Characterizing X86 and ARM Serverless Performance Variation:

A Natural Language Processing Case Study

Danielle Lambion, Robert Schmitz,
Robert Cordingly, Navid Heydari, Wes Lloyd
dlambion@uw.edu, rgs1@uw.edu, rcording@uw.edu,
navidh2@uw.edu, wlloyd@uw.edu

April 9, 2022

School of Engineering and Technology
University of Washington, Tacoma

5th Workshop on Hot Topics in Cloud Computing Performance (HotCloudPerf 2022)

Outline

- Background
- Research Questions
- Natural Language Processing Use Case
- Experimental Approach
- Experimental Results
- Conclusions

PROBLEM: Serverless Computing Performance Variation

- Cloud providers abstract serverless platform configuration and management details from end users
- Abstraction leads to reduced observability necessary for root-cause performance analysis
  - AWS Lambda abstracts processor type and multi-tenancy, factors that lead to performance variation

Characterizing X86 and ARM Serverless Performance Variation

- Investigate support for x86_64 and ARM64 CPU architectures on AWS Lambda
- Harnessed container images to package and deploy 3-step Natural Language Process (NLP) pipeline
- Investigate performance variation for a 24-hour period across four cloud regions and two CPU architectures

CPU Steal

- The number of tenants sharing cloud servers on serverless FaaS platforms is obscured
- The cpuSteal metric, available from Linux procfs has been shown to indicate resource contention (Intel)
- There is no cpuSteal on ARM so there is no oversubscription on ARM, every core is a physical core

Related Work

- Schad et al. (2010) evaluated performance variation of Infrastructure-as-a-Service (IaaS) cloud and object storage platforms on Amazon EC2
- Leitner and Cito (2016) evaluated performance variation on VMs (IaaS) deployed on Amazon, Google, Azure, and IBM for 72-hour periods
- Uta and Obaseki (2018) emulated network bandwidth variability results on (IaaS) from Ballani et al. (2011) to investigate performance variation implications for big data workloads
- Wang et al. (2018) observed multiple CPU types and VM configurations on serverless Function-as-a-Service (FaaS) platforms to account for performance variation from CPU heterogeneity to produce a performance model
- Other related work identified the number of function “tenants” increases when scaling up concurrent requests on AWS Lambda (FaaS)

Related Work - 2

- Samuel Ginzburg and Michael J Freedman. 2020. **Serverless Isn't Server-Less: Measuring and Exploiting Resource Variability on Cloud FaaS Platforms.**
- Investigated diurnal patterns on AWS Lambda serverless FaaS platform
  - Studied on intra-region performance on us-east-1 over one week
  - Investigated multi-region performance variation on us-east-1 and ap-northeast-2 over 48 hours
  - Identified a relationship between function performance and cpuSteal metric
  - Demonstrated better performing instances could be found and exploited with short-running performance tests for cost savings (2-8% on average)

Outline

- Background
- Research Questions
- Natural Language Processing Use Case
- Experimental Approach
- Experimental Results
- Conclusions

Research Questions

<u>RQ-1:</u> What are the performance and cost implications of adopting ARM64 vs. x86_64 CPU architecture on a commercial serverless FaaS platform?

<u>RQ-2:</u> What performance variation results from use of alternative cloud regions where the state of resource contention is likely to change on a commercial serverless FaaS platform?

Outline

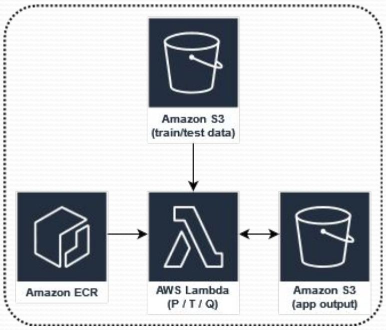
- Background
- Research Questions
- Natural Language Processing Use Case
- Experimental Approach
- Experimental Results
- Conclusions

Natural Language Processing Use Case

- Performed Topic Modeling
- Find groups of words (topics) from a set of text data that represent textual information
  - A training set of news headlines is used to create a topic model, which generates a set number of topics
- New headlines can query the topic model to select the best fitting topic representation

Natural Language Processing Use Case - 2

- **Preprocessing (P) function:** load and prepare news headline data for model training
  (avg. runtime ARM64: 2.64 min, x86_64: 2.82 min)
- **Training (T) function:** train a latent Dirichlet allocation (LDA) topic model using the output from the data preprocessing function
  (avg. runtime ARM64: 3.75 min, x86_64: 3.02 min)
- **Query (Q) function:** query the model for topics with new headlines
  (avg. runtime ARM64: 5.86 min, x86_64: 6.63 min)
- Total Pipeline Runtime:
  ARM64: 12.25 minutes
  x86_64: 12.47 minutes



Outline

- Background
- Research Questions
- Natural Language Processing Use Case
- Experimental Approach
- Experimental Results
- Conclusions

Experimental Approach

- Tested across 4 regions: Tokyo (ap-northeast-1), Frankfurt (eu-central-1), Ohio (us-east-2), and Oregon (us-west-2)
- Lambda functions were deployed via container images
- Deployed an (ARM64) c6gd.large EC2 instance in each region as client to invoke Lambda functions
- Invoked each Lambda function in the NLP pipeline sequentially using the AWS CLI with a 2-second delay between each function call
- The Serverless Application Analytics Framework (SAAF) was used to obtain profiling data for each function call
- 112 pipeline executions were performed in each region/architecture for a 24-hour period

Outline

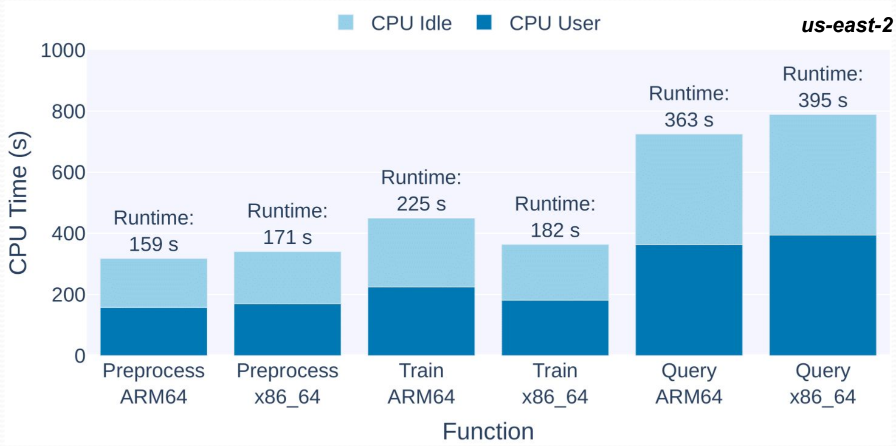
- Background
- Research Questions
- Natural Language Processing Use Case
- Experimental Approach
- Experimental Results
- Conclusions

RQ-1: Architecture Performance

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

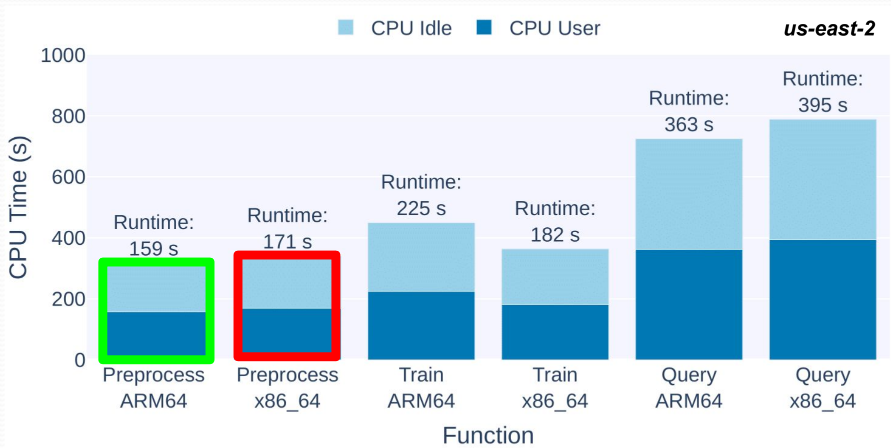
RQ-1: Resource Utilization

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



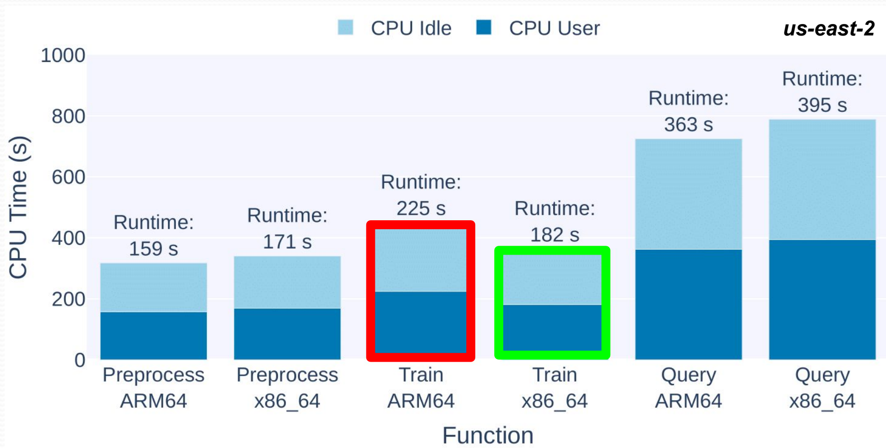
RQ-1: Resource Utilization

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



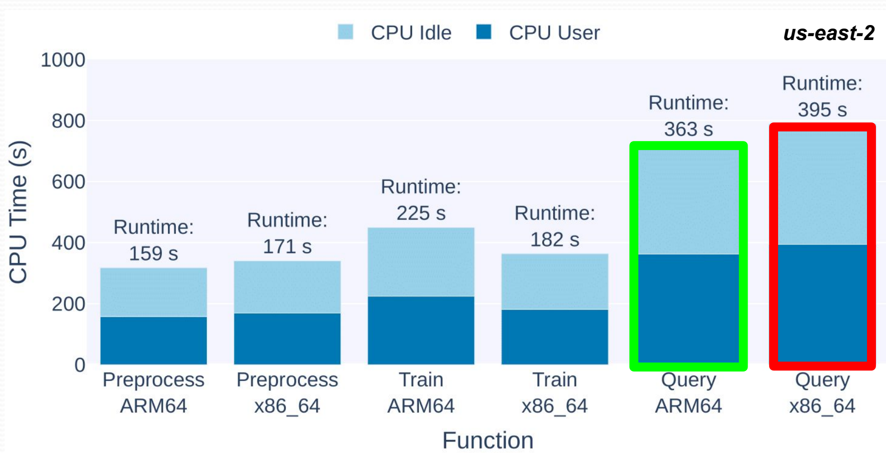
RQ-1: Resource Utilization

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



RQ-1: Resource Utilization

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



RQ-1: Runtime and Cost Comparison

CPU architecture runtime comparison

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

across all regions

RQ-1: Runtime and Cost Comparison

CPU architecture runtime comparison

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

across all regions

RQ-1: Runtime and Cost Comparison

CPU architecture runtime comparison

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

across all regions

RQ-1: Runtime and Cost Comparison

CPU architecture runtime comparison

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

across all regions

RQ-1: Runtime and Cost Comparison

CPU architecture runtime comparison

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

across all regions

RQ-1: Runtime and Cost Comparison

<u>RQ-1 (summary)</u>

ARM64 (~1.7%) faster NLP pipeline runtime

x86_64 (~27%) more expensive (10k runs)

x86_64 (2.8x) more performance variation

RQ-2: Performance Variation

What performance variation results from the use of alternative cloud regions over 24-hours where the state of resource contention is likely to change to host a multi-step NLP pipeline on a commercial serverless FaaS platform?

RQ-2: Performance Over 24 hours

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

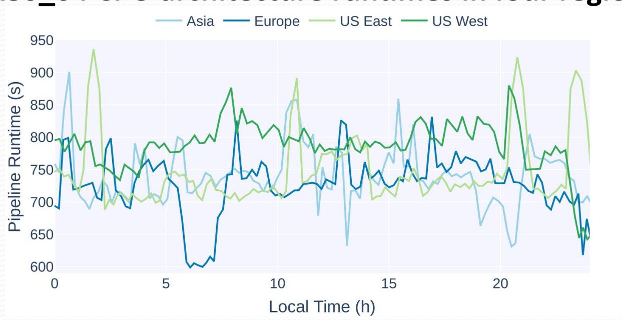
RQ-2: Performance Over 24 hours

CPU steal across AWS regions for x86_64

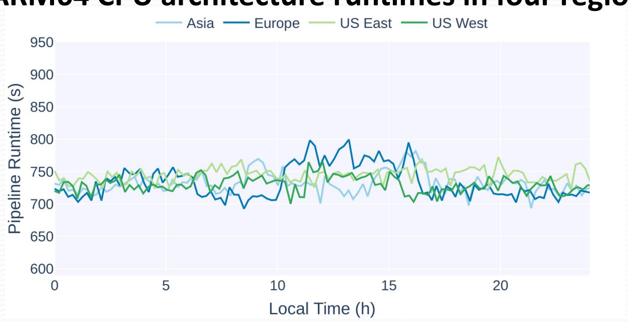
metric/region	us-east-2	us-west-2	eu-central-1	ap-northeast-1
avg cpuSteal/min	8.89	18.26	4.24	4.79
% of eu-central-1	209.7	431.6	100.0	113.0
R^2 runtime	0.618	0.379	0.427	0.39
Pearson (r)	0.7861	0.6157	0.6537	0.6249

RQ-2: Performance Over 24 hours

x86_64 CPU architecture runtimes in four regions

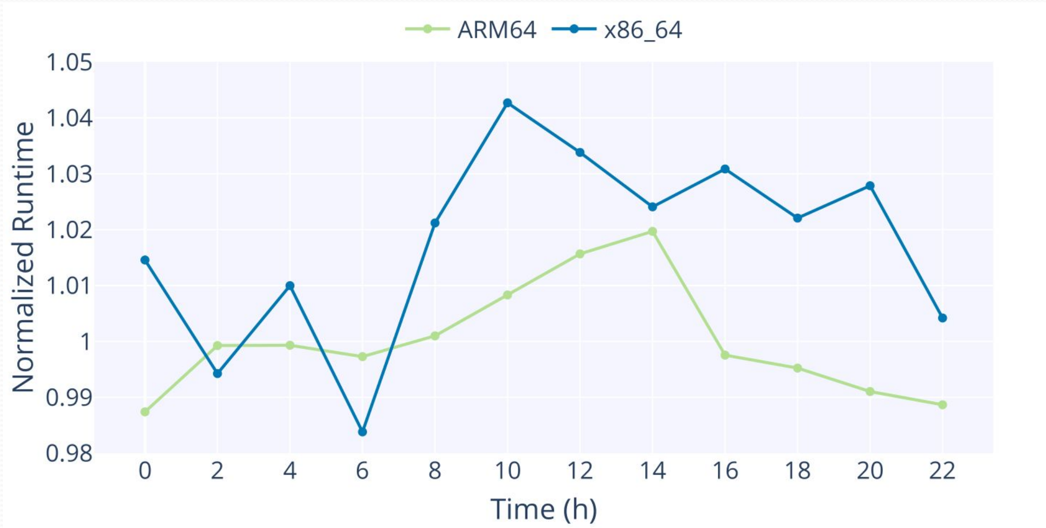


ARM64 CPU architecture runtimes in four regions



RQ-2: Performance Over 24 hours

Global two-hour average NLP pipeline runtime normalized against global ARM64 average runtime



RQ-2: Performance Over 24 hours

Global two-hour average NLP pipeline runtime

RQ-2 (summary)

Performance on both architectures degraded during business hours across 4 time zones

During the night (midnight to 6am) performance improved

Normalized Runtime

Time (h)

Outline

- Background
- Research Questions
- Natural Language Processing Use Case
- Experimental Approach
- Experimental Results
- Conclusions

Conclusions

(RQ-1):

- ARM64 architecture on AWS Lambda featured both discounted cost and lower resource contention versus x86_64
- **up to ~33.4% cost savings** for our NLP pipeline - ARM64 in us-west-2
- ARM64 cost savings, however, may only be temporary

(RQ-2):

- Potential to improve non-latency sensitive workload performance by leveraging regions outside regular business hours
- **6% global average runtime differential** across four regions from 6:00am-8:00am vs. 10:00pm-12:00pm

THANK YOU FOR WATCHING

Questions or Comments?

Please Email:

dlambion@uw.edu, rgs1@uw.edu,
rcording@uw.edu, or wlloyd@uw.edu