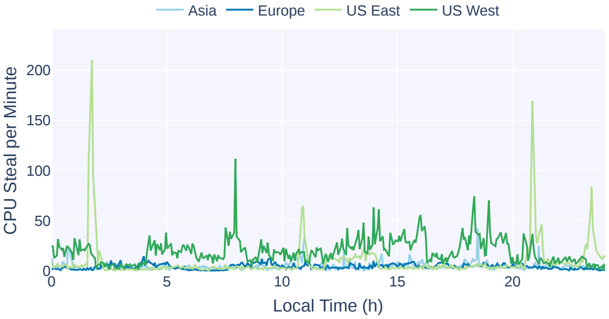

Figure 6: CPU architecture x86_64 CPU steal time in four regions.

runtime on Intel processors from 10:00 a.m. to 12:00 p.m. was 6% slower in contrast to 6:00 a.m. to 8:00 a.m., and 3.3% slower on ARM processors from 2:00 p.m. to 4:00 p.m. vs. 12:00 a.m. to 2:00 a.m. Figure 8 shows that NLP pipeline runtime tended to be faster outside regular business hours on any CPU architecture and region for our 24-hour study.

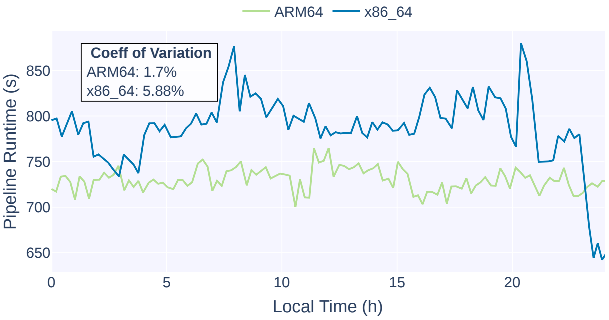

Figure 7: CPU architecture x86_64 versus ARM64 container runtimes in US West, Oregon.

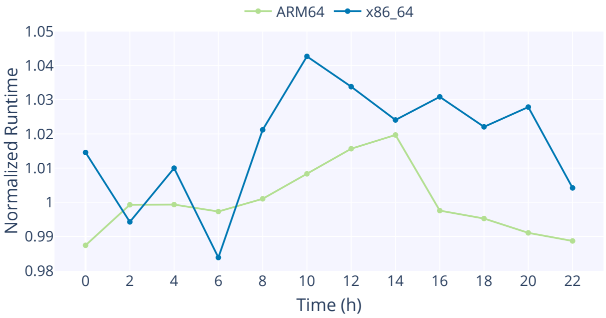

Figure 8: Global two-hour average NLP pipeline runtime normalized against global ARM64 average runtime.

5 CONCLUSIONS

Our study has helped identify performance differences of the ARM64 vs. x86_64 architectures on AWS Lambda while identifying diurnal performance patterns globally. The ARM64 architecture provided faster and more consistent runtime performance on average than the x86_64 architecture.

We summarize our findings with respect to the research questions as follows:

(RQ-1): Researchers and practitioners should be encouraged to adopt the ARM64 architecture on AWS Lambda when possible due not only to the discounted cost, but also because higher resource contention for x86_64 resources further exacerbates the cost differential. ARM64 cost savings, however, may only be temporary if the discount drives more users to adopt this architecture. We demonstrated up to **33.4% cost savings** for our NLP pipeline by leveraging ARM64 CPUs vs. x86_64 CPUs in us-west-2, the region exhibiting the highest resource contention.

(RQ-2): Researchers and practitioners running non-latency sensitive workloads may consider redirecting their workloads to leverage regions outside regular business hours. For example, we observed a **6% global average runtime differential** across four regions from 6:00 to 8:00 a.m. vs. 10:00 to 12:00 p.m.

ACKNOWLEDGMENTS

This research is supported by the NSF Advanced Cyberinfrastructure Research Program (OAC-1849970), National Institutes of Health (NIH) National Institute of General Medical Sciences (NGMS) grant 5R01GM126019, and the AWS Cloud Credits for Research program.

REFERENCES

[1] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. 2020. Firecracker: Lightweight Virtualization for Serverless Applications. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 419–434. https://www.usenix.org/conference/nsdi20/presentation/agache

[2] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. 2011. Towards Predictable Datacenter Networks. In *Proceedings of the ACM SIGCOMM 2011 Conference* (Toronto, Ontario, Canada) *(SIGCOMM '11)*. Association for Computing Machinery, New York, NY, USA, 242–253. https://doi.org/10.1145/2018436.2018465

[3] Steven Bird, Edward Loper, and Ewan Klein. 2009. Natural language processing with Python.

[4] Robert Cordingly, Wen Shu, and Wes J. Lloyd. 2020. Predicting Performance and Cost of Serverless Computing Functions with SAAF. In *2020 IEEE Intl Conf on*

Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech). IEEE, 640–649. https://doi.org/10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00111

[5] Mohammadbagher Fotouhi, Derek Chen, and Wes J. Lloyd. 2019. Function-as-a-Service Application Service Composition: Implications for a Natural Language Processing Application. In *Proceedings of the 5th International Workshop on Serverless Computing* (Davis, CA, USA) *(WOSC '19)*. Association for Computing Machinery, New York, NY, USA, 49–54. https://doi.org/10.1145/3366623.3368141

[6] Andrei Frumusanu. 2020. Amazon's arm-based Graviton2 against AMD and Intel: Comparing cloud compute. https://www.anandtech.com/show/15578/cloud-clash-amazon-graviton2-arm-against-intel-and-amd

[7] Samuel Ginzburg and Michael J. Freedman. 2020. Serverless Isn't Server-Less: Measuring and Exploiting Resource Variability on Cloud FaaS Platforms. In *Proceedings of the 2020 Sixth International Workshop on Serverless Computing* (Delft, Netherlands) *(WoSC'20)*. Association for Computing Machinery, New York, NY, USA, 43–48. https://doi.org/10.1145/3429880.3430099

[8] Brendan Gregg. 2017. AWS EC2 Virtualization 2017: Introducing Nitro. https://www.brendangregg.com/blog/2017-11-29/aws-ec2-virtualization-2017.html. Accessed: 2022-01-25.

[9] Xinlei Han, Raymond Schooley, Delvin Mackenzie, Olaf David, and Wes J Lloyd. 2020. Characterizing public cloud resource contention to support virtual machine co-residency prediction. In *2020 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, IEEE, 162–172.

[10] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, Joseph E. Gonzalez, Raluca Ada Popa, Ion Stoica, and David A. Patterson. 2019. Cloud Programming Simplified: A Berkeley View on Serverless Computing. https://doi.org/10.48550/ARXIV.1902.03383

[11] Artur Klauser. 2020. Building Multi-Architecture Docker Images With Buildx. https://medium.com/@artur.klauser/building-multi-architecture-docker-images-with-buildx-27d80f7e2408

[12] Rohit Kulkarni. 2017. A Million News Headlines. https://www.kaggle.com/therohk/million-headlines.

[13] Philipp Leitner and Jürgen Cito. 2016. Patterns in the chaos—a study of performance variation and predictability in public iaas clouds. *ACM Transactions on Internet Technology (TOIT)* 16, 3 (2016), 1–23.

[14] Wes Lloyd, Shrideep Pallickara, Olaf David, Mazdak Arabi, and Ken Rojas. 2017. Mitigating resource contention and heterogeneity in public clouds for scientific modeling services. In *2017 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 159–166. https://doi.org/10.1109/IC2E.2017.29

[15] Wes Lloyd, Shruti Ramesh, Swetha Chinthalapati, Lan Ly, and Shrideep Pallickara. 2018. Serverless computing: An investigation of factors influencing microservice performance. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 159–169. https://doi.org/10.1109/IC2E.2018.00039

[16] David Perez, Ling-Hong Hung, Sonia Xu, Ka Yee Yeung, and Wes Lloyd. 2020. Characterizing Performance Variation of Genomic Data Analysis Workflows on the Public Cloud. In *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*. IEEE, IEEE, 680–683.

[17] David Perez, Ling-Hong Hung, Sonia Xu, Ka Yee Yeung, and Wes Lloyd. 2020. An Investigation on Public Cloud Performance Variation for an RNA Sequencing Workflow. In *Proceedings of the 11th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics* (Virtual Event, USA) *(BCB '20)*. Association for Computing Machinery, New York, NY, USA, Article 96, 7 pages. https://doi.org/10.1145/3388440.3414859

[18] Danilo Poccia. 2020. New for AWS Lambda – Container Image Support. https://aws.amazon.com/blogs/aws/new-for-aws-lambda-container-image-support

[19] Radim Rehurek and Petr Sojka. 2011. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic* 3, 2 (2011).

[20] Jörg Schad, Jens Dittrich, and Jorge-Arnulfo Quiané-Ruiz. 2010. Runtime measurements in the cloud: observing, analyzing, and reducing variance. *Proc. VLDB Endow.* 3 (2010), 460–471.

[21] Robert Schmitz, III and Danielle Lambion. 2022. *FaaS performance variability study using topic-modeling*. https://github.com/lambiond/faas_variability_topic_modeling

[22] Alexandru Uta and Harry Obaseki. 2018. A Performance Study of Big Data Workloads in Cloud Datacenters with Network Variability. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering* (Berlin, Germany) *(ICPE '18)*. Association for Computing Machinery, New York, NY, USA, 113–118. https://doi.org/10.1145/3185768.3186299

[23] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking Behind the Curtains of Serverless Platforms. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 133–146. https://www.usenix.org/conference/atc18/presentation/wang-liang