Sky Computing for Serverless: Infrastructure Assessment to Support Performance Enhancement

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

Xinghan Chen
University of Washington
Tacoma, Washington, United States
kirito20@uw.edu

Ling-Hong Hung
University of Washington
Tacoma, Washington, United States
lhhung@uw.edu

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

Abstract

Serverless Function-as-a-Service (FaaS) cloud computing platforms enable developers to deploy applications that automatically scale and manage computing infrastructure. The "serverless" nature of these platforms leads to infrastructure obfuscation, unpredictable performance, and inconsistent costs. This unpredictability stems from a diverse, constantly shifting pool of Infrastructure-as-a-Service (IaaS) hardware used to host functions, including varying CPU generations, clock speeds, memory types, and server capacities. As a result, serverless function instances run on heterogeneous hardware configurations, leading to variability in performance and inconsistent costs.

To address this challenge and to leverage hardware heterogeneity as an advantage for serverless sky computing, this research develops a novel methodology to infer the distribution of key hardware characteristics (e.g., CPU model, CPU clock speed, number of allocated hosts) in distinct serverless availability zones (AZ). Using these observations, our serverless Sky Computing system can aggregate resources from many AZs to route requests to the highest-performance infrastructure that is available. Using our approach we demonstrate up to 18.2% in cost savings by capitalizing on hidden hardware heterogeneity in the cloud.

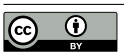
CCS Concepts

• **Computer systems organization → Cloud computing**.

Keywords

Serverless computing, FaaS, Cloud computing, Infrastructure characterization, Progressive sampling

ACM Reference Format:
Robert Cordingly, Xinghan Chen, Ling-Hong Hung, and Wes Lloyd. 2025. Sky Computing for Serverless: Infrastructure Assessment to Support Performance Enhancement. In *2025 IEEE/ACM 18th International Conference on Utility and Cloud Computing (UCC '25), December 1–4, 2025, France, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3773274.3774266



UCC '25, France, France

ACM ISBN 979-8-4007-2285-1/2025/12
https://doi.org/10.1145/3773274.3774266

1 Introduction

Serverless Function-as-a-Service (FaaS) cloud platforms enable developers to deploy code without managing the underlying servers. However, this 'serverless' abstraction comes at the cost of infrastructure obfuscation and unpredictable performance. In practice, FaaS providers utilize a constantly shifting pool of heterogeneous hardware to host ephemeral function instances (FIs), leading to inconsistent execution performance and unpredictable costs for the same workload. For example, a function may run up to 50% slower (and thus cost more with time-based billing) on one server versus another because of hidden differences in CPU type or load contention.

Sky computing has emerged as a vision to harness cloud diversity as an advantage rather than a drawback. Sky computing refers to federating resources from multiple clouds via a unified middleware layer [8, 29]. Serverless sky computing aims to pool FaaS deployments from multiple availability zones (AZ), regions, and clouds to then intelligently route function invocations to wherever the best performance can be obtained at that moment.

To realize this vision, we introduce a novel system and methodology for serverless sky computing performance optimization. At a high level, our approach first profiles the available infrastructure of FaaS deployments to each zone, region, or provider to infer hardware characteristics such as the available CPU models and clock speed distributions and their impact on function runtime. Using a tree-based recursive sampling strategy, our infrastructure analysis method efficiently uncovers a spectrum of back-end server types present in a given AZ. We then characterize a zone's CPU type distribution based on thousands of polling samples. Finally, our sky computing runtime uses CPU characterizations to make intelligent routing decisions. For each incoming function request, it can decide whether to retry execution for a faster server, route the request to a different region or zone with superior hardware, or employ a hybrid of these strategies.

1.1 Research Questions

In evaluating our approach, we address the following research questions (RQs):

RQ-1 Infrastructure Variation: What CPU variation can be observed in the server infrastructure used to host serverless FaaS

platforms across different zones and regions? How does the infrastructure pool change over time?

Serverless FaaS platforms are affected by CPU heterogeneity, since public cloud providers use dynamic sets of provisioned servers backed by various CPU types to run serverless functions [9, 11, 18]. For RQ-1, we characterize CPU variation across serverless AZs, regions, and cloud providers and investigate how CPU allocations change over time.

RQ-2 Infrastructure Characterization: Using progressive sampling, how many samples are required to accurately infer the server infrastructure used to host a distinct serverless FaaS platform deployment? How can we balance the cost versus accuracy of sampling to infer server characteristics?

We consider that each serverless FaaS platform has a distinct deployment for each cloud AZ (e.g., us-east-2a, us-east-2b, etc.). Each deployment consists of a temporarily provisioned set of machine instances on which the platform is hosted. For AWS Lambda, bare metal EC2 instances are likely used to host many microVMs that host individual FIs [6].

RQ-3 Performance Optimization: By exploiting serverless CPU variation and sky computing resource aggregation, to what extent are runtime and cost improvements possible for diverse workloads?

We can route requests to execute workloads across FaaS platform deployments in different AZs, regions, and even clouds. By characterizing FaaS hardware deployments and directing workloads to the best CPUs using intelligent routing, in this paper, we report on the possible runtime and cost benefits for a variety of serverless workloads.

1.2 Paper Contributions

This paper makes the following research contributions:

(1) We introduce and expand on tools to enable serverless sky computing, such as dynamic functions and the sky mesh, as discussed in Sections 3 and [13].
(2) We introduce a powerful infrastructure sampling technique capable of observing available infrastructure in an AZ.
(3) We created CPU characterizations for 41 regions on AWS Lambda, IBM Code Engine, and Digital Ocean Functions.
(4) We thoroughly investigated ways to minimize sampling costs and monitored temporal variation in CPU characterizations to determine a characterization's usable lifespan.
(5) Using these CPU characterizations, we investigated three methods of performance enhancement: (1) a regional approach that routes requests to zones with fast CPUs, (2) retry approaches that rerun functions on poor performing infrastructure, and (3) a hybrid approach combining both.

2 Background and Related Work

The challenges of performance variation due to heterogeneous hardware on serverless platforms have been addressed in previous literature. Our study builds on previous work and applies sky computing concepts to FaaS platforms to aggregate resources and achieve performance improvements.

2.1 Hardware Heterogeneity

Hardware heterogeneity has been investigated on public clouds for over a decade. Multiple publications observed that identical instance types on Amazon EC2 and other cloud providers (e.g. Rackspace Cloud) often had varying CPU configurations that backed infrastructure-as-a-service (IaaS) instances, leading to notable performance variation [15, 20, 24, 30]. Leveraging this hardware heterogeneity, Ou et al. offered a trial-and-better approach, where the virtual machine (VM) CPU type was checked at boot time, and the VM was relaunched if the best CPUs were not found, providing a performance and cost savings of 15 to 30%. Our study builds on these methods in the context of FaaS platforms.

FaaS platforms provide unique challenges as they have a constantly changing pool of available infrastructure that are used to host execution environments for serverless functions known as function instances (FIs). FIs are constantly being provisioned and deprovisioned, and batches of requests will span many FIs. Unlike IaaS VMs that can be initialized and held for days or months, FaaS FIs are often only used for a few minutes before they are destroyed.

2.2 Sky Computing

Sky computing refers to an emerging paradigm in which resources from multiple cloud providers are pooled and exposed through a unifying interface [8, 29]. The goal is to aggregate resources from multiple clouds through a common interface to then provide performance enhancement, reduced costs, higher availability, etc. Chasins et al. and Stoica et al. suggest that the barriers to achieving sky computing are more economic than technical, such as data egress fees and lack of cross-provider billing agreements. Sky computing aims to give users the freedom to utilize the best combination of cloud resources at any given time without being locked into a single vendor. Yunhao Mao discussed creating SkyBridge, a proof of concept data management system enabling multi-cloud data storage [22]. Yang et al. created SkyPilot, an intercloud broker for large language model training and machine learning workloads where workloads are dynamically moved across available cloud providers to reduce cost by up to 80% and increase availability [31].

2.3 Performance Variability in Serverless Platforms

Research has been conducted to observe the infrastructure available on FaaS platforms and to measure how their performance varies over time [9, 14, 17]. Kelly et al. conducted an hour-by-hour analysis of the major FaaS platforms over a month, finding that fluctuations in background load can impact function latency and throughput over the course of a day [18]. Schirmer et al. specifically examined Google Cloud Functions and observed a pronounced diurnal pattern, dubbing it 'The Night Shift' [27]. During peak daytime hours when cloud infrastructure is heavily utilized by many users, serverless functions experienced slower execution and higher tail latencies. Alongside that, Schirmer et al. developed a system called Minos that can benchmark individual FIs to observe performance variability and focus workloads to run on instances with better performance and potentially lower resource contention [26]. Lambion et al. analyzed the performance variability of running a compute-bound Natural Language Processing pipeline on x86_64 and ARM64

CPUs on AWS Lambda on multiple cloud regions over a 24-hour day [19].

2.4 Serverless Aggregation

Aggregating serverless resources across multiple environments (regions, clouds, or edge sites) has been explored as a way to improve performance, reliability, and even sustainability of FaaS applications. Several recent systems address aspects of this distributed serverless orchestration. Smith et al. created FaDO (FaaS Functions and Data Orchestrator), a tool designed to allow data-aware functions scheduling across multiple serverless compute clusters present at different locations (e.g. edge and cloud) [28]. FaDO further provides users with an abstraction of the serverless compute cluster's storage, allowing users to interact with data across different storage services through a unified interface.

Baarzi et al. discuss the merits of what they term Virtual Serverless Providers (VSPs), which aggregate FaaS offerings from multiple clouds under one virtual provider [7]. Their architecture introduces a broker layer between the user and the cloud functions: function invocations are routed through this broker, which decides which cloud's FaaS service to use for each invocation. By doing so, they achieved up to 4.2x higher throughput, 98.8% fewer SLO latency violations, and 54% cost reduction in their experiments, compared to sticking with a single provider [7]. Sampé et al. presented Lithops, a toolkit for transparent multicloud computing using serverless backends [25]. Lithops wraps the Python multiprocessing API, allowing an ordinary Python program to execute in parallel across many cloud function invocations on different providers without code changes. Jindal et al. took a step toward federated FaaS scheduling with a system called Courier [16]. Courier can route function requests among a set of deployments, in their case, an on-premise OpenWhisk installation and public cloud FaaS (AWS and Google). They showed that even simple policies like round-robin distribution across providers can improve overall throughput and latency in a heterogeneous FaaS deployment compared to traditional single-cluster load balancers.

These efforts demonstrate a growing interest in unifying and optimizing serverless across distributed infrastructure. Our work differs in its emphasis on hardware heterogeneity and improving overall serverless hosting costs. We push the scale to a global level: our experimental sky computing platform spans 41 cloud regions, across multiple cloud providers, with a total of over 1,600 function deployments.

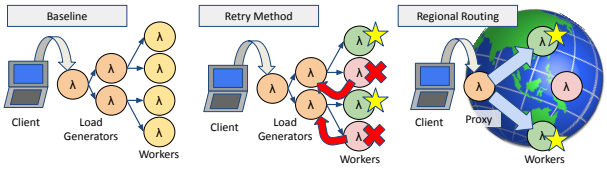


Figure 1: Performance enhancement approaches utilizing CPU characterization data of regions and workloads.

3 Methodology

This section presents the methodology we used to construct and evaluate the compute infrastructure for a global *serverless sky computing* platform.

3.1 FaaS Infrastructure Sampling

Heterogeneous hardware on serverless platforms is known to cause considerable performance variation based on the available infrastructure. Previously, profiling a large sample of a FaaS AZ's infrastructure was impossible due to low concurrency limits set by cloud providers. For example, AWS Lambda had a limit of 1,000 concurrent function requests on the accounts used in this study. We developed a FaaS infrastructure sampling technique that is able to observe significantly more infrastructure without being limited by concurrent request quota.

Whenever functions are invoked, an execution environment is created known as a function instance (FI) where a user's code is executed. FIs then persist for about five minutes after a function request has finished [21]. For thorough infrastructure sampling, we do not actually need to have a high number of concurrent requests, but instead we want to observe a high number of unique FIs.

To maximize the number of FIs sampled, we deployed 100 distinct serverless functions each with the same code. Each deployment had unique memory settings, ranging from 10,140 to 10,240 MB, and unique source code packages with identical logic (for example, each function printed a different 'hello' message, but all other logic remained the same). To observe infrastructure, we then made 1000 parallel requests to one of the functions (referred to as a poll). Requests were made through a branching tree of recursive function invocations to maximize parallelism as shown in Figure 1. The functions executed a sleep function to pause for a short duration to ensure that all 1000 parallel requests generated by our clients executed concurrently on unique FIs. All of these deployment decisions (e.g., memory setting, code variation) were made to maximize the number of unique FIs and the number of host servers used. By executing a poll on one function endpoint, waiting for requests to return, and then executing another poll on a different function endpoint, we can observe 2000 unique function instances without being limited by the maximum concurrent request quota. By leveraging all 100 function endpoints, we can observe the hosting infrastructure (e.g., underlying CPU, etc.) for 100,000 FIs. Using the Serverless Application Analytics Framework (SAAF) [5] we can build hardware characterizations of the ratio of each CPU type for a given region as shown in Figure 2.

3.2 Dynamic Functions

To facilitate the development and prototyping of a serverless sky platform, we developed adaptable serverless functions called "dynamic functions". Dynamic functions provide a generic execution environment that is preemptively deployed to every serverless FaaS AZ, region, and cloud, which can then be used to execute serverless workloads on demand. With dynamic functions, the function source code is provided as an input parameter in the function payload or via an object storage service (e.g., Amazon S3 or Google Cloud Storage) or AWS Lambda Layers [1–3].

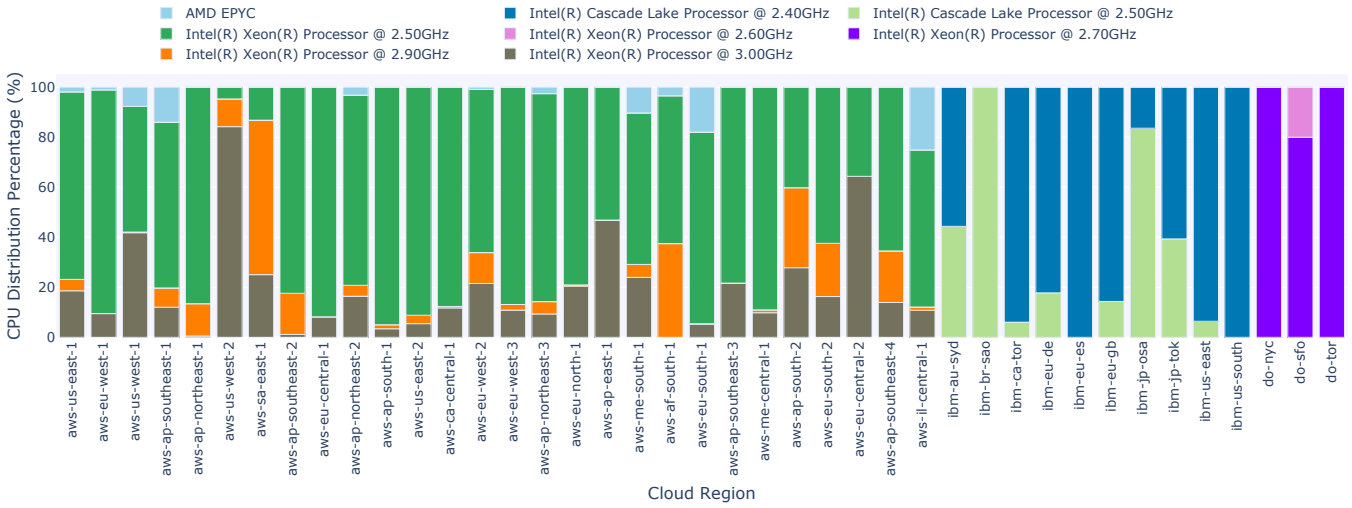


Figure 2: CPU distributions observed on AWS Lambda, IBM Code Engine, and Digital Ocean functions. (Section 4.2)

By removing the workload source code from the deployment package, dynamic functions can be deployed once and then repurposed to run any desired workload without redeployment. Dynamic functions are implemented using Python, and since the source code is interpreted in Python, running the code supplied through the function payload results in very little additional overhead compared to running a native Python function [4]. For all dynamic function executions in this study, decoding and executing the source code provided in the payload added less than 1 millisecond to each function invocation. We did not use external storage services for these functions.

To help improve performance and functionality, dynamic functions can also include dependencies, files, and other arbitrary data in the request payload. The FaaS Experiment Toolkit (FaaSET) [5, 13] includes tools that can take files, compress, encode, and automatically generate the payload for a request to a dynamic function. When the dynamic function receives the request, it will decode, decompress, and store the payload data on the ephemeral file system of the serverless function. Dynamic functions then record hashes of the payload data to determine if the decode and decompress process should repeat for future requests. If a second request is made to the same FI with the same payload data, then the dynamic function takes advantage of the cached data. Even for a maximum payload input size of 5 MB, the decode time remains minimal (at most 70 ms).

3.3 Sky Mesh

To prototype a serverless sky computing platform and accelerate development and testing, we created a sky mesh. The sky mesh consists of a large deployment of dynamic functions to every region on AWS Lambda, IBM Code Engine, and Digital Ocean functions. On these platforms, functions are deployed with a variety of memory settings and configurations. On AWS Lambda, dynamic functions are deployed with 128 MB, 256 MB, 512 MB, 1 GB, 2 GB, 4 GB, 6 GB, 8 GB and 10 GB memory settings to both x86_64 and ARM64 CPU architectures. The sky mesh consists of more than 1,600 deployments on AWS Lambda alone. On platforms such as IBM Code Engine, there are substantially fewer deployment options and regions. IBM Code Engine offers only three memory settings, 1 GB, 2 GB, and 4 GB. Due to the much smaller scope of the platform, we are able to deploy dynamic functions and cover the entire function configuration space on IBM Code Engine with only 30 function deployments. The goal of the sky mesh and dynamic functions is to offer generic pre-deployed functions which can be readily harnessed to deploy any serverless workload anywhere on demand without the need to deploy new functions or reconfigure anything.

3.4 Serverless Smart Routing System

Prior work has introduced a serverless smart request routing system built on an early prototype sky mesh that dynamically routed function invocations to cloud regions with the lowest real-time carbon intensity, while bounding network latency with a client–region distance heuristic [12]. That study showed that aggressive, carbon-aware request placement across multiple providers can cut both latency and CO_2e without requiring any modifications to user code. This adaptive routing layer is a prerequisite for any practical serverless sky computing middleware and provides the foundation on which to achieve higher-level performance goals.

This study builds on that routing system in two ways. First, we extend its decision space from purely climate data and latency metrics to hardware performance objectives, exploiting the fact that serverless AZs expose continuously evolving, heterogeneous CPU pools: newly introduced data centers may offer faster infrastructure, while user demand, and thus resource contention, fluctuates temporally and geographically. Second, we combine the routing system with our infrastructure sampling method that profiles each zone's infrastructure pool and learns when rerouting to a more distant, but faster, region can achieve performance gains and reduce overall hosting cost. Our goal is to build a performance-aware system that turns global heterogeneity from a liability into an optimization opportunity.

3.5 Experimental Design

We designed five experiments to address our research questions. Experiments 1 (EX-1) and EX-2 characterize infrastructure across individual zones and across the globe (RQ-1). EX-3 and EX-4 evaluate progressive sampling and temporal stability to balance cost and accuracy in profiling (RQ-2). Finally, EX-5 applies these characterizations to performance-aware routing strategies (RQ-3). These experiments provide a comprehensive foundation for assessing the benefits of infrastructure characterization for serverless sky computing.

EX-1: Serverless Infrastructure Characterization

Our first goal was to sample as many FIs as possible using our infrastructure sampling method described in Section 3.1. We monitor and record the CPUs assigned to each FI with the SAAF profiling tool [5]. Our approach appears to be able to saturate the available infrastructure within the targeted AZ. After approximately 20 to 30 polling cycles (20,000 and 30,000 allocated FIs), we observed that the platform started to return errors for more than 90% of our requests.

To validate that our method saturated the AZ, we performed a parallel experiment using a separate AWS account. We replicated the exact sampling method and after observing errors from the initial AWS account (i.e. after 20-30 polling cycles), we initiated the second account's experiment and immediately encountered saturation errors. More than 90% of the requests from the second account failed after the first iteration of the sampling method. Both AWS accounts were completely independent and isolated from each other. We used different accounts, email addresses, and payment methods. The host machines initiating function requests were separate, they were located on different networks, the deployed functions were unique to each account, and we tested using different invocation methods such as function URLs and API Gateway. The only commonality was the target function deployments were on the same AZ on both accounts. These findings strongly indicate that our sampling mechanism effectively observed the total provisioned infrastructure pool for the FaaS platform within the AZ.

We used these results as the ground truth for the available infrastructure in an AZ. We can now incorporate this knowledge into our smart routing system, to facilitate the routing of requests to zones characterized as having higher performance CPUs with the goal of enhancing performance and reducing costs.

EX-2: Global Infrastructure Characterization

Building on the findings of EX-1, we expanded our scope to encompass all regions on AWS Lambda, IBM Code Engine, and Digital Ocean functions, totaling 41 regions. To provide a comprehensive overview of the available infrastructure in each region, we utilized our sampling technique described in Section 3.1 to build a global hardware characterization.

EX-3: Progressive Sampling Evaluation

This experiment focuses on evaluating the minimum cost required to characterize the provisioned hardware for a serverless FaaS platform's deployment to a distinct AZ using progressive sampling. Building on the 100 function deployment methodology employed in EX-1, we deployed the same set of functions to eleven AZs on AWS Lambda: ca-central-1a, eu-north-1a, ap-northeast-1a, sa-east-1a, eu-central-1a, ap-southeast-2a, us-west-1a, us-west-1b, us-east-2a, us-east-2b and us-east-2c. These AZs covered a diverse range of CPU configurations.

We conducted polls in all of these regions until failure using our sampling method. By incrementally increasing the number of polls as a form of progressive sampling, we can compare our CPU characterization using progressively more data compared to the final characterization when requests began to fail. This experiment enables us to examine the relationship between the number of samples and the accuracy of our infrastructure characterizations. This enables us to investigate the accuracy versus cost trade-off of characterizing the provisioned infrastructure of FaaS platforms.

EX-4: Characterization of Temporal Variation

To investigate the varying nature of the provisioned serverless FaaS infrastructure, we expanded on EX-3 by repeating our progressive sampling test in five AZs (us-west-1a, us-west-1b, sa-east-1a, eu-north-1a and ca-central-1a) daily for two weeks. This experiment had multiple objectives: (1) assessing the applicability of a CPU characterization from a single day to others and (2) determining the sufficient number of samples required for accurate zone characterization.

In contrast to the previous experiment, observations were conducted every 22 hours instead of precisely 24 hours apart, thereby slightly shifting the poll time over the two-week period. This approach enabled the observation of the CPU distribution at varying times throughout the day within each zone, minimizing extensive sampling.

Beyond our observations of heterogeneous CPUs, the number of samples necessary to reach failure also exhibited temporal variation. This variation likely reflects the allocation of IaaS machine instances to host the serverless FaaS AZ at any given moment and could potentially influence performance as the platform scales to accommodate the current demand of FaaS users.

Finally, with the CPU characterization's established, we evaluated the performance implications of making smart routing decisions based off our knowledge of the available hardware. To create realistic workloads, we employed twelve serverless functions defined in Table 1. The source code for these functions was programmed by us or adopted from the Serverless Benchmark Suite (SeBS) [10]. All of the functions were modified to be packaged as dynamic functions, redesigned to remove any dependencies on external services, and updated to include additional CPU-based decision-making logic, which will be discussed later.

EX-5: Performance Optimization

To provide a baseline and collect the necessary profiling data, we ran each of our workloads 10,000 times on each of the AZs used in EX-4. This initial profiling step not only provided a baseline average runtime for how the workloads would perform without any smart routing, but it also provided samples of how the workloads perform on each CPU.

With the baseline established, we created three routing methods that our smart routing system could utilize:

Regional Routing: This approach utilizes the smart routing system to direct requests to AZs that have more high performance CPUs to run workloads. Due to the difference in CPU distributions between zones in different regions, as shown in Figure 2, this approach has the potential for substantial performance and cost benefits.

Table 1: Function Names and Descriptions

Function	vCPUs	Description
Graph MST	1	Generates a graph and calculates its minimum spanning tree.
Graph BFS	1	Generates a graph and performs a breadth-first search.
Page Rank	1.2	Generates a graph and computes the PageRank of each node.
Disk Writer	1	Generates text, repeatedly writes it to disk, and deletes it.
Disk Write and Process	1	Writes a large text file and then runs several shell commands (`wc`, `base64`, `sha1sum`, `cat`) on it in a loop.
Zipper	2	Generates files and compresses them into ZIP archives.
Thumbnailer	1	Generates a random bitmap image and scales it to different sizes.
Sha1 Hash	1	Takes an input string and produces its SHA-1 hash.
JSON Flattener	1	Recursively generates a large JSON object and flattens it into key–value pairs.
Math Service	2	Builds large arrays and repeatedly performs arithmetic operations on them.
Matrix Multiply	2	Generates large matrices and executes multiply and dot operations in loops.
Logistic Regression	2	Runs logistic-regression SGD across two threads on a generated dataset for the requested epochs.

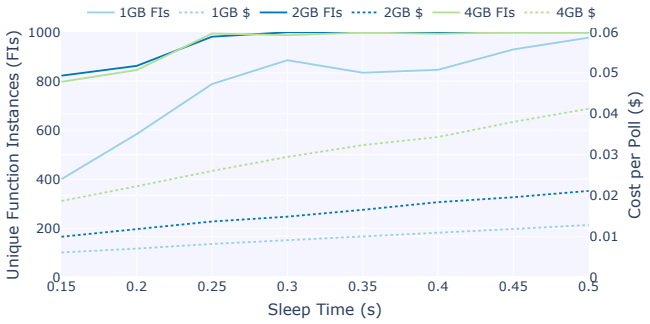


Figure 3: Infrastructure sampling technique cost comparison to number of observed unique FIs (FIs).

This regional approach has trade-offs. Routing requests to AZs located further away will introduce additional network latency versus routing to nearby zones. However, network latency to FIs is not included in the billable runtime of the FaaS platform. Although latency and overall round trip time for a request are likely to increase, faster runtime will reduce overall costs.

Retry Method: When a function is invoked using the retry approach, it first checks the CPU of the FI. The function then makes the decision to run the workload or immediately return. This is applicable to regions that show notable performance variation due to hardware heterogeneity and is inspired by the IaaS methodologies in [15, 24].

For batch-based workloads, the retry approach can be particularly advantageous. FIs with poorly performing CPUs can be held for a brief moment (e.g. 150 ms), and then instead of returning immediately a new request can be issued to guarantee requests do not route to poor performing FIs.

The retry approach can also be tuned by specifying the CPUs that are banned. If all but one CPU are banned, then all function requests will be routed to the ideal CPU. This has a trade-off though, if the AZ only has a small number of the ideal CPUs then the overhead of additional retries grows rapidly. If the retry approach is too selective and too many CPUs are banned, then the overhead of these retries will consume any performance benefits. The overhead can be mitigated by only banning very poorly performing CPUs, and ultimately the effectiveness will depend on the specific CPU distributions in the AZ.

Hybrid Approach: The hybrid approach combines the retry and regional approaches to further optimize performance. By initially selecting a region with favorable CPU distributions, the system can substantially reduce the reliance on retries. Subsequently, the retry logic within this region acts as an additional fine-tuning mechanism, targeting specific optimal CPUs to handle workloads more efficiently.

This combined approach allows for enhanced flexibility, leveraging regional differences in CPU availability to reduce overhead from excessive retries while still benefiting from precise CPU targeting within regions. By strategically limiting retries to regions where hardware heterogeneity is observed, the hybrid approach can substantially reduce overall execution costs and improve performance, effectively balancing the trade-offs inherent in each individual method.

4 Experimental Results

The following sections present the results of experiments defined in Section 3.5.

4.1 EX-1: Infrastructure Observation Verification

In experiment 1, we evaluated the effectiveness of our polling technique at observing a maximum amount of serverless infrastructure per AZ, as described in Section 3.5. A key characteristic of AWS Lambda's design that we leverage for our approach is that each new FI created is guaranteed to stay active for a minimum of five minutes [21]. By saturating the available server infrastructure in the AZ in less than five minutes, we are able to profile the highest number of ephemeral FIs used to host the FaaS platform.

To achieve the desired parallelism to enable resource profiling, we invoked a sleep function to sustain requests on specific FIs and to force requests to spawn new FIs. Figure 3 illustrates the impact of varying the sleep interval to achieve the creation of the desired number of unique FIs and the associated cost. **We observed that a sleep interval of 0.25 seconds provided an optimal balance by maximizing creation of unique FIs in 2GB and 4GB memory allocations while incurring minimal total costs, less than two cents per poll.** Shorter sleep intervals reduced costs but increased the likelihood that client requests are routed to existing FIs. Longer sleep intervals kept more FIs active, forcing the creation of new FIs to serve client requests at the expense of higher runtime costs. As we decrease function memory, the sleep time needed to achieve full coverage increased.

Figure 4 depicts the degradation of the FaaS platform's ability to scale and create new FIs resulting from our successive polls. A clear threshold is visible at approximately 20,000 sequential polls, beyond which scaling degradation occurred in this AZ (us-west-1a). Initially, each poll was successful at creating nearly 900 new FIs, which is the maximum client requests we sent per round. As we saturated all available servers in the AZ with FIs, the failure rates escalated dramatically. At the end of our experiment, leading to 80 to 98% failure of our client requests. This finding suggests that

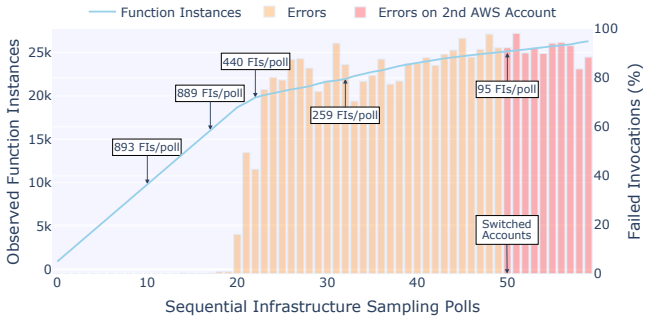


Figure 4: Observed FIs (FIs) and failed requests with an increasing number of sequential polls across two accounts.

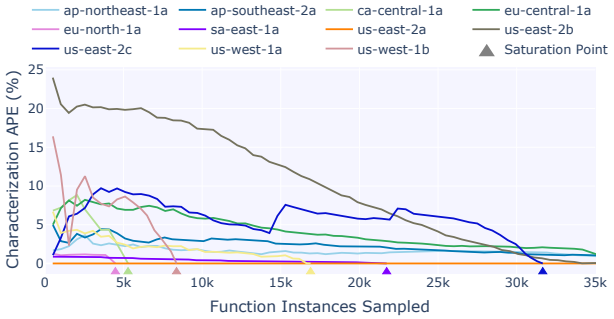


Figure 5: Characterization error relative to the number of FIs on 11 AZs on AWS Lambda.

our polling technique saturates the available serverless infrastructure, while the platform reacts and scales slowly to meet demand. This behavior reflects the dynamic nature of serverless provisioning, where infrastructure availability can degrade under sustained high-intensity access, highlighting the importance of understanding system scaling limits during these characterization efforts. To verify that we were not simply being rate limited, we performed the experiment across two AWS accounts, with different clients, and observed many failed function calls on the secondary account immediately after exhausting the AZ using the first account. To avoid this, we defined the failure point to stop sampling as when more than 50% of the requests in a sampling poll failed.

4.2 EX-2: Global Infrastructure Characterization

We utilized our infrastructure sampling technique to observe infrastructure heterogeneity on a global scale on every region on AWS Lambda, IBM Code Engine, and Digital Ocean functions in EX-2. On AWS Lambda, we identified four distinct CPU types: three Intel Xeon processors with clock speeds of 2.5 GHz, 2.9 GHz, and 3.0 GHz, respectively, and one AMD EPYC processor as shown in Figure 2. The 2.5 GHz processor was the most prevalent, followed by the 3.0 GHz processor, while the AMD EPYC processor was relatively uncommon. On IBM Code Engine, we observed two Intel Cascade Lake processors with clock speeds of 2.4 and 2.5 GHz. In Digital Ocean Functions, we identified Intel Xeon Processors with clock speeds of 2.6 and 2.7 GHz.

Several key observations emerged from our analysis: (1) not all AWS Lambda regions included all four CPU types; (2) the AMD EPYC processor was particularly rare, and was most frequently observed in the il-central-1 region; (3) every region had the 2.5 GHz Xeon processor, and (4) all but af-south-1 hosted the 3.0 GHz processor. Although the 2.5 GHz processor was generally most common, regions like us-west-2 demonstrated notable variation, where the 3.0 GHz processor was most prevalent. This considerable heterogeneity underscores the potential for performance improvements by strategically leveraging regions based on their in situ CPU distributions. Given the lack of observed hardware heterogeneity on IBM Cloud Functions and Digital Ocean functions, the remaining experiments focused solely on AWS Lambda.

4.3 EX-3: Progressive Sampling Evaluation

Experiment 3 focused on determining the minimal cost required to achieve an accurate hardware characterization of AWS Lambda AZs. In EX-1, we found that the cost to fully saturate an AZ is approximately $0.20. If required to sample dozens of AZs, multiple times per day, the profiling cost for sky computing quickly balloons.

To evaluate the minimal costs to characterize an AZ, we incrementally increased the number of polling requests executed across eleven AZs. Figure 5 shows the absolute percentage error (APE) of progressive sampling to determine zone-wide CPU distributions versus ground truth characterization obtained by complete polling until saturation. The figure highlights the different failure points between zones. For example, eu-north-1a fails after approximately 5,000 function calls, indicating that AWS Lambda had a smaller number of provisioned ephemeral servers, whereas eu-central-1a sustained ten times the calls before failing. This variability in failure point further indicates that our sampling method is saturating regions and not hitting a fixed rate limit.

Remarkably, a single poll in most AZs resulted in a low APE of 10% or less with a maximum observed error of 25% in us-east-2b. To achieve 95% characterization accuracy, on average six polls were required across all AZs. One in 11 AZs experienced anomalous spikes in error following sequential polls that revealed previously unseen hardware, likely due to the platform allocating new instances dynamically. Despite such outliers, the general trend indicated a consistent reduction in error rates as the number of samples approached the respective zone's failure threshold, demonstrating an effective trade-off between accuracy and cost in hardware characterization. In contrast, us-east-2a exhibited a unique scenario, consistently returning 0% error as all requests utilized the 2.5 GHz Intel Xeon processor exclusively.

4.4 EX-4: Temporal Infrastructure Variation

To evaluate temporal variability in serverless infrastructure for EX-4, we sampled five AWS AZs daily for a two-week period. We further investigated how temporal factors impact the accuracy and cost of hardware characterization. For each zone, we compared intermediate characterizations at various numbers of polls with the ground truth obtained just before failures. As shown in Figure 6 demonstrates that the number of polls required to achieve a characterization within 5% absolute percent error (APE) of the final

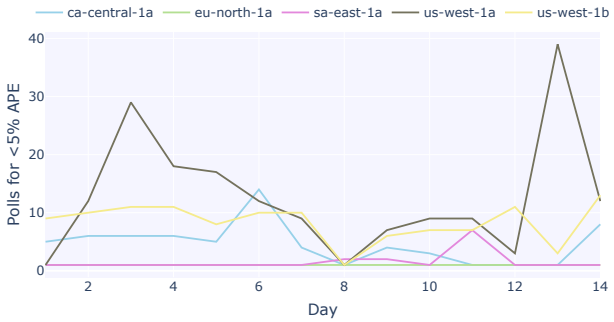


Figure 6: Function instances needed to be sampled to achieve 95% characterization accuracy.

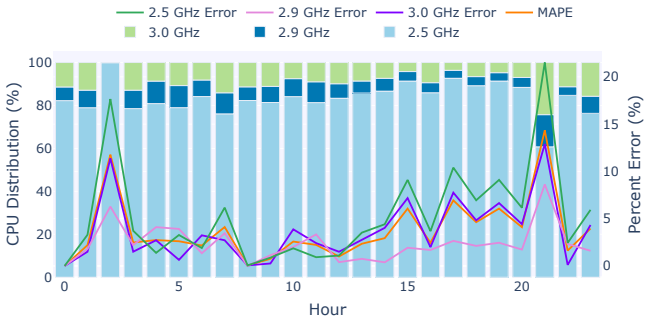


Figure 8: Change in CPU distribution characterization accuracy over 24 hours on us-west-1b compared to initial poll.

distribution varied by zone over time. **Across all days, it took on average 5.65 sampling polls to achieve 95% characterization accuracy.** 1.41, 2.62, and 10.5 polls were needed for 85%, 90%, and 99%, respectively.

We also investigated the temporal stability of hardware characterizations to determine how long a characterization remains valid. Using the CPU distribution obtained on day one as the ground truth, we compared this against the characterizations made for the subsequent thirteen days in each AZ. As shown in Figure 7, some zones, such as ca-central-1a, us-west-1a, and us-west-1b, exhibited considerable changes in infrastructure distribution, with CPU characterization APE (i.e. changes from the day 1 profile) increasing to 20 to 50% as early as day two. In contrast, sa-east-1a and eu-north-1a showed strong temporal consistency, maintaining characterization errors at or below 10% for two-weeks. To understand short-term variability, we performed high-frequency sampling of us-west-1b every hour for 24 hours. Figure 8 presents both the hourly CPU distributions and the resulting characterization error when using the first hour as the baseline. Although this zone showed considerable infrastructure differences for certain hours, the CPU characterization for 22 out of 24 hours had less than 10% variation from the baseline. These findings highlight, while some AZs exhibit long-term stability, others require more frequent observations offering an opportunity to classify AZs behavior to determine sampling requirements. For example, stable AZs require less sampling to save on profiling costs such as sa-east-1a and eu-north-1a, while others like ca-central-1a and us-west-1a may require more samples.

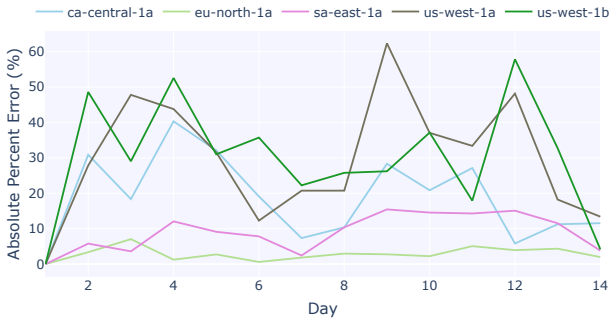


Figure 7: Characterization accuracy degradation over time.

4.5 EX-5: Performance Enhancement by Smart Routing

To evaluate the potential of smart routing strategies, we profiled 12 functions including computational, I/O-bound, and mixed-use cases, as listed in Table 1. Each function was executed 10,000 times across multiple AWS Lambda AZs to capture performance across the four major CPUs used by the platform. Figure 9 summarizes the relative performance of each function by CPU, normalized to the Intel Xeon 2.5 GHz processor, the most common hardware variant observed. A clear performance hierarchy emerged: the Intel Xeon 3.0 GHz processor consistently delivered the fastest runtimes, showing 5% to 15% improvements for most functions. The 2.9 GHz processor was generally slower than the baseline by 15–30%, and the AMD EPYC processor was the slowest overall, with up to 50% longer runtimes for functions such as logistic_regression and math_service. Notably, a few exceptions were observed—disk_writer, disk_write and_process, and sha1_hash—where performance deviated from this trend. In particular, the AMD EPYC processor slightly outperformed the baseline for disk_writer, suggesting that certain functions may be less sensitive to raw CPU speed and more affected by cache, I/O handling, or other architectural features. Our profiling results underscore the potential benefits of routing decisions that are sensitive to hardware heterogeneity.

To evaluate the effectiveness of our retry-based optimization strategies, we leveraged average function runtime observations on

Figure 9: Workload performance normalized to the performance of the 2.5 GHz Processor on AWS Lambda.

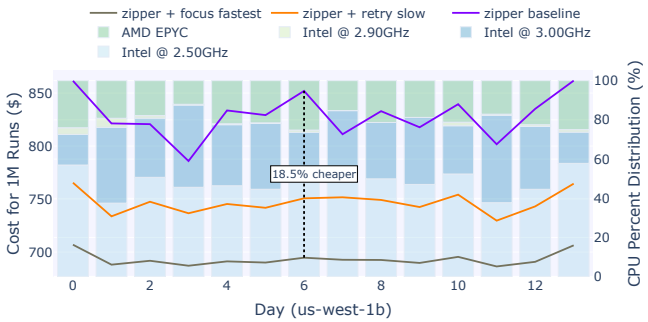


Figure 10: Zipper function performance across a constantly changing CPU distribution with different retry methods.

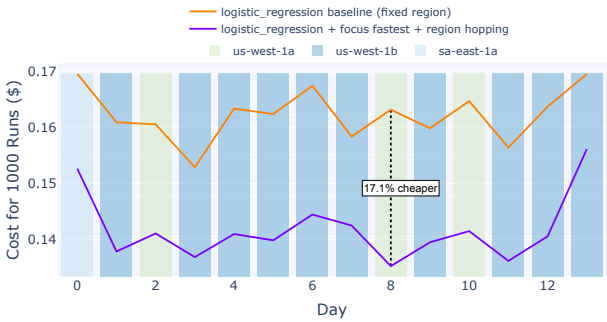


Figure 11: Logistic Regression cost using a hybrid retry and region hopping method compared to running in us-west-1b.

all available CPUs combined with our two weeks of CPU distribution data gathered for EX-4. For each function, we calculated the expected runtime over time based on the evolving CPU distribution in each AZ. We then executed bursts of 1,000 function invocations using two variants of the retry method: *retry slow* and *focus fastest*. In the *retry slow* strategy, invocations were re-tried only when placed on the two slowest CPUs, typically AMD EPYC and the 2.9 GHz Xeon. In contrast, the *focus fastest* method aggressively retried any invocation not placed on the 3.0 GHz Xeon, the fastest-performing processor. Aggressive retrying introduces overhead due to client calls queuing up and having to wait from reattempting failed invocations. In zones with diverse CPUs, we found that performance gains often outweighed these queuing costs. As shown in Figure 10, the zipper function benefited substantially from smart retrying. **Over the two-week period, the *focus fastest* method achieved a cumulative 16.5% cost savings, with a maximum 18.5% single-day cost savings despite retrying more than 50% of invocations.** The more conservative *retry slow* method yielded a consistent 10.1% reduction in execution cost throughout the 2-week period. This example highlights a scenario where aggressive retries can lead to substantial cost savings.

Beyond single-zone retries, we explored a multi-region optimization strategy that dynamically selected execution AZs based on daily CPU characterizations. This *region hopping* approach prioritized AZs with the highest prevalence of high-performance CPUs, most notably the 3.0 GHz Xeon, on a given day. By continuously adapting to the observed infrastructure composition, region hopping avoids persistently degraded zones and instead routes function calls to zones that offer favorable execution conditions. This technique can be used independently or in combination with our retry method to compound performance gains. Figure 11 demonstrates the effectiveness of region hopping using the logistic_regression function. The baseline approach fixed all requests to us-west-1b and is compared with a smart routing approach that switches between us-west-1a, us-west-1b, and sa-east-1a, as shown by the shaded sections in the plot. **The logistic regression function with region hopping method achieved the highest overall cost reduction of 13.3% over the 2-week period, with a maximum single day savings of 17.1% even accounting for the additional runtime due to retries.** The graph breadth-first search function (graph-bfs) with the hybrid approach of region hopping and retries had the highest total savings throughout the 2-week period of 18.2% compared to running in us-west-1b. **Averaged for all functions, our hybrid approach achieved an average cost savings of 10.03% with a standard deviation of 3.70%.** Only $2.80 was spent performing infrastructure characterizations.

4.6 Limitations and Future Work

The cost improvements we have demonstrated come with an inherent trade-off in added latency. For the retry method, a minimum delay of 150 milliseconds is introduced per round to attempt to defer execution in favor of identifying a higher performing FI. This added latency makes the approach particularly well suited for asynchronous background tasks or high-concurrency batch workloads where response time is not critical. For a 1,000 invocation workload on us-west-1b, an average of 5 retries are needed to have all requests run on the 3.00 GHz processor, resulting in an added cost of $0.03 which is easily made up by the performance improvement. In cases where only a small number of parallel invocations are issued, the opportunity to encounter optimal hardware is reduced, potentially diminishing the benefits of retries. The retry method remains a practical strategy for many applications that prioritize cost efficiency over low latency–such as RNA sequencing or batch workloads [23].

Our infrastructure observation technique is highly cost-effective, requiring only $0.04 to characterize a single AZ, but this overhead can accumulate as more zones are profiled regularly. However, since CPU characterizations are independent of specific workloads, profiling can be performed once and used repeatedly to inform scheduling decisions for a wide range of functions. Our findings suggest that cloud providers could improve developer outcomes by offering greater visibility into infrastructure characteristics, reducing the need for active probing altogether. However, this highlights the potential value of incorporating infrastructure characterization directly into a sky computing middleware layer. In future work, to completely eliminate the overhead of polling, hardware characterizations can be constructed passively as part of the normal function execution.

5 Conclusions

This study introduces important advances for serverless sky computing, including dynamic functions, a novel infrastructure profiling method, and a smart routing mechanism to optimize performance and cost. By deploying functions across a broad *'sky'* of cloud environments and observing underlying hardware characteristics, we exposed the hidden heterogeneity in serverless infrastructure (RQ-1). Our study, conducted at a global scale, profiled serverless platforms in 41 cloud regions across AWS Lambda, IBM Code Engine, and Digital Ocean Functions. Our sampling technique was able to accurately characterize the available infrastructure of an AZ for only $0.04 (RQ-2) with 95% accuracy. Our analysis revealed heterogeneous CPU configurations in different zones and helped inform our smart routing approach to aggregate resources from many AZs to utilize the highest-performance infrastructure available.

Empirical results demonstrate that harnessing infrastructure heterogeneity can yield substantial improvements in hosting costs (RQ-3). An aggressive retry strategy that reschedules functions on faster CPUs achieved up to 18.5% daily cost savings compared to the baseline execution. A hybrid approach that combines multi-region routing with retries consistently delivered cost reductions, up to 18.2% cumulative savings, with only $2.80 spent on sampling over two weeks. These gains underscore the broader implications of our approach: By dynamically adapting to heterogeneous infrastructure on a global scale, serverless applications can turn hardware variability into an advantage. Sky computing for serverless offers a promising path toward more cost-efficient, high-performance, and scalable serverless computing, enabling cloud functions to capitalize on the best available resources worldwide. All source code and data sets are available on GitHub [5].

Acknowledgments

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

References

[1] 2025. Amazon S3. https://aws.amazon.com/s3
[2] 2025. AWS Lambda. https://aws.amazon.com/lambda
[3] 2025. Google Cloud Storage. https://cloud.google.com/storage
[4] 2025. Python. https://www.python.org
[5] 2025. SAAF: Serverless Application Analytics Framework. github.com/wlloyduw/SAAF
[6] 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 Symp. on networked systems design and implementation (NSDI 20)*. 419–434.
[7] Ataollah Fatahi Baarzi, George Kesidis, Carlee Joe-Wong, and Mohammad Shahrad. 2021. On merits and viability of multi-cloud serverless. In *Proc. of the ACM Symp. on Cloud Computing*. 600–608.
[8] Sarah Chasins, Alvin Cheung, Natacha Crooks, Ali Ghodsi, Ken Goldberg, Joseph E Gonzalez, Joseph M Hellerstein, Michael I Jordan, Anthony D Joseph, Michael W Mahoney, et al. 2022. The sky above the clouds. *arXiv preprint arXiv:2205.07147* (2022).
[9] Xinghan Chen, Ling-Hong Hung, Robert Cordingly, and Wes Lloyd. 2023. X86 vs. ARM64: An Investigation of Factors Influencing Serverless Performance. In *Proc. 9th Intl. Workshop on Serverless Computing (WoSC '23) co-located with ACM/IFIP Middleware 2023*. ACM, 7–12. https://doi.org/10.1145/3631295.3631394
[10] Marcin Copik, Grzegorz Kwasniewski, Maciej Besta, Michal Podstawski, and Torsten Hoefler. 2021. Sebs: A serverless benchmark suite for function-as-a-service computing. In *Proc. of the 22nd Inter. Middleware Conf.* 64–78.
[11] Robert Cordingly. 2021. Serverless Performance Modeling with CPU Time Accounting and the SAAF Framework.
[12] Robert Cordingly, Jasleen Kaur, Divyansh Dwivedi, and Wes Lloyd. 2023. Towards serverless sky computing: An investigation on global workload distribution to mitigate carbon intensity, network latency, and cost. In *2023 IEEE Inter. Conf. on Cloud Eng. (IC2E)*. IEEE, 59–69.
[13] Robert Cordingly and Wes Lloyd. 2022. FaaSET: A Jupyter notebook to streamline every facet of serverless development. In *Companion of the 2022 ACM/SPEC Inter. Conf. on Performance Eng.* 49–52.
[14] Robert Cordingly, Wen Shu, and Wes J Lloyd. 2020. Predicting Performance and Cost of Serverless Computing Functions with SAAF. In *6th IEEE Int. Conf. on Cloud and Big Data Computing (CBDCOM 2020)*.
[15] Benjamin Farley, Ari Juels, Venkatanathan Varadarajan, Thomas Ristenpart, Kevin D Bowers, and Michael M Swift. 2012. More for your money: exploiting performance heterogeneity in public clouds. In *Proc. of the Third ACM Symp. on Cloud Computing*. 1–14.
[16] Anshul Jindal, Julian Frielinghaus, Mohak Chadha, and Michael Gerndt. 2021. Courier: Delivering Serverless Functions Within Heterogeneous FaaS Deployments. In *Proc. of the 14th IEEE/ACM Inter. Conf. on Utility and Cloud Computing*. ACM.
[17] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, and Others. 2019. Cloud programming simplified: a berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383* (2019).
[18] Daniel Kelly, Frank Glavin, and Enda Barrett. 2020. Serverless computing: Behind the scenes of major platforms. In *2020 IEEE 13th Inter. Conf. on Cloud Computing (CLOUD)*. IEEE, 304–312.
[19] Danielle Lambion, Robert Schmitz, Robert Cordingly, Navid Heydari, and Wesley Lloyd. 2022. Characterizing X86 and ARM Serverless Performance Variation: A Natural Language Processing Case Study. In *Companion of the 2022 ACM/SPEC Inter. Conf. on Performance Eng. (ICPE '22 Companion)*. ACM. https://doi.org/10.1145/3491204.3527465
[20] 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 Inter. Conf. on Cloud Eng. (IC2E)*. IEEE, 159–166.
[21] Wes Lloyd, Minh Vu, Baojia Zhang, Olaf David, and George Leavesley. 2018. Improving application migration to serverless computing platforms: Latency mitigation with keep-alive workloads. In *2018 IEEE/ACM Inter. Conf. on Utility and Cloud Computing Companion (UCC Companion)*. IEEE, 195–200.
[22] Yunhao Mao. 2022. SkyBridge: A Cross-cloud Storage System for Sky Computing. In *23rd Inter. Middleware Conf. Ph.D. Symp.* ACM, 15–17.
[23] Xingzhi Niu, Dimitar Kumanov, Ling-Hong Hung, Wes Lloyd, and Ka Yee Yeung. 2019. Leveraging serverless computing to improve performance for sequence comparison. In *Proc. of the 10th ACM international conference on bioinformatics, computational biology and health informatics*. 683–687.
[24] Zhonghong Ou, Hao Zhuang, Andrey Lukyanenko, Jukka K Nurminen, Pan Hui, Vladimir Mazalov, and Antti Ylä-Jääski. 2013. Is the same instance type created equal? exploiting heterogeneity of public clouds. *IEEE transactions on cloud computing* 1, 2 (2013), 201–214.
[25] Josep Sampé, Pedro García-López, Marc Sánchez-Artigas, Gil Vernik, Pol Roca-Llaberia, and Aitor Arjona. 2021. Toward Multicloud Access Transparency in Serverless Comp. *IEEE Software* 38, 1 (2021), 68–74.
[26] Trever Schirmer, Valentin Carl, Nils Höller, Tobias Pfandzelter, and David Bermbach. 2025. Minos: Exploiting Cloud Performance Variation with Function-as-a-Service Instance Selection. In *2025 IEEE Inter. Conf. on Cloud Engineering (IC2E)*. 194–199. https://doi.org/10.1109/IC2E65552.2025.00036
[27] Trever Schirmer, Nils Japke, Sofia Greten, Tobias Pfandzelter, and David Bermbach. 2023. The Night Shift: Understanding Performance Variability of Cloud Serverless Platforms. In *Proc. of the 1st Workshop on Serverless Systems, Applications and Methodologies (SESAME '23)*. ACM. https://doi.org/10.1145/3592533.3592808
[28] Christopher Peter Smith, Anshul Jindal, Mohak Chadha, Michael Gerndt, and Shajulin Benedict. 2022. Fado: Faas functions and data orchestrator for multiple serverless edge-cloud clusters. In *2022 IEEE 6th Inter. Conf. on Fog and Edge Computing (ICFEC)*. IEEE, 17–25.
[29] Ion Stoica and Scott Shenker. 2021. From cloud computing to sky computing. In *Proc. of the Workshop on Hot Topics in Operating Systems*. 26–32.
[30] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking Behind the Curtains of Serverless Platforms. *2018 USENIX Annual Technical Conf. (USENIX ATC 18)* (2018).
[31] Zongheng Yang, Zhanghao Wu, Michael Luo, Wei-Lin Chiang, Romil Bhardwaj, Woosuk Kwon, Siyuan Zhuang, Frank Sifei Luan, Gautam Mittal, Scott Shenker, et al. 2023. SkyPilot: An Intercloud Broker for Sky Computing. In *20th USENIX Symp. on Networked Systems Design and Implementation (NSDI 23)*. 437–455.