Managing Vendor Lock-in in Serverless Edge-to-Cloud Computing from the Client Side

Haidong Zhao

Managing Vendor Lock-in in Serverless Edge-to-Cloud Computing from the Client Side

Haidong Zhao

November 15, 2022

MASTER'S THESIS

Mobile Cloud Computing Chair

Institut für Kommunikationssysteme

Fakultät IV

Technische Universität Berlin

Examiner 1: Prof. Dr.-Ing. David Bermbach

Examiner 2: Prof. Dr.-Ing. Stefan Tai

Advisor 1: Dr.-Ing. Nikolaos Georgantas

Advisor 2: M.Sc. Tobias Pfandzelter

Sworn Affidavit

I hereby declare that the thesis submitted is my own, unaided work, completed without any unpermitted external help. Only the sources and resources listed were used.

The independent and unaided completion of the thesis is affirmed by affidavit:

Paris, November 15th, 2022

Haidong Zhao

Abstract

Serverless computing is a widely adopted cloud execution model composed of Function-as-a-Service (FaaS) and Backend-as-a-Service (BaaS) offerings. The increased level of abstraction makes vendor lock-in inherent to serverless computing, raising more concerns than previous cloud paradigms. Multi-cloud serverless is a promising emerging approach against vendor lock-in, yet multiple challenges must be overcome to tap its potential. First, we need to be aware of the performance and cost of each FaaS provider. Second, a multi-cloud architecture needs to be proposed before deploying a multi-cloud workflow. Domain-specific serverless offerings must then be integrated into the multi-cloud architecture to improve performance or save costs. Moreover, dealing with serverless offerings from multiple providers is challenging. Finally, we require workload portability support for serverless multi-cloud.

In this thesis, we present a multi-cloud library for cross-serverless offerings. We develop the End Analysis System (EAS) to support comparison among public FaaS providers in terms of performance and cost. Moreover, we design proof-of-concept multi-cloud architectures with domain-specific serverless offerings to alleviate problems such as data gravity. Finally, we deploy workloads on these architectures to evaluate several public FaaS offerings.

Zusammenfassung

Serverloses Computing ist ein weit verbreitetes Cloud-Ausführungsmodell, das aus Function-as-a-Service- (FaaS) und Backend-as-a-Service- (BaaS) Angeboten besteht. Aufgrund der höheren Abstraktionsebene ist die Anbieterbindung beim serverlosen Computing inhärent und wirft mehr Bedenken auf als frühere Cloud-Paradigmen. Multi-Cloud Serverless ist ein vielversprechender neuer Ansatz gegen Vendor Lock-in, doch um sein Potenzial auszuschöpfen, müssen mehrere Herausforderungen bewältigt werden. Erstens müssen wir uns der Leistung und der Kosten jedes FaaS-Anbieters bewusst sein. Zweitens muss eine Multi-Cloud-Architektur vorgeschlagen werden, bevor ein Multi-Cloud-Workflow eingesetzt werden kann. Domänenspezifische serverlose Angebote müssen dann in die Multi-Cloud-Architektur integriert werden, um die Leistung zu verbessern oder Kosten zu sparen. Darüber hinaus ist der Umgang mit serverlosen Angeboten von mehreren Anbietern eine Herausforderung. Schließlich benötigen wir Unterstützung für die Portabilität von Workloads für serverlose Multi-Clouds.

In diesem Beitrag stellen wir eine Multi-Cloud-Bibliothek für serverlose Angebote vor. Wir entwickeln das End Analysis System (EAS), um den Vergleich zwischen öffentlichen FaaS-Anbietern in Bezug auf Leistung und Kosten zu unterstützen. Darüber hinaus entwerfen wir Proof-of-Concept-Multi-Cloud-Architekturen mit domänenspezifischen serverlosen Angeboten, um Probleme wie Datenschwere zu lindern. Schließlich setzen wir Arbeitslasten auf diesen Architekturen ein, um verschiedene öffentliche FaaS-Angebote zu bewerten.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Serverless computing is a promising cloud execution model that supports event-driven resource provisioning while eliminating the need for resource management and allocation. This paradigm enables a fine-grained pay-as-you-go billing model [43]. Given its widespread adoption, cloud vendors improve and differentiate their serverless offerings, aiming at locking in their clients, as has already been the case with previous cloud computing paradigms [49, 51].

Vendor lock-in is exacerbated in serverless computing as public providers only allow customers to use Function-as-a-Service (FaaS) and Backend-as-a-Service (BaaS) offerings [43]. More importantly, a public BaaS offering can only natively trigger a FaaS offering from the same provider. Customers are locked into a single provider due to this *tight coupling*.

Migrating even a simple workload may encounter tricky issues or possibly a dead-end [36, 69, 72]. One solution to mitigate the effect of vendor lock-in is to deploy multi-cloud [16]. Multi-cloud is not only a technical problem but also an economic and social one. Several factors may propel us to adopt and deploy it. First, it can increase availability. Second, we cannot make sure that the currently used FaaS provider is suitable for every workload, as FaaS providers typically have different compute-to-memory ratios, resource management policies [64, 70], and pricing schemes. Third, FaaS providers may offer heavy discounts to

a company that holds a contract with them. Fourth, FaaS is still in its infancy and lacks geo-distributed deployment. To meet regulatory constraints and improve performance, a company may have to adopt a new vendor that enables local FaaS deployment. As a result, we employ an analysis system that takes into account both performance and cost factors, helping in selecting the best-of-breed FaaS provider.

A real workload in serverless computing is typically supported by both FaaS and BaaS [27]. To extend a workload across clouds, a well-designed multi-cloud architecture to manage these serverless offerings should be used. Besides, domain-specific offerings from small providers may need to be integrated into multi-cloud architectures to make up for what dominant public providers cannot support, helping solve some long-lasting issues such as data gravity [28]. As more and more data accumulates, it becomes increasingly difficult for data portability since cloud providers charge prohibitively expensive fees for data egress. Data gravity also worsens the vendor lock-in in serverless computing. Consider that using a hardware-accelerated FaaS to retrieve data from another provider's storage service can incur an expensive data egress fee. On the other hand, it is typically free to move data from a storage service to the same provider's FaaS. Thus, this locks customers to a single provider and may limit serverless applications to simple workloads.

The Serverless Framework [63] helps in deploying serverless applications but may struggle to orchestrate a well-designed multi-cloud architecture. This is due to its limited application template support, especially for multi-cloud. An early effort [9] used it to orchestrate a serverless multi-cloud architecture but left each workload only able to run on a single cloud. This necessitates data synchronization across storage services among different providers, thus resulting in at least double the data egress fee. As pointed out above, domain-specific serverless offerings may also need to be integrated into multi-cloud architectures. This further increases the deployment difficulties. Therefore, a multi-cloud library would be a practical approach to allow easy access to serverless offerings from different providers. Unfortunately, traditional multi-cloud libraries are packaged with heavy dependencies and may incorporate

unrelated modules (e.g., management) for serverless. This bogs down runtime memory and further worsens the cold-start issue. Besides, the library needs to support domain-specific offerings and provide a friendly interface. In light of these requirements, we designed and implemented a multi-cloud library for serverless computing.

1.1 Dissertation Contributions

1. We discuss the benefits of multi-cloud deployments as well as the necessity of redesigning a multi-cloud library for serverless computing (Chapter 2)

2. We develop a library[1] to help in the deployment of a flexible vendor-choice multi-cloud architecture and address issues that we have discussed in the traditional multi-cloud library (Section 3.2)

3. We implement the End Analysis System (EAS) to support the performance evaluation and *fine-grained* cost calculation of public FaaS providers (Section 3.3)

4. We discuss the design of multi-cloud architectures and related opportunities and challenges (Chapter 4)

5. We report the benchmarking results of public FaaS providers. The results suggest that different workloads may require different FaaS providers. Depending on the case, the network traffic fee and latency may also need to be considered (Chapter 4)

1.2 Dissertation Organization

The rest of this dissertation is organized as follows. Chapter 2 reviews background concepts in the field and discusses the motivation for this work; Chapter 3 describes the design prin-

[1]Our software is in progress and we make it available as open-source at https://github.com/hd-zhao/serverless_multicloud

ciples and implementation details of the multi-cloud library, EAS, and relevant supporting tools. Chapter 4 reports our benchmarking results of public FaaS providers in multi-cloud architectures; Meanwhile, we discuss the design of multi-cloud architectures and possible challenges. Chapter 5 discusses some design choices and limitations of this work, and potential improvements in the future. Chapter 6 discusses the related techniques and systems to this dissertation. Chapter 7 presents the summary and concluding remarks

Chapter 2

Background and Motivation

In this chapter, we provide the background and motivation for this thesis. In Section 2.1, we list the main public and open-source FaaS providers and summarize some limitations of public FaaS providers. In Section 2.2, we classify multi-cloud deployments into three categories in terms of their workload distribution.

Adopting multi-cloud implies incurring extra investments. We suggest the benefits and potential demands of serverless multi-cloud in Section 2.3. To really reap the benefits of multi-cloud, workload portability across clouds should be achieved. We think the multi-cloud library would be a practical approach to achieve this goal. However, traditional multi-cloud libraries with unrelated function modules and heavy dependencies risk running out of allocated memory for a function and worsening the cold-start issue. Thus, we elucidate the requirements a serverless multi-cloud library should satisfy in Section 2.4.

2.1 Serverless Computing

Many applications have benefited from serverless computing [18, 30, 44, 67]. FaaS is the core of it [43]. Public FaaS providers include Amazon Web Services (AWS) with AWS Lambda [7], Google's Cloud Functions (GCF) [20], Alibaba's Function Compute (AFC) [31], Microsoft's Azure Functions [10], etc. Although public FaaS offerings are billed as usage, the billing duration of a function instance is slightly longer than the actual execution time. This increment depends on the billing granularity of each provider; e.g., billing granularity for AWS Lambda is 1 ms; for AFC and GCF, it is 100 ms [8, 21, 32].

Despite the potential of serverless, there are a few limitations to be addressed. Here, we summarize some demands targeting the public FaaS providers:

Resource support. FaaS is used for general purpose workloads, as most cloud providers only offer CPU resources for it. However, domain-specific hardware may also provide significant performance improvements and efficiency for workloads with high computational requirements [58]. FaaS providers should provide heterogeneous hardware options.

Resource allocation. FaaS providers have different compute-to-memory ratios. Different workloads have different compute resource requirements. To handle compute-intensive workloads or improve performance, more memory should be allocated for more powerful hardware. This could waste memory resources and increase the cost if the workload is not data-intensive.

Tight coupling between FaaS and BaaS. An operation in BaaS can only be configured to trigger a FaaS offering from the same provider. This limits the interoperability between different providers' FaaS and BaaS offerings. Yet we may redirect the triggered request by using a short-running function instance [16]. However, this may not only lead to double billing and performance degradation but also more complicated debugging environments.

2.2 Multi-Cloud Classification

We investigated multi-cloud deployments and discussions in both industry and academia [19, 40], and classify the multi-cloud in terms of workload distribution:

Each workload only runs on a single cloud. One classical case is hybrid clouds, which migrate some workloads to public clouds. According to a survey [1], the hybrid cloud approach dominates the multi-cloud market. Another representation of this classification is running segmented solutions in different clouds. For example, using Google Docs for collaborative writing and Outlook's email service.

A workload runs across multiple clouds. This refers to the workload migration enabled. This deployment requires us to choose shared services among providers. Deploying a workload across multiple clouds can improve availability and help in finding the best FaaS provider for each workload.

A workload is divided among multiple clouds. A pipeline task or a workload can be separated into several stages. At each stage, we can choose an optimal cloud offering instead of all stages in a single cloud. Researchers from Berkeley proposed a transparent cloud targeting this objective [19, 68]. In a scenario where a FaaS offering collaborates with another provider's BaaS offering, it falls under this category.

2.3 How to Benefit From Multi-Cloud Deployment

Mitigate the risks of data gravity and being locked into a FaaS provider. Ingress data is typically free of charge by cloud providers. However, when data flees from cloud providers to the Internet, providers usually charge a data egress fee, as shown in Table 2.1. Consider if we want to retrieve 10 TB of data from a storage site in London and the storage service is provided by Alibaba, we need to pay $143.36 to save a month and $716.8 to move

Table 2.1: The pricing strategy of some public FaaS providers

Provider	Duration Fee (GB-second)		Invocation Fee (million)		Data Egress Fee (GB / Month)		Ephemeral Storage Fee (GB-second)	
	Free quotas	*Beyond*	*Free quotas*	*Beyond*	*Free quotas*	*Data Egress*	*Free quotas*	*Beyond*
AWS Lambda[a][8]	400,000	$\$1.67 \times 10^{-5}$	1	\$0.20	100 GB	\$0.09	512 MB	$\$3.58 \times 10^{-8}$
Google Cloud Functions[b][21]	400,000	$\$2.5 \times 10^{-6}$	2	\$0.40	5 GB	\$0.12	Integrated with memory	
Alibaba Function Compute [32]	400,000	$\$1.67 \times 10^{-5}$	1	\$0.20	\$0.07 in Frankfurt and London		512 MB	NAS File System

[a] The x86 architecture pricing scheme, and the data egress fee is tiered pricing: \$0.09 in the first 10 TB, \$0.085 in the next 40 TB, etc.

[b] Tier 1 pricing. GCF also charges for CPU time.

out, 5 times the price difference. With Amazon S3 [3], \$239.76 for saving a month and \$891 for moving out. Even if moving out data with high bandwidth just takes minute-level time, it is much more costly than storing it for a month.

This is data gravity. Customers tend to store their data with one provider for performance considerations. Therefore, with data growing, it is more unlikely to move data out with the prohibitively expensive data egress fee. Moreover, FaaS is a data-shipping architecture [39]. Data is unidirectionally moved to the FaaS. Outbound data from an object storage site to the same provider's FaaS is normally free, but to the Internet is costly, especially for large-scale data. This worsens the lock-in issue and precludes customers from using other domain-specific FaaS (e.g., a FaaS offering with hardware acceleration). In light of this, we must employ some low or free egress fee storage services to prevent data gravity from happening.

Geo-distributed demands and regulatory requirements. Public FaaS providers usually have deployments in metropolises but with sporadic deployments in other regions, especially developing countries. To adopt serverless computing, a company in this greenfield may use a new provider's FaaS for its local deployment. On the other hand, data protection mandates would restrict data transfer to protect privacy data [54]. This further calls for local processing capability. As described above, a FaaS platform interoperating with another provider's BaaS offering is also a kind of multi-cloud.

Table 2.2: vCPU quotas with allocated memory

Provider	**Memory**						
	128MB	*256MB*	*512MB*	*1GB*	*2GB*	*4GB*	*8GB*
AWS Lambda[a][25]	0.072	0.144	0.288	0.579	1.16	2.32	4.64
Cloud Functions [21]	0.083	0.167	0.333	0.583	1	2	2
Function Compute	$0.05 \sim 0.1$	$0.1 \sim 0.25$	$0.15 \sim 0.5$	$0.25 \sim 1$	$0.5 \sim 2$	$1 \sim 4$	$2 \sim 8$

[a] Its compute power is in proportion to the allocated memory, with 1769 MB of memory allocation, a function has the equivalent of 1 vCPU.

Increase availability, optimize performance and reduce cost. Compared with traditional compute instances, deploying multi-cloud with FaaS offerings to increase availability does not suffer from being double charged because of its pay-as-you-go billing model [43].

Public FaaS providers have different compute-to-memory ratios, resource management policies [70], pricing schemes, etc. One FaaS provider may perform well for one workload but may be clumsy in another. This requires us to select the best-of-breed FaaS provider based on the characteristics of running workloads. Memory allocation value is an important factor, as it not only ensures if the memory is enough for running workloads but also determines how many hardware resources are allocated to the function. Table 2.2 shows how many vCPUs a function instance can get with its allocated memory at some public FaaS providers. This information would influence the choice of the best provider for a given memory allocation. Power hardware incurs more expenses per time unit but may shorten the billing duration, resulting in a lower execution cost. Therefore, we need to consider both performance and cost factors for memory allocation.

Mitigate the risk of an application being locked to a serverless provider. Due to the lack of standard and proprietary serverless offerings among providers, we may find it challenging or impossible to migrate an application [36, 72]. Multi-cloud deployment suggests choosing common services in order to preclude being locked into a vendor.

2.4 Redesign Multi-Cloud Library

As pointed out in Chapter 1, using the Serverless Framework faces problems when deploying a flexible vendor-choice multi-cloud architecture. We suppose that the multi-cloud approach currently is the most feasible approach to manage FaaS and BaaS offerings from different providers while enabling a flexible vendor-choice multi-cloud architecture.

Previous multi-cloud libraries, such as Apache jclouds [4], Apache Libcloud [5], fog [29], are primarily used for traditional cloud services and include general services such as DNS and compute instance management. Using these irrelevant libraries and dependencies involves more memory usage on the function instance, resulting in the need to allocate more memory to support it. Besides, a function instance needs more time to download the cumbersome libraries when cold-start happens. In light of this, we consider that a serverless multi-cloud library should be proposed while satisfying the following requirements.

Versatile and Scalable. In addition to supporting backend off-the-shelf services from well-known public clouds, the serverless multi-cloud library should take into account some small providers who may offer domain-specific services.

Conversion-enabled and one-for-all. One is able to use the multi-cloud library to access any BaaS provider's services with ease. Besides, we also account for a situation where a company and its engineers are integrated and trained by a cloud provider, respectively. If this company plans to port some workloads to other providers, the multi-cloud library should also provide the conversion from one provider's software development kits (SDKs) to that of another one. This allows engineers to access different providers' serverless offerings with their accustomed SDKs.

Lightweight and clean dependency. A FaaS platform initially offers an underlying execution environment (runtime) for multiple high-level programming languages and their corresponding standard libraries. We need to upload the code and, perhaps with its supported

third-party libraries, to runtime. Using too many third-party modules or bundling heavy dependencies of multi-cloud libraries would result in additional memory allocation and worsen the cold start since an instance should spend more time preparing for the runtime (i.e., downloading code and third-party libraries from persistent storage). Therefore, a clean-dependency multi-cloud library that only downloads needed libraries to runtime can avoid this problem.

Chapter 3

System Design and Implementation

In Section 3.1, we describe the system design from a high level. As depicted in Fig. 3.1, the system is composed of three parts: the workload deployed in the multi-cloud architecture; the multi-cloud library (Section 3.2), which enables the utilization of different providers' FaaS and BaaS offerings (common services) with ease; and the End Analysis System (Section 3.3), which supports performance evaluation and *fine-grained* cost calculation for several public FaaS providers. We observed the time drift happening in function instances in some FaaS providers, and this issue is especially serious in AFC. To measure *end-to-end delay*, it is indispensable to require (approximate) actual time from function instances. Therefore, we introduce a countermeasure in Section 3.4.

3.1 System Design Overview

As described in Section 2.3, a workload may be divided into several stages. Each workload/-task stage may work on the most optimized cloud offering. Consider that if a workload stage is to store infrequently used data, it may opt for a storage service with the lowest fee for long-term data storage. On the other hand, if this workload stage is for computing, hard-

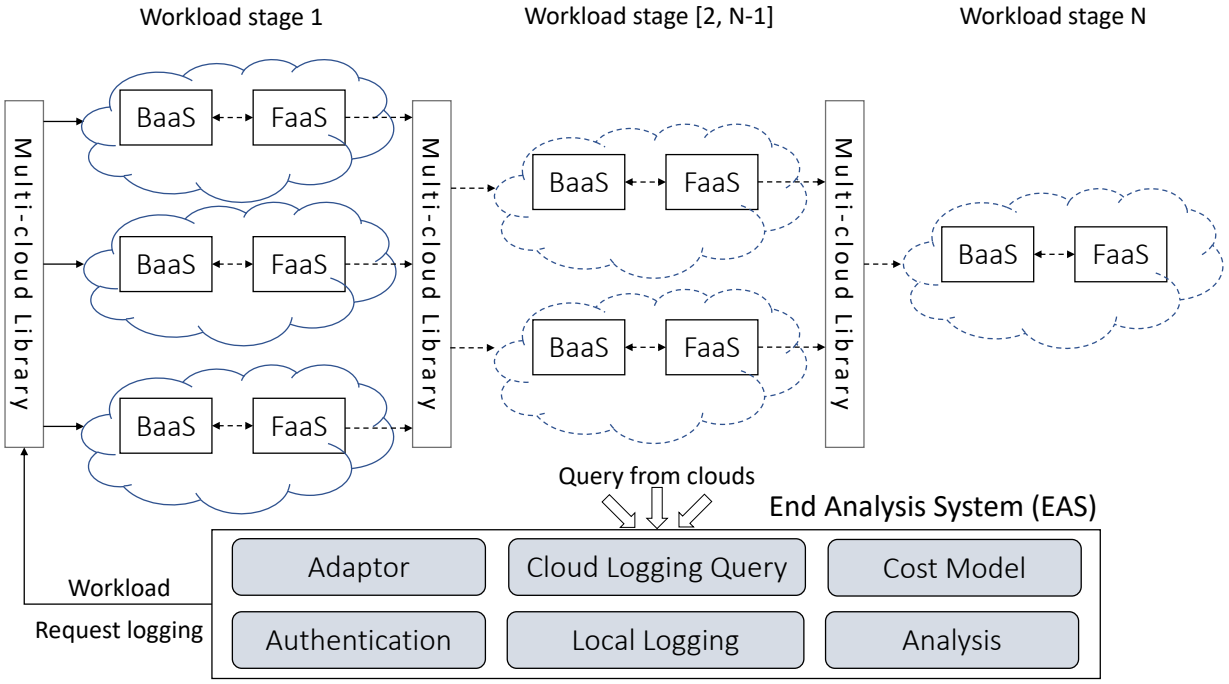


Figure 3.1: System design overview

ware acceleration FaaS may be employed for it. As a result, a workload may be served by more than one cloud provider. In Section 3.2, We introduced a serverless multi-cloud library to facilitate workload portability across clouds. This library attempts to achieve the goals discussed in Section 2.4.

As discussed in Section 2.3, one incentive to deploy multi-cloud is to select the best-performance FaaS provider for each workload, as there are heterogeneous pricing strategies, compute-to-memory ratios, and resource management policies among providers. Thus, In Section 3.3, we detail EAS, which enables an accurate performance evaluation and cost calculation. We suggest that we need to consider not only *response time* but also *end-to-end delay*. As an invocation may be served by a BaaS offering before invoking FaaS, we may need to know how long it takes for a request to be served. However, we observed serious time drift in function instances, thus, we provide a countermeasure in Section 3.4. Besides, the cost of running a workload with FaaS is composed of multiple parts, as shown in Table 2.1. It would be naive

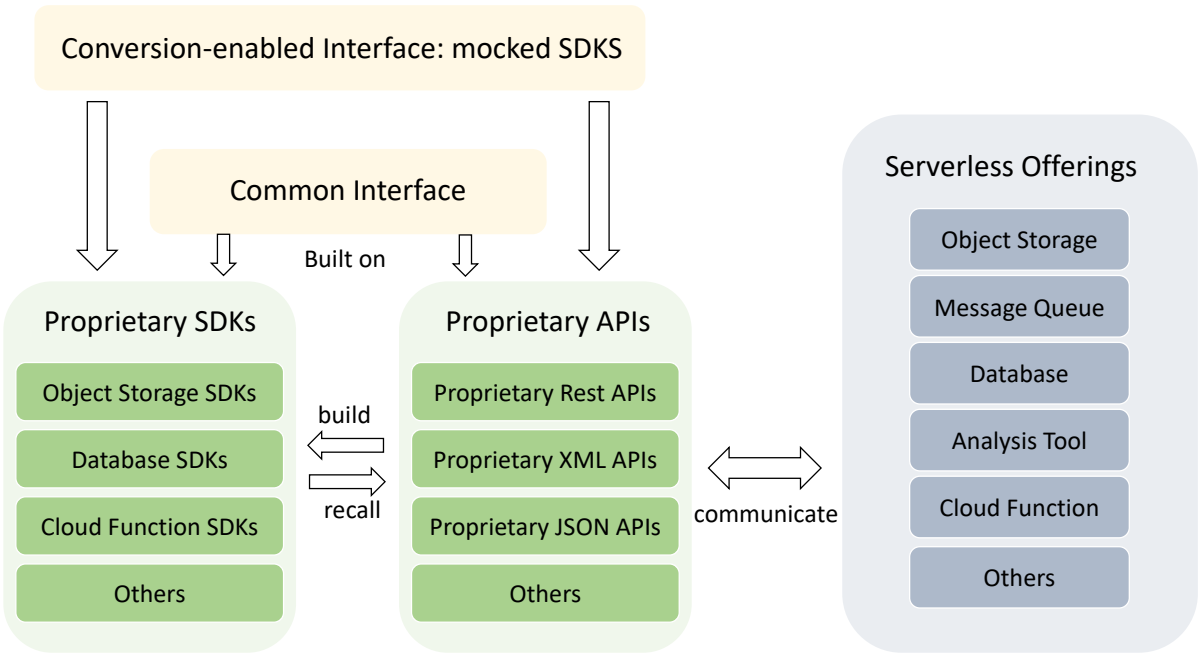


Figure 3.2: Multi-cloud library architecture

to equal the FaaS cost to the duration fee when the data egress fee cannot be ignored. Thus, EAS also supports a *fined-grained* cost calculation, clarifying each part's expenses.

3.2 Multi-Cloud Library Design and Implementation

Fig. 3.2 depicts the architecture of the proposed multi-cloud library. Public cloud providers provision serverless offerings (e.g., cloud functions, object storage services, message queues), allowing customers to access these services with their proprietary SDKs and/or APIs. Compared with APIs, customers prefer to use SDKs, which are typically well-documented and user-friendly. Thus, our proposed multi-cloud is mainly built on top of SDKs unless they are unavailable (this could happen for a cloud provider's new service).

First, we provide a common interface that is on top of the proprietary SDKs and/or APIs of cloud services, as other multi-cloud libraries do. Second, we also consider the situation

where a company has already been deeply integrated with a cloud provider and is trained with its products. This might make the company hesitant to adopt multi-cloud. To allow a gentle learning curve, therefore, we further provide a conversion-enabled interface on top of the proprietary SDKs and APIs. This conversion-enabled interface (aka mocked SDKs) retains the syntax and semantics of the translated SDKs to the utmost extent. Besides, as cloud providers take over resource provision and management in serverless computing, the library only needs to focus on service utilization. Moreover, we intentionally do not import any dependencies for this multi-cloud library. Developers only package the code with the required proprietary SDKs. This avoids wasting function instance memory and worsening the cold start.

We have implemented a proof-of-concept multi-cloud library for cloud functions and object storage services. We chose Python as the library's first support language, as it is the most popular language for serverless deployment [27] and is almost supported by all cloud providers.

3.3 End Analysis System (EAS)

We developed the EAS to benchmark which FaaS provider is performing best in terms of performance and cost for each workload. The EAS is composed of the following components:

Authentication. To gain access to FaaS and BaaS offerings of different providers, the keys or passwords are saved in configuration files. We make minor modifications to these configuration files in order to integrate them into the multi-cloud library.

Adapter. One benefit of deploying multi-cloud is the ability to select the best FaaS provider for a workload. FaaS-triggered requests are diverse because FaaS can be triggered using HTTP triggers or operations in its triggered BaaS offerings. The adapter is a component that helps to manage these trigger requests in order to switch between benchmarked FaaS

providers with ease. This component is also integrated with the workload generator. At present, we have employed the Poisson arrival process for requests.

Cloud's Logging Query. As Public FaaS is acting as a black box, programmers are only allowed to debug their programs by reading logs from log services offered by the providers. Logging services for FaaS can be asynchronous, such as Amazon CloudWatch[1] for AWS Lambda. Asynchronous logging introduces less overhead compared with synchronous logging. Based on our observations during the work, the cold start of the serverless instances may result in the absence of some logs when only using logging services from providers (typically asynchronous logging). We can use synchronous logging as a workaround as public FaaS is a black box. The synchronous logging method transmits the current states of serverless functions via HTTP [62]. We may also insert some codes to get the status of a serverless instance, e.g., we let the instance record the current timestamp into logs to measure the latency in this thesis. However, to prevent time drift in serverless instances, we describe how we get the actual time for serverless instances in Section 3.4. With heterogeneous cloud logging services, we need to transform the logs to a unified format for further analysis and comparison after the query.

Local Logging. Cloud logging is not enough if we need to further evaluate the performance of FaaS providers. First, the cloud logs only tell us when a serverless instance starts and finishes. To measure the latency between the end and the FaaS, we need to record when the request was sent. Second, we may need to evaluate the response time of the request in order to evaluate the performance of FaaS for some workloads. Moreover, we may need to get the status of serverless instances via HTTP.

Cost Model. Table 2.1 summarizes the pricing schemes of some public FaaS providers we use for evaluation during our writing period. The duration fee, request fee, and network traffic fee are the three main components of the cost of FaaS offerings. We temporarily ignore the deployment and disk space upgrade costs in our cost model. At present, the cost model

[1] https://aws.amazon.com/cloudwatch/

only supports the cost calculation of FaaS offerings. This cost table of FaaS providers is summarized at the time of writing. How to update it as time goes by needs to be discussed. We propose to publish this data on the shared database and maintain the updates on price changes with the help of the community and beneficiaries. A company can download this FaaS cost table from this website and then adjust it if it has a special contract with a cloud provider. We leave the cost calculation and estimation of BaaS offerings to future work.

Analysis. We perform analysis based on the cloud and local logs on two dimensions: performance and cost. For the performance, we focus on the billing duration and latency (request serving latency, response time, etc.). For the cost, we support *fine-grained* estimation, including the duration fee, the invocation fee, the data egress fee, and perhaps disk storage fee. Some providers do not directly support GB-second or GHz-second billing data. Thus, for each provider above, we need to calculate the duration fee based on its billing duration and billing granularity.

3.4 Function Instances Time Drift Alleviation

The latency between the end system and code execution should be evaluated, as we should know when the request is served in addition to the billing duration. Besides, when a timeout occurs, we should keep track of when the request will be reserved again, especially at the moment of a high volume of requests. However, we found a serious time drift in some serverless function instances. We cannot rely on the actual time provided by serverless instances anymore, or it does not support us measuring the latency at a millisecond level.

Fig. 3.3 depicts a workaround to get the actual time of code execution. We request an Internet time server[2] using the *ntplib* library [48] in t1 and then receive the response with the current UTC time T in t4. We estimate the round-trip time (RTT) to be t4 - t1 and

[2] http://worldtimeapi.org/api/timezone/Europe/London

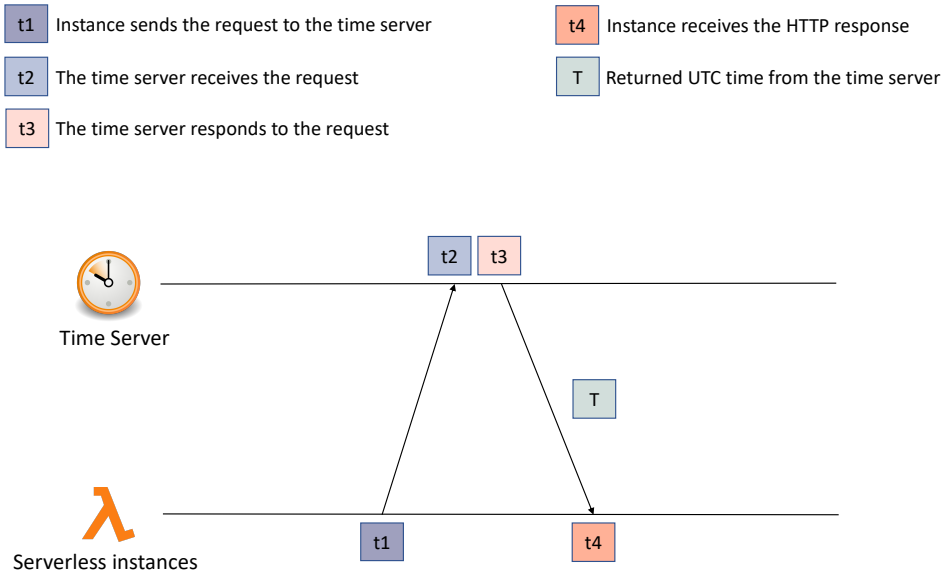


Figure 3.3: Serverless instance requests the actual time from the time server

count this interval with local timer. Finally, we use the value T minus half of the RTT to approximately approach the actual time t1.

Chapter 4

Evaluation

In Section 4.1, we introduce multi-cloud architectures that employ zero-egress-fee data storage service. These two architectures fall into two categories: pipeline-enabled or pipeline-disabled. Then we talk about the challenges of data portability when we employ this storage service. In Section 4.2, we elucidate the benchmarks used for the evaluation. In Section 4.3, we clarify the experimental environment for the evaluation. In Section 4.4, we report our evaluation results and discuss our findings.

4.1 Proof-of-Concept Multi-Cloud Architectures

As discussed in Section 2.3, we may also benefit from the domain-specific services from some small providers. We present two proof-of-concept multi-cloud architectures, in which we employ Cloudflare R2 [22], a zero data egress fee storage service.

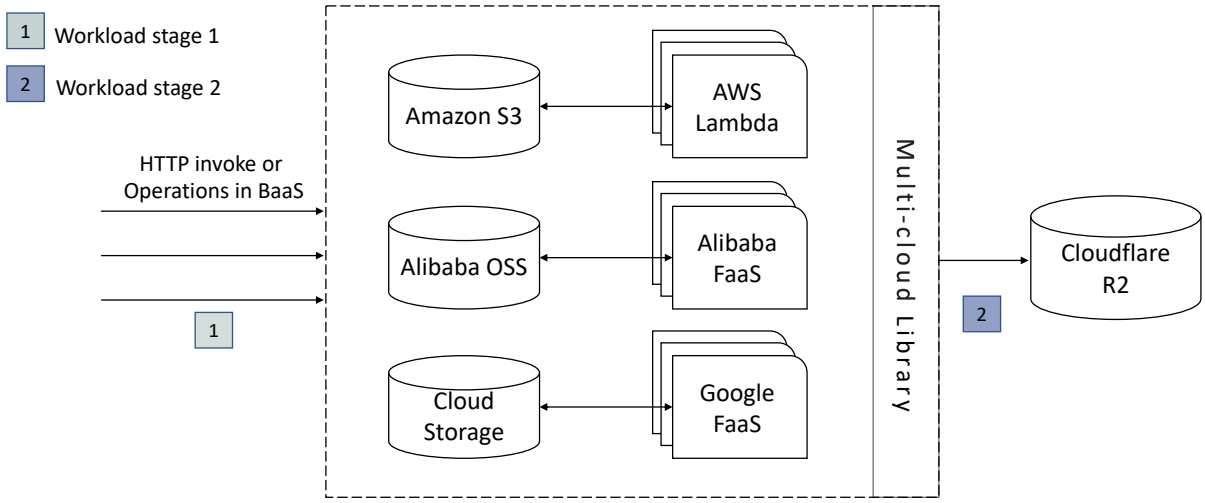


Figure 4.1: Pipeline-enabled multi-cloud architecture

4.1.1 Pipeline-Enabled Multi-Cloud Architecture

Fig. 4.1 depicts a two-stage workload architecture. This architecture enables pipeline processing, as we let a FaaS offering be triggered by a BaaS offering from the same provider.

Optimization 1: Avoid data distribution by saving data at one storage site. The Serverless Framework with limited application templates does not support flexible vendor-choice multi-cloud architectures [9] and may lead to at least double the data egress fee as the data needs to be synchronized across storage providers. What we need to do is to put processed data from FaaS providers into one storage service.

Optimization 2: Leverage the zero egress fee storage service. In Section 2.3, we have discussed the prohibitively expensive data egress fee and how the data gravity issue has limited serverless applications. In light of this, we employ Cloudflare R2 to store data in order to save money on network traffic fees and avoid data gravity.

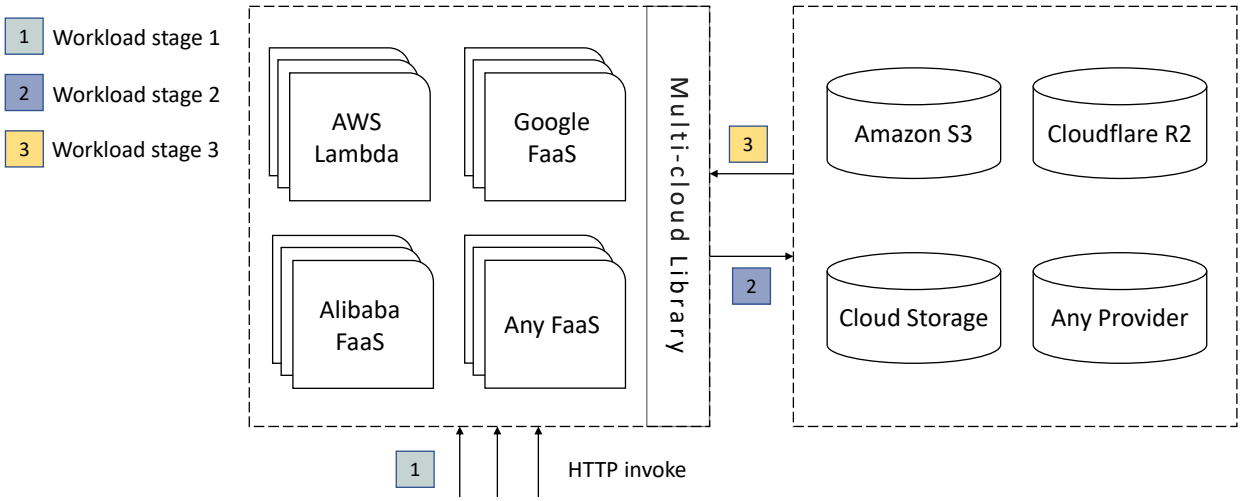


Figure 4.2: Pipeline-disabled multi-cloud architecture

4.1.2 Pipeline-Disabled Multi-Cloud Architecture

Fig. 4.2 depicts a multi-stage workload architecture. In this architecture, we further break the boundary between FaaS and BaaS offerings and do not require that they come from the same provider. Therefore, this architecture does not support pipeline processing anymore, as a public provider does not allow its BaaS offering to trigger a FaaS offering from another provider. In the short term, there are no workarounds for it without performance degradation and cost increase.

This multi-cloud architecture may appear in several scenarios. For example, a company would like to port some workloads to FaaS but may find that its cooperating cloud vendor does not support local FaaS deployment. As a result, in order to satisfy its performance and regulatory requirements, this company may need to look for a new cloud provider. Most public FaaS providers impose time-out limits and rigid compute-to-memory ratios on function instances. Besides, they only support general-purpose hardware. Nimble FaaS providers, which want to attract customers and differentiate their products, may offer solutions in light of this situation. This would attract more customers to migrate more diverse workloads to it.

4.1.3 Data Portability

As discussed in Section 2.3, data gravity is the real threat, especially in serverless computing, hindering data portability. Cloud providers do not impede data movement but charge a prohibitively expensive data egress fee. As a result, cloud customers may find that they are progressively trapped in data lock-in. This motivates us to employ a zero egress fee storage service in the architecture as shown above. How to migrate data from traditional storage services to this zero egress fee storage service needs to be discussed nevertheless. In an optimal situation, newly processed data can be stored in a zero-egress fee storage service first, and then customers can retrieve it from there. This avoids double billing. Otherwise, it should plan how to move redundant data out based on the company's own situation and needs.

4.2 Application Benchmarks

Serverless benchmark workloads can be classified into micro-benchmark or application-oriented benchmarks [23, 34, 45, 71]. To obtain a more practical evaluation of FaaS providers, we focus on application benchmarks. However, to find the bottleneck later, we may also need to add micro-benchmarks.

Image Processing. We chose the image resizing application, specifically, the thumbnail application. The function instance downloads the picture from a bucket to disk space before image processing. We use *PILLOW* library [52] to resize the image to half its original size. Finally, the function instance uploaded the resized image to the object storage service (Cloudflare R2) for persistent storage.

Machine Learning Inference. Machine learning (ML) inference workloads are critical, as the latency of their prediction results has a big effect on customers' satisfaction. In Facebook, the vast majority of online inference runs on the abundant CPU resources [37]. However,

in academia, many works proposed new inference systems leveraging GPU resources [35, 53]. FaaS can also aid the compute instances in coping with the spiked loads to build a cost-effective ML inference system [73]. We load a ResNet-8 model [38] that trained with CIFAR-10 dataset [46]. The original image has already been converted into the ResNet-8 shape required for prediction.

Machine Learning Training. A bulk of papers have discussed how to use FaaS for ML training workloads [15, 41, 57]. There are some positive results in achieving better performance with FaaS for ML workloads, but at a higher cost for the moment. We trained a machine learning model based on a 100 MB truncated dataset collected from Amazon Fine Foods reviews. We applied TF-IDF (term frequency-inverse document frequency) to evaluate the importance a word has to a document among a set of documents and logistic regression model for classification, using the machine learning library *scikit-learn* [61]. After cleaning the dataset, we removed the 100 most infrequently used words in text feature extraction and limited the maximum number of iterations to 1000 in the logistic regression model.

4.3 Experiment Setup

We deployed the End Analysis System on a Google Cloud Engine e2-medium instance with 2 vCPUs and 4 GiB of memory, hosted in London. The workload request is also sent from this instance, which then triggers the serverless platforms. For all FaaS and BaaS offerings, we select the London region for deployment. In this work, we select the serverless providers: AWS, Alibaba, and Google Cloud, with the Python 3.9 runtime for evaluation. Fig. 2.2 shows that AFC supports a little flexibility in compute-to-ratios with its latest version at the time of writing. As AFC's duration does not charge CPU time [32], we select the largest allowed vCPU for a given memory size.

4.4 Experiment Results

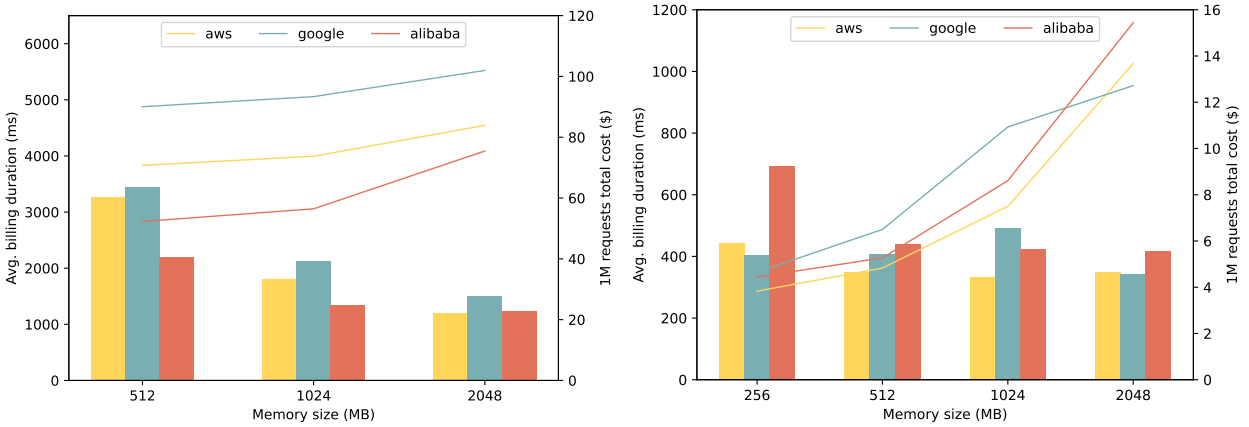
We assume that the user-oriented workload follows a Poisson arrival process with an arrival rate λ. Workloads used to increase productivity, such as ML training and data analysis tasks, can be configured by developers as needed. The cold start issue for these workloads is not as important as for user-oriented workloads, as the importance of latency diminishes.

4.4.1 Best-of-breed FaaS provider selection

Selecting the best FaaS provider for a workload is not trivial, given the heterogeneity in resource management and allocation, pricing strategies, etc. among providers. Therefore, we developed EAS to quantitatively compare FaaS providers.

Image Processing is a common and relatively compute-intensive workload in serverless computing. Fig. 4.3a shows that 1024MB memory size fares well with this workload, as three providers all perform well while maintaining cost-effectiveness. Among the three providers, AFC outperforms its counterpart. This has several reasons. First, AFC provides more powerful computing resources with this memory size, from 512MB to 2056MB (Table 2.2). Besides, the egress data size is relatively large, while AFC charges the lowest network traffic fee (Table 2.1).

ML inference is an interactive workload that is sensitive to latency. We load a tiny inference model due to the computing power of the serverless platform. In this workload, the computing power has a limited impact on the execution time, as shown in Fig. 4.3b. Despite multiple increases in computing power, there is no discernible difference in performance. As a result, a small amount of memory is sufficient to run this workload. AFC does not perform well in this benchmark. This could be due to its resource management policies. Both AWS Lambda and GCF perform well in memory allocations of 215MB and 512MB. However, AWS Lambda might be better because its overall cost is the lowest.



(a) Image Processing benchmark (egress data size size: 482KB)

(b) Machine Learning inference benchmark (egress data size: 10KB)

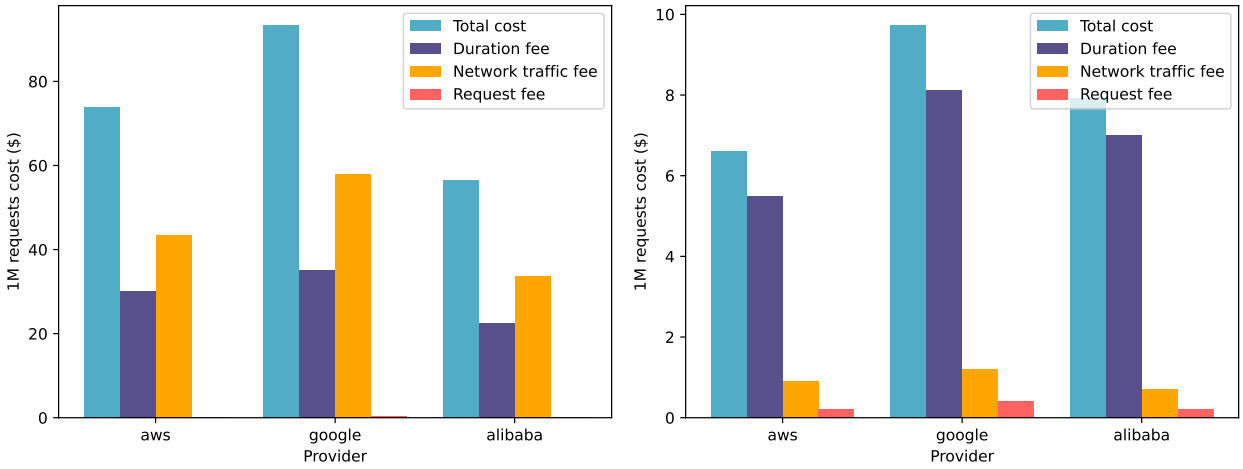
Figure 4.3: Select the best FaaS provider for each workload: Performance and cost co-analysis

These findings suggest that different FaaS providers may be best suited to different workload requirements. Finding the best-of-breed FaaS provider for a workload requires analysis from the perspectives of both performance and cost.

4.4.2 Cost analysis under the microscope

Table 2.1 shows that the cost of FaaS is composed of multiple parts: the duration fee, the request fee, the network traffic fee, etc. We need to consider data egress fees when traffic flows into the Internet instead of the cloud provider's data center networking. In particular, we have proposed to integrate a free egress fee storage service into multi-cloud. Therefore, it certainly leads to network traffic fees. Nonetheless, frequently accessed data can benefit a lot from this storage service. Besides, this helps to alleviate the data gravity that further worsens the lock-in issue.

A pitfall is to simply equal the overall cost to the duration fee in FaaS. This works without considering the network traffic fee and the ephemeral storage fee; the request fee is too tiny to be considered. Fig. 4.4 shows each part of the overall cost in two benchmarks. The average



(a) Image Processing benchmark with 1024MB of memory allocation and 482KB of egress data size per instance. Avg. billing duration in sequence: 1812 ms, 2180 ms, 1346 ms

(b) Machine Learning inference benchmark with 1024MB of memory allocation and 10KB of egress data size per instance. Avg. billing duration in sequence: 332 ms, 492ms, 423ms

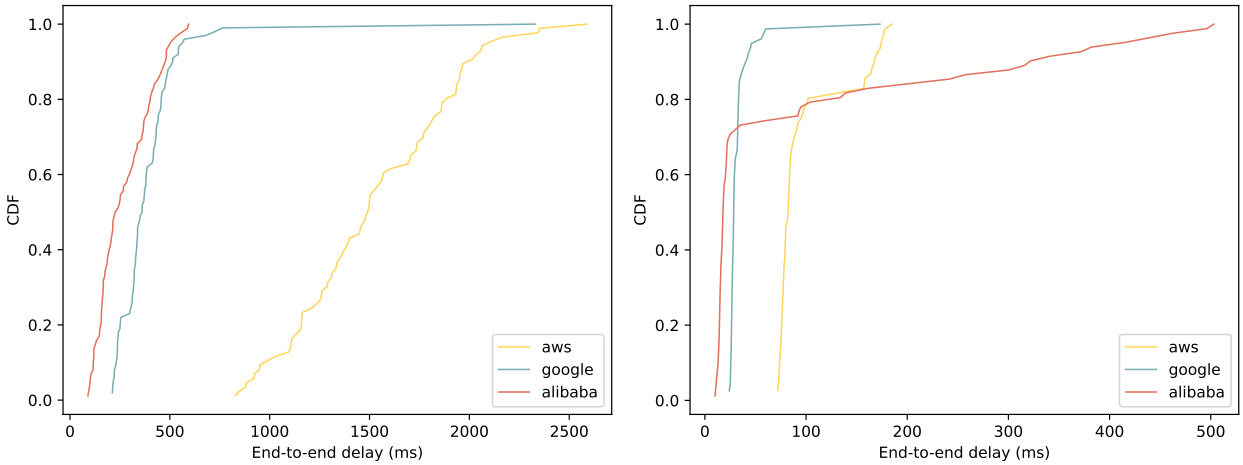
Figure 4.4: Overall cost disassemble

billing duration in the left picture is three times greater than that in the right picture. However, the network traffic is greater than the duration fee in the left picture; on the other hand, the duration fee dominates the overall cost in the right picture. This is due to the fact that large egress data sizes would incur a prohibitively expensive network traffic fee.

4.4.3 Impact of BaaS offerings on latency

Previous benchmarking efforts concentrated on execution time as the main performance indicator. However, a serverless workflow also comprises both FaaS and BaaS offerings. Therefore, the inefficiency in BaaS offerings may also affect the overall performance. The latency should be tolerable, especially for interaction workloads [24].

We conduct experiments in pipeline-enabled multi-cloud (Fig. 4.1) architecture to evaluate it. Fig. 4.5a shows the cumulative distribution function (CDF) of the storage trigger type latency. This latency measures the time from when data is sent from an end-user, then stored



(a) Storage trigger: EAS - object storage - FaaS (b) HTTP trigger: EAS - FaaS

Figure 4.5: End-to-end delay of image resizing application (1024MB of memory allocation) in two workflows after discarding the initial (cold) execution

in storage, until this operation in BaaS triggers the start of a function instance. Fig. 4.5b shows the CDF of the end-to-end delay between the payload sent from the end and the start of the function instance.

AWS's object storage service affects the delay a bit. With the HTTP trigger type, the delay is around 80–200 ms. However, the delay increases to 800–2500 ms if triggered by the storage trigger type. On the other hand, Alibaba's object storage service affects this latency slightly, as the delay in two workflows is within 500 ms. It is a tolerable value. When GCF is triggered by Google's object storage service, only a very small proportion of requests' delays are violently affected. These results show that the BaaS trigger type may have an impact on the latency. However, whether this latency is tolerable depends on the cloud provider selection.

4.4.4 Imbalance of memory allocation and usage

Table 4.1 shows each FaaS provider's compute power when the memory allocation is 2 GB. Even if we chose a small dataset (100 MB) for training traditional ML algorithms, it would

Table 4.1: The FaaS configuration and cost table

	Machine Learning Training		
	AWS	*Google*	*Alibaba*
Data Size	100 MB	100 MB	100 MB
vCPU	1.16	1	1.33
Memory Allocation	2 GB	2 GB	2 GB
Memory Usuage	≈700 MB	≈675 MB	≈720 MB
avg. Billing Duration	133 s	159 s	138 s
1K req. Duration Fee*	$ 4.43	$ 4.64	$ 4.63

take more than 2 minutes to converge. The time-out duration is short in FaaS, which means deploying deep learning algorithms on it will be ineffective for a large training dataset as continued training is needed after a timeout. Besides, we notice the imbalance between memory allocation and usage. We need to allocate more memory to the FaaS in order to gain powerful computing capabilities. However, the computing power increases with the allocated memory for most public FaaS providers. In particular, when training a model with a relatively small dataset, many memory resources will be wasted. We suppose that there would be an opportunity for the FaaS market to support more flexible configurations: GPU enabled (which AFC has enabled), flexible compute-to-memory ratio, etc.

Chapter 5

Discussion

In this chapter, we discuss the design principles and limitations of this dissertation. Meanwhile, we propose potential future improvements.

Benchmarking based on the characteristics of real-world workloads. Benchmarking is to mimic an application as closely as possible to get meaningful results [12]. Thus, the FaaS production workloads' characteristics should be studied. Azure Functions Trace [64] reveals that most serverless applications are invoked infrequently, e.g., 81% of the applications are invoked less than once per minute on average. However, this trace only allows for granularity at 1 minute, we do not know the inter-arrival at the second level. In the benchmarking, we assumed the arrival requests followed the Poisson process. However, bursty or skewed workloads can also be used to evaluate the resource management and scalability of FaaS providers [17, 59, 76]. Currently, most serverless requests are infrequent, so we expect that the distribution choice would not seriously violate the results of this work. Nevertheless, when serverless computing starts to take up a larger part of the cloud business [60], scalability issues will become pressing. A company may increase the power of the function to meet a high volume of concurrent requests in a limited number of provisioned instances, despite incurring a higher cost per function instance. Meanwhile, the machine for workload generation needs

to be deployed on a powerful machine, enabling the generation of a high volume of concurrent requests.

Limitations in selecting the best FaaS provider. We keep the same memory size for each benchmark when selecting the best FaaS provider. However, the resource allocation policies are diverse among FaaS providers. Because there are only a few hardware options for AFC and GCF (as shown in Table 2.2), it is relatively simple to find an optimal resource allocation value that provides high performance while remaining cost-effective. AWS Lambda's hardware resource, on the other hand, is proportional to the allocated memory [25]. There could be a better choice for a selected benchmarked memory size for AWS Lambda in nearby memory size. We may need to conduct several experiments on nearby memory sizes for a possible better choice.

For new FaaS providers and in a production environment. After benchmarking the serverless platforms, we need two phases for EAS to compare FaaS providers: transformation and analysis. In the transforming phase, we need to collect cloud logs from different cloud providers. This phase needs to transform diverse cloud logs into a unified format for later analysis. Therefore, the majority of efforts to welcome new providers concentrate on this phase. With a unified data format, the workloads to accept new providers in the analysis phase are quite low. EAS can also plug into a real production environment. EAS can analyze performance (billing duration) and cost as long as it retrieves logs for its supporting FaaS providers. In a complex FaaS workflow, we can also add logs to function instances and retrieve them back to EAS to obtain additional information, such as latency, for better evaluating and comparing serverless offerings.

Practicability of serverless multi-cloud. The serverless multi-cloud library would facilitate the utilization of serverless offerings. The vision of cross-platform serverless orchestration, on the other hand, is still a long way off. This may be due to the fact that, as the abstraction level of serverless computing increases, the providers take on more control. This could diminish the value of serverless multi-cloud as more human resources are required.

However, there could be performance-cost improvements when adopting multi-cloud, offsetting the increased cost of human efforts. We project that this could happen when serverless represents a higher proportion of a company's infrastructure investment. We are optimistic about the potential vendor lock-in issues in FaaS. Nimble providers who intend to compete with big cloud providers may proactively adopt their standards. This has already happened, as Cloudflare R2 adopted the SDK of AWS's object storage service. On the other hand, regulatory factors may limit the ability of big players to differentiate their products.

Combine with edge cloud. In this thesis, we concentrate on serverless multi-cloud. With the increasing awareness of data privacy and evolving regulatory requirements, edge clouds are gaining wide attention. As for the auto-scaling and pay-as-you-go properties of FaaS, it is promising for the edge cloud to offload computations to serverless to cope with occasional load spikes or for other benefits. However, how to organize resources in this hybrid cloud environment and design workloads is left to future work.

Chapter 6

Related Work

Multi-Cloud. In previous cloud computing paradigms, interoperability and portability have been extensively investigated to alleviate vendor lock-in [33, 51, 75]. Public cloud providers are unlikely to adopt the standards [49]. As a result, client-centric multi-cloud, one of the interoperability approaches, gains more attention and thrives till now [14, 50]. With increased abstraction layers in serverless computing, migrating an even simplified workload is more likely to encounter tricky issues or dead-ends [36, 72]. Moreover, a public BaaS offering can only natively trigger a FaaS offering from the same provider. The serverless multi-cloud approach is not only able to mitigate this concern, but also used to bring additional benefits. Lithops is a multi-cloud serverless computing framework for big data analytics and parallel jobs, supporting running the same code across multiple clouds [56]. Baarzi et al. proposed developing a latency-aware virtual service provider to schedule requests and aggregate resources of public FaaS providers, proving doing so can increase the concurrency limit and availability [11]. As the released Azure Functions Trace [64] reveals, most serverless applications are invoked infrequently. With a default concurrency limit (e.g., AWS Lambda supports 1000 concurrent requests for personal use), public FaaS providers should be able to handle the majority of workloads with ease. Therefore, we mainly focus on selecting the best FaaS provider in terms of performance and cost for general cases. Some work

extended FaaS to on-premises environments and designed scheduling algorithms in the edge-cloud continuum [6, 65]. This requires the deployment of open-source FaaS on the edge server, and our focus is on the utilization of public FaaS offerings. Jindal et al. leverage different FaaS configurations (memory allocation, region, and provider) to schedule for better performance [42]. Real-world workloads are typically supported by both FaaS and BaaS offerings. We also consider the performance of FaaS when it is coupled with BaaS offerings. A few studies added features to existing open-source tools or frameworks for serverless multi-cloud environments [9, 47]. We are also inspired by Sky Computing, which envisioned and proposed a transparent cloud on top of cloud providers to provide best-of-breed service in terms of performance and cost for each workload [19, 68]. In serverless computing, we provide an analysis system to evaluate the performance and cost, helping in selecting the best FaaS provider for each workload. The multi-cloud library makes multi-cloud architecture deployment easier, but it falls short of the transparent cloud. To achieve the vision of a transparent cloud, we believe that cross-platform standards should be available first. Another difference is that sky computing is acting as a middleman, representing users in selecting the best-of-breed cloud services. Our software is designed for users; they use it to make their own choice.

Benchmarking and resource allocation. To better evaluate real-world deployment, our benchmarks mainly reference application-oriented benchmarks [23, 34, 45, 71]. Public FaaS providers handle resource management for users but allow users to configure allocated memory for workloads. The configured memory size does not only determine the hardware resources allocated to the function but also relates to its network, I/O, etc [70]. Thus, choosing a proper memory value for a workload has been extensively investigated. Spillner used memory tracing to prevent the over-allocation of memory resources [66]. A number of studies searched for the optimal memory size for a workload from a performance and/or cost perspective [2, 26, 55, 74]. Our work is in the context of multi-cloud, and we focus on cross-platform performance and cost comparison. Thus, we chose the same memory size for each benchmarking in order to quantitatively compare FaaS platforms. After we select the best FaaS providers

for this workload, we may be able to use the above optimization techniques as a complement to find a more optimal memory size in a very small range. Moreover, we also consider the performance when a function is coupled with a BaaS offering and support cross-platform fine-grained cost calculation: the duration fee, network fee, etc. In serverless computing, resource allocation aims to find a trade-off between performance and cost. Bilal et al. went back and argued about the potential of decoupling memory and computing resource allocation [13].

Chapter 7

Conclusion

In this thesis, we discussed the potential and challenges of multi-cloud. To support and facilitate it, we developed a library to manage the FaaS and BaaS offerings of different providers while satisfying the needs of serverless multi-cloud. We implemented an analysis tool that enabled performance measurement and *fine-grained* cost calculation for FaaS providers. We also designed benchmarks and evaluated FaaS providers with the abovementioned tools. We presented how to select the best-of-breed FaaS provider for a given workload.

Appendix A

Acronyms

AWS Amazon Web Services

GCF Cloud Functions

AFC Function Compute

EC2 Elastic Compute Cloud

FaaS Function-as-a-Service

BaaS Backend-as-a-Service

HTTP Hypertext Transport Protocol

API Application Programming Interface

SDK Software Development Kit

IaaS Infrastructure as a Service

PaaS Platform as a Service

SaaS Software as a Service

RPC Remote Procedure Call

JSON JavaScript Object Notation

CSV Comma-separated Values

CPU Central Processing Unit

GPU Graphics Processing Unit

TPU Tensor Processing Unit

vCPU Virtual Central Processing Unit

Bibliography

[1] Brian Adler. *Cloud Computing Trends: Flexera 2022 State of the Cloud Report*. https://www.flexera.com/blog/cloud/cloud-computing-trends-2022-state-of-the-cloud-report/. Accessed: 2022-11-08.

[2] Nabeel Akhtar et al. "COSE: configuring serverless functions using statistical learning". In: *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE. 2020, pp. 129–138.

[3] *Amazon S3*. https://aws.amazon.com/lambda/pricing/. Accessed: 2022-11-08.

[4] *Apache jclouds*. https://jclouds.apache.org/. Accessed: 2022-11-08.

[5] *Apache Libcloud*. https://libcloud.apache.org/. Accessed: 2022-11-08.

[6] Austin Aske and Xinghui Zhao. "Supporting multi-provider serverless computing on the edge". In: *Proceedings of the 47th International Conference on Parallel Processing Companion*. 2018, pp. 1–6.

[7] *AWS Lambda*. https://aws.amazon.com/lambda/. Accessed: 2022-11-08.

[8] *AWS Lambda Pricing*. https://aws.amazon.com/lambda/pricing/. Accessed: 2022-11-08.

[9] Azure. *Multicloud solutions with the Serverless Framework*. https://docs.microsoft.com/en-us/azure/architecture/example-scenario/serverless/serverless-multicloud. Accessed: 2022-11-08.

[10] *Azure Functions*. https://azure.microsoft.com/en-us/products/functions/. Accessed: 2022-11-08.

[11] Ataollah Fatahi Baarzi et al. "On merits and viability of multi-cloud serverless". In: *Proceedings of the ACM Symposium on Cloud Computing*. 2021, pp. 600–608.

[12] David Bermbach, Erik Wittern, and Stefan Tai. *Cloud service benchmarking*. Springer, 2017.

[13] Muhammad Bilal et al. "With great freedom comes great opportunity: Rethinking resource allocation for serverless functions". In: *arXiv preprint arXiv:2105.14845* (2021).

[14] Nour El Houda Bouzerzour, Souad Ghazouani, and Yahya Slimani. "A survey on the service interoperability in cloud computing: Client-centric and provider-centric perspectives". In: *Software: Practice and Experience* 50.7 (2020), pp. 1025–1060.

[15] Joao Carreira et al. "A case for serverless machine learning". In: *Workshop on Systems for ML and Open Source Software at NeurIPS*. Vol. 2018. 2018.

[16] Paul Castro et al. "Hybrid Serverless Computing: Opportunities and Challenges". In: *arXiv preprint arXiv:2208.04213* (2022).

[17] Paul Castro et al. "Serverless programming (function as a service)". In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2017, pp. 2658–2659.

[18] Paul Castro et al. "The rise of serverless computing". In: *Communications of the ACM* 62.12 (2019), pp. 44–54.

[19] Sarah Chasins et al. "The Sky Above The Clouds". In: *arXiv preprint arXiv:2205.07147* (2022).

[20] *Cloud Functions*. https://cloud.google.com/functions. Accessed: 2022-11-08.

[21] *Cloud Functions pricing*. https://cloud.google.com/functions/pricing. Accessed: 2022-11-08.

[22] *Cloudflare R2*. https://www.cloudflare.com/products/r2/. Accessed: 2022-11-08.

[23] Marcin Copik et al. "Sebs: A serverless benchmark suite for function-as-a-service computing". In: *Proceedings of the 22nd International Middleware Conference*. 2021, pp. 64–78.

[24] Jeffrey Dean and Luiz André Barroso. "The Tail at Scale". In: *Commun. ACM* 56.2 (Feb. 2013). Place: New York, NY, USA Publisher: Association for Computing Machinery, pp. 74–80. ISSN: 0001-0782. DOI: 10.1145/2408776.2408794. URL: https://doi.org/10.1145/2408776.2408794.

[25] AWS Lambda Documentation. *Configuring Lambda function options*. https://docs.aws.amazon.com/lambda/latest/dg/configuration-function-common.html. Accessed: 2022-11-08.

[26] Simon Eismann et al. "Sizeless: Predicting the Optimal Size of Serverless Functions". In: *Proceedings of the 22nd International Middleware Conference*. Middleware '21. Québec city, Canada: Association for Computing Machinery, 2021, pp. 248–259. ISBN: 9781450385343. DOI: 10.1145/3464298.3493398. URL: https://doi.org/10.1145/3464298.3493398.

[27] Simon Eismann et al. "The State of Serverless Applications: Collection, Characterization, and Community Consensus". In: *IEEE Transactions on Software Engineering* 48.10 (2022), pp. 4152–4166. DOI: 10.1109/TSE.2021.3113940.

[28] FACTION. *What is Data Gravity? How it Can Influence Your Cloud Strategy*. https://www.factioninc.com/blog/data-gravity-as-the-center-of-your-multi-cloud-universe/. Accessed: 2022-11-08.

[29] *fog*. http://fog.io/. Accessed: 2022-11-08.

[30] Sadjad Fouladi et al. "Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads". In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, Mar. 2017, pp. 363–376. ISBN: 978-1-931971-37-9. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/fouladi.

[31] *Function Compute*. https://www.alibabacloud.com/product/function-compute. Accessed: 2022-11-08.

[32] *Function Compute Pricing*. https://www.alibabacloud.com/product/function-compute/pricing. Accessed: 2022-11-08.

[33] Fotis Gonidis et al. "Cloud application portability: an initial view". In: *Proceedings of the 6th Balkan Conference in Informatics*. 2013, pp. 275–282.

[34] Martin Grambow et al. "Befaas: An application-centric benchmarking framework for faas platforms". In: *2021 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE. 2021, pp. 1–8.

[35] Arpan Gujarati et al. "Serving DNNs like Clockwork: Performance Predictability from the Bottom Up". In: *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 443–462. ISBN: 978-1-939133-19-9. URL: https://www.usenix.org/conference/osdi20/presentation/gujarati.

[36] Robin Hartauer, Johannes Manner, and Guido Wirtz. "Cloud Function Lifecycle Considerations for Portability in Function as a Service." In: *CLOSER*. 2022, pp. 133–140.

[37] Kim Hazelwood et al. "Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective". In: vol. 2018-February. 2018. DOI: 10.1109/HPCA.2018.00059.

[38] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[39] Joseph M. Hellerstein et al. "Serverless Computing: One Step Forward, Two Steps Back". In: *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. www.cidrdb.org, 2019. URL: http://cidrdb.org/cidr2019/papers/p119-hellerstein-cidr19.pdf.

[40] Pablo Iorio. *Multicloud trends*. https://pablo-iorio.medium.com/multicloud-patterns-and-trends-a66ea092185e. Accessed: 2022-11-08.

[41] Jiawei Jiang et al. “Towards demystifying serverless machine learning training”. In: *Proceedings of the 2021 International Conference on Management of Data*. 2021, pp. 857–871.

[42] Anshul Jindal et al. “Courier: delivering serverless functions within heterogeneous FaaS deployments”. In: *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing* (2021).

[43] Eric Jonas et al. “Cloud programming simplified: A berkeley view on serverless computing”. In: *arXiv preprint arXiv:1902.03383* (2019).

[44] Eric Jonas et al. “Occupy the cloud: Distributed computing for the 99%”. In: *Proceedings of the 2017 symposium on cloud computing*. 2017, pp. 445–451.

[45] Jeongchul Kim and Kyungyong Lee. “FunctionBench: A Suite of Workloads for Serverless Cloud Function Service”. In: *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. 2019, pp. 502–504. DOI: 10.1109/CLOUD.2019.00091.

[46] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images”. In: (2009).

[47] Oliviu Matei et al. “Functionizer-A Cloud Agnostic Platform for Serverless Computing”. In: *International Conference on Advanced Information Networking and Applications*. Springer. 2021, pp. 541–550.

[48] *ntplib*. https://pypi.org/project/ntplib/. Accessed: 2022-11-08.

[49] Justice Opara-Martins, Reza Sahandi, and Feng Tian. “Critical review of vendor lock-in and its impact on adoption of cloud computing”. In: *International Conference on Information Society (i-Society 2014)*. IEEE. 2014, pp. 92–97.

[50] Dana Petcu. “Multi-cloud: expectations and current approaches”. In: *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*. 2013, pp. 1–6.

[51] Dana Petcu. "Portability and interoperability between clouds: challenges and case study". In: *European conference on a service-based internet*. Springer. 2011, pp. 62–74.

[52] *Pillow*. https://pillow.readthedocs.io/en/stable/. Accessed: 2022-11-08.

[53] Francisco Romero et al. "INFaaS: Automated Model-less Inference Serving". In: *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, July 2021, pp. 397–411. ISBN: 978-1-939133-23-6. URL: https://www.usenix.org/conference/atc21/presentation/romero.

[54] Barbara Russo et al. "Cloud computing and the new EU general data protection regulation". In: *IEEE Cloud Computing* 5.6 (2018), pp. 58–68.

[55] Gor Safaryan et al. "SLAM: SLO-Aware Memory Optimization for Serverless Applications". In: *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*. 2022, pp. 30–39. DOI: 10.1109/CLOUD55607.2022.00019.

[56] Josep Sampe et al. "Toward Multicloud Access Transparency in Serverless Computing". In: *IEEE Software* 38.1 (2021), pp. 68–74. DOI: 10.1109/MS.2020.3029994.

[57] Marc Sánchez-Artigas and Pablo Gimeno Sarroca. "Experience Paper: Towards enhancing cost efficiency in serverless machine learning training". In: *Proceedings of the 22nd International Middleware Conference*. 2021, pp. 210–222.

[58] Klaus Satzke et al. "Efficient gpu sharing for serverless workflows". In: *Proceedings of the 1st Workshop on High Performance Serverless Computing*. 2020, pp. 17–24.

[59] Joel Scheuner et al. "CrossFit: Fine-grained Benchmarking of Serverless Application Performance across Cloud Providers". In: *Proceedings of the 15th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2022)*. 2022.

[60] Johann Schleier-Smith et al. "What serverless computing is and should become: the next phase of cloud computing". In: *Communications of the ACM* 64 (May 2021), pp. 76–84. DOI: 10.1145/3406011.

[61] *scikit-learn*. https://scikit-learn.org/stable/. Accessed: 2022-11-08.

[62] SentinelOne. *Serverless Logging: How It Works When You're in the Cloud*. https://www.sentinelone.com/blog/serverless-logging-cloud/. Accessed: 2022-11-08.

[63] *Serverless Framework*. https://www.serverless.com/. Accessed: 2022-11-08.

[64] Mohammad Shahrad et al. "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider". In: *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. 2020, pp. 205–218.

[65] Christopher Peter Smith et al. "FaDO: FaaS Functions and Data Orchestrator for Multiple Serverless Edge-Cloud Clusters". In: *2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC)*. 2022, pp. 17–25. DOI: 10.1109/ICFEC54809.2022.00010.

[66] Josef Spillner. "Resource Management for Cloud Functions with Memory Tracing, Profiling and Autotuning". In: *Proceedings of the 2020 Sixth International Workshop on Serverless Computing*. WoSC'20. Delft, Netherlands: Association for Computing Machinery, 2021, pp. 13–18. ISBN: 9781450382045. DOI: 10.1145/3429880.3430094. URL: https://doi.org/10.1145/3429880.3430094.

[67] Josef Spillner, Cristian Mateos, and David A Monge. "Faaster, better, cheaper: The prospect of serverless scientific computing and hpc". In: *Latin American High Performance Computing Conference*. Springer. 2017, pp. 154–168.

[68] Ion Stoica and Scott Shenker. "From cloud computing to sky computing". In: *Proceedings of the Workshop on Hot Topics in Operating Systems*. 2021, pp. 26–32.

[69] Davide Taibi, Josef Spillner, and Konrad Wawruch. "Serverless Computing-Where Are We Now, and Where Are We Heading?" In: *IEEE Software* 38.1 (2021), pp. 25–31. DOI: 10.1109/MS.2020.3028708.

[70] Liang Wang et al. “Peeking Behind the Curtains of Serverless Platforms”. In: *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, July 2018, pp. 133–146. ISBN: ISBN 978-1-939133-01-4. URL: https://www.usenix.org/conference/atc18/presentation/wang-liang.

[71] Tianyi Yu et al. “Characterizing Serverless Platforms with ServerlessBench”. In: *Proceedings of the ACM Symposium on Cloud Computing*. SoCC ’20. Association for Computing Machinery, 2020. DOI: 10.1145/3419111.3421280. URL: https://doi.org/10.1145/3419111.3421280.

[72] Vladimir Yussupov et al. “Facing the unplanned migration of serverless applications: A study on portability problems, solutions, and dead ends”. In: *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*. 2019, pp. 273–283.

[73] Chengliang Zhang et al. “MArk: Exploiting Cloud Services for Cost-Effective, SLO-Aware Machine Learning Inference Serving”. In: *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. Renton, WA: USENIX Association, July 2019, pp. 1049–1062. ISBN: 978-1-939133-03-8. URL: https://www.usenix.org/conference/atc19/presentation/zhang-chengliang.

[74] Miao Zhang et al. “Video processing with serverless computing: A measurement study”. In: *Proceedings of the 29th ACM workshop on network and operating systems support for digital audio and video*. 2019, pp. 61–66.

[75] Zhizhong Zhang, Chuan Wu, and David WL Cheung. “A survey on cloud interoperability: taxonomies, standards, and practice”. In: *ACM SIGMETRICS Performance Evaluation Review* 40.4 (2013), pp. 13–22.

[76] Haidong Zhao et al. “Supporting Multi-Cloud in Serverless Computing”. In: *arXiv preprint arXiv:2209.09367* (2022).