Demystifying the Cost of Serverless Computing: Towards a Win-Win Deal

Fangming Liu*, *Senior Member, IEEE*, Yipei Niu

Abstract—Serverless is an emerging computing paradigm that greatly simplifies the development, deployment, and maintenance of cloud applications. However, due to potential cost issues brought by the widely adopted pricing, it is difficult to answer how to use and operate serverless computing services from the perspectives of users and providers. To demystify the cost of serverless computing, we present one of the first studies that develops an analytical model for serverless cost from the perspectives of users and providers, by comparing it to Infrastructure-as-a-Service. Based on the model, driven by real-world traces, extensive simulation results verify the following cost issues: 1) For the users, serverless is not always cost-saving, even possibly leading to expense explosion; 2) The serverless providers are in urgent need of widening use scenarios to improve resource utilization and raise revenue; 3) The prevailing pricing fails to neither reduce the risk of expense explosion nor meet the need of attracting more workloads. To remove the cost barrier, we propose *future function*, auction-based pricing for serverless, to offer discounts to the users as well as boost profit for the providers. Experimental results show the duration price of functions can be reduced by 57.5% on average for 13.5% of users yet without harming the revenue of providers.

Index Terms—Serverless, FaaS, Cost Model, Pricing, Auction.

1 Introduction

Serverless is a new cloud computing paradigm that significantly simplifies the development, deployment, and maintenance of applications. Users submit code to serverless platforms and the code is executed as functions when they are triggered. Different from Infrastructure-as-a-Service (IaaS), serverless users do not need to maintain any resource, e.g., virtual machines (VMs), which greatly reduces operational cost.

Furthermore, with serverless, users are charged based on the number of requests for functions and the execution duration of functions. On the one hand, benefiting from the on-demand charging, the users only pay for how many resources they use, which is intuitively cost-saving. However, on the other hand, with the pricing scheme, the users are at risk of unexpectedly growing cost for three reasons: 1) The expense of functions linearly increases with requests and duration; 2) The duration price of functions is about $3.5\times$–$5.6\times$ higher than that of VMs [1]; 3) The duration price of functions is fixed and no discount is offered.

Such a paradox naturally raises two key questions from the perspective of users: *can serverless always be cost-saving and when serverless is cost-saving*? Concerning the cost issues from users, other questions arise from the perspective of providers as well: *can serverless always be profitable and how to profit more from serverless*?

To answer the questions above, in this paper, we develop an analytical cost model for serverless, so as to reveal how to use and operate serverless computing services from the perspectives of users and providers compared to IaaS. Through extensive trace-driven simulations, our insights firstly derive exact conditions when serverless is more economical than IaaS. Specifically, serverless is cost-saving when the workload demand is generally low, no matter how the request arrival patterns are. Particularly, if the workload demand fluctuates wildly, serverless is more cost-saving. Then our cost model reveals serverless is always profitable, since the duration price of functions is higher than that of VMs. Additionally, serverless inherently allows the providers to further boost profit by over-committing more resources (selling more resources than the physical infrastructure actually has).

However, our simulation results verify multiple cost issues of users and providers. For serverless users, if user requests continuously arrive constantly, the expense of

- *This work was supported in part by National Key Research & Development (R&D) Plan under grant 2022YFB4501703 and in part by The Major Key Project of PCL (PCL2022A05). (Corresponding author: Fangming Liu)*
- *F. Liu is with Peng Cheng Laboratory, and Huazhong University of Science and Technology, China. E-mail: fangminghk@gmail.com.*
- *Y. Niu is with the National Engineering Research Center for Big Data Technology and System, the Services Computing Technology and System Lab, Cluster and Grid Computing Lab in the School of Computer Science and Technology, Huazhong University of Science and Technology, China. E-mail: niuypei@hust.edu.cn.*

1. For AWS EC2 `c6g.8xlarge` instances, the on-demand pricing plan is 0.472×10^{-5}\$/GB·second and the reserved price is 0.297×10^{-5}\$/GB·second. The duration price of AWS Lambda is 1.667×10^{-5}\$/GB·second (x86 type). Hence, the duration price of functions is about $3.5\times$–$5.6\times$ higher than that of VMs ($3.5 = \frac{1.667\times10^{-5}}{0.472\times10^{-5}}$, $5.6 = \frac{1.667\times10^{-5}}{0.297\times10^{-5}}$). The reason why the prices of functions are higher is that FaaS platforms manage the resources of functions for users by introducing extra services and components.

serverless is lower at first but finally surpasses that of IaaS, even leading to expense explosion. What's worse, the prevailing pricing strategy charges the users a fixed price, leaving little opportunity for users to bargain. The users yearn for discounts, especially when the workload demand is predictably high. Concerning the potential cost issue, users may hesitate to adopt serverless. As a result, from the perspective of providers, although serverless is always profitable for the high duration price and resource overcommitment, the cost issues would hinder the growth of users and workload demand, finally leading to a fall in revenue. Worse, the widely adopted pricing of serverless fails to provide any discount as an incentive to attract users and increase workload demand.

These undesirable drawbacks restrict serverless to only be used for limited scenarios and possibly cause a decline in serverless. However, technically serverless can be a general programming platform and should be able to support any type of workload. To realize this, we should remove the cost barrier by changing the pricing.

To provide a bilateral pricing strategy and enable the provider and users to agree on a satisfactory price, we propose *future function* based on the auction theory, where functions won by bidders will be delivered in the future instead of imminently. Specifically, the provider periodically releases a certain amount of discounted functions, and all the users are allowed to bid for them at lower prices (bidding price). At the beginning of the time slot, the serverless provider selects specific bidders as winners and determines their payments (which are lower than the bidding price). The winners are charged the payment during the *future* period of time while others who fail to win the discounted functions have to pay based on the original price. We adopt a sub-optimal yet efficient algorithm, so as to ensure truthfulness and social warfare maximum. Driven by the real-world traces, simulation results show that the duration price can be reduced by 57.5% on average yet the revenue of providers barely shrinks.

The contributions of this paper are summarized as follows.

- We develop an analytical cost model for serverless. By systematically examining a series of critical factors affecting the cost for both users and providers, we provide a qualitative understanding of serverless cost.
- Through extensive trace-driven simulations, we analyze when and why serverless is more economical or profitable than IaaS. Furthermore, we reveal that serverless is not always cost-saving for the users and the providers may face a loss of profit due to today's static pricing strategy.
- We design the *future function*, auction-based pricing for serverless, to alleviate expense explosion for users and raise revenue for providers. The pricing plan attracts extra workloads to shape user demand, ensuring as many users can receive discounts as possible and the revenue of the provider does not shrink.

The rest of this paper is organized as follows. Section 2 characterizes the performance and user demand of serverless. Section 3 examines the cost and identifies the expense explosion of serverless users. Section 4 further analyzes the cost of providers. Section 5 reveals the cost barrier between users and providers. Section 7 proposes the future function and develops the auction-based pricing for serverless. Section 8 discusses related work. Section 9 concludes the whole paper.

2 Characterizing Serverless

In this section, we develop a basic model for serverless to characterize the performance and cost of serverless. A serverless platform encompasses two parts: Function-as-a-Service (FaaS) and Backend-as-a-Service (BaaS). FaaS provisions computing services to cloud users based on functions. Since functions are stateless, BaaS is introduced to maintain mutable states for functions, for example, Pywren [1] uses S3 to store intermediate data during shuffling stages.

2.1 Pricing Scheme of Serverless

The pricing scheme of serverless differs across cloud providers. Hence, we select AWS Lambda [2], Microsoft Azure Function [3], GCP Function [4], and Alibaba Cloud Compute Function [5] to discuss it.

Pricing of FaaS. We compare the pricing plan of FaaS providers in Table 1. The pricing plan of existing FaaS platforms contains two tiers: free tier and charge tier. After the free quota of the free tier is used up, the provider will charge users for future function execution based on the pricing of the charge tier. The charge tier contains two parts: request and duration. The request part is the price of each request that triggers a function. The duration part is the price of function execution time. As shown in Table 1, the unit of duration price is defined as memory capacity multiplexing execution time, i.e., GB $\cdot$ second. Apart from memory, some providers charge users for CPU usage, including, Aliyun and GCP[2].

Pricing of BaaS. The additional expense of interacting with BaaS typically consists of two parts: data transfer and service requests. Service requests are charged based on the number of requests. IaaS and FaaS share the same pricing strategy.

2.2 Performance of Serverless

The process of serving requests on serverless platforms basically contains three phases: initialization, computing, and backend services. As a result, the total execution latency of serverless contains three parts: initialization latency of functions, service latency of BaaS, and computing latency of FaaS.

Initialization latency of functions. Functions are mostly short-lived, the initialization latency, especially cold start latency, hence plays a critical role in the overall performance. In contrast, concerning that VMs are running for a long period of time, although the startup time of VMs is higher, the startup latency is relatively low.

Service latency of BaaS. Since functions on serverless are stateless, running functions need to interact with native

2. Here we only discuss the fundamental pay-as-you-go schemes of Serverless function.

Table 1: The comparison of pricing policies among FaaS providers.

Provider	Free tier			Charge tier			Configuration		
	Request Price‡	Duration Price		Request Price‡	Duration Price		Memory		vCPU:Mem†
		Mem (GB·s)	CPU (vCPU·s)		Mem (GB·s)	CPU (vCPU·s)	Min	Max	
AWS	1 M	400000	0.00	0.2 per M	0.0000166667	0.00	128MB	10GB	1:1.728
Azure	1 M	400000	0.00	0.2 per M	0.000016	0.00	128MB	1536MB	Unknown
GCP	2 M	400000	8333.33	0.4 per M	0.0000025	0.0000042	128MB	32GB	1:1.5–1:4
Alibaba	1 M	1000000	500000	0.2 per M	0.000002138	0.00002138	128MB	32GB	1:1–1:4

† The number of vCPU to the capacity of memory. (The number of vCPU : GB)
‡ "M" stands for million.

cloud services to maintain states, such as storage service, database, message queue, etc. Compared to serverless, VMs can maintain states in persistent storage locally and transmit states via networking directly. Therefore, the latency of managing states on IaaS can be omitted.

Computing latency of FaaS. Functions are more fine-grained than VMs. As shown in Table 1, the memory capacity of a function is less than 10GB. Only AWS and Azure release the vCPU configuration of a function. Azure Compute Unit (ACU) is not an authoritative metric, because ACU is proposed and defined by Azure. ACU provides a standard for comparing CPU performance across Azure products. One ACU is currently standardized on Azure Standard_A1 as 100[3]. The memory-to-vCPU ratio (MB per vCPU) of AWS Lambda is 1769, which roughly equals that of compute-optimized EC2 instances, i.e., 2048. Resources of a FaaS platform can be classified into three levels: platform, framework, and function. Platform-level resources are consumed by services that support the container orchestration platform, such as Kubernetes, including etcd, kube-proxy, and so on. Framework-level resources are occupied by components that implement FaaS frameworks, such as OpenFaaS and OpenWhisk, including queueing components, front-end components, etc. Function-level resources are the remaining parts allocated to functions for execution. Since FaaS users do not own any cloud resources, as compared to IaaS, the extra resource cost is on the framework level, not on the platform level. As a result, the memory-to-vCPU ratio of EC2 instances is 15% higher than that of AWS Lambda. We define a certain amount of vCPU and memory as a computing unit (e.g., 1vCPU and 1769MB memory). Hence, a function is one computing unit and a VM has multiple identical units.

For two identical computing units, if the resource utilization is fixed, the performance can be enforced to be the same. As a result, the computing latency of a function and a VM are the same, but the throughput of a VM is multiple times higher than that of a function.

2.3 User Demand of Serverless

To build the mathematical model, we first consider a discrete-time model $t \in \{1, 2, ..., t, ...\}$. a_t is the request arrival rate during the t^{th} time slot. The number of vCPU and the capacity of memory in one computing unit are R^{vCPU} and R^{Mem}, respectively. The resource demand of request i on each computing unit is denoted as a vector $\vec{r}_i$. Here we take vCPU and memory into consideration, i.e., $\vec{r}_i = (r_i^{vCPU}, r_i^{Mem})$.

3. Azure compute unit (ACU), https://learn.microsoft.com/en-us/azure/virtual-machines/acu

Table 2: The notations of cost model.

Notation	Description
a_t	The request arrival rate during the t^{th} time slot.
$K_S(t)$	The expense of serverless during the t^{th} time slot.
P_F	The duration price of FaaS.
l_i^S	The total execution time of request i.
P_B	The sum of request prices of FaaS and BaaS.
R^{vCPU}	The number of vCPU in one computing unit.
R^{Mem}	The capacity of memory in one computing unit.
V_F^{vCPU}	The vCPU utilization of FaaS.
V_F^{Mem}	The memory utilization of FaaS.
$\vec{r}_i$	The resource demand of request i .
N_R	The scale of reserved VMs.
$N_O(t)$	The scale of on-demand VMs.
$K_I(t)$	The expense of IaaS during the t^{th} time slot.
P_R	The price of reserved VMs.
P_O	The price on-demand VMs.
V_I^{vCPU}	The vCPU utilization of all the rented VMs
V_I^{Mem}	The memory utilization of all the rented VMs

3 Understanding Cost of Users

In this section, we develop a cost model for serverless from the perspective of users. To analyze when, why, and how much serverless is more economical, we introduce the user cost model of IaaS as a baseline.

To compare the cost of serverless and IaaS, we consider the following setups: 1) Only one user is served on the IaaS and serverless platforms, respectively; 2) Resource demand of requests on the IaaS and serverless are the same; 3) The resources allocated to the user are exclusively supported by the same scale of underlying physical infrastructure, implying that the scale is fixed when comparing FaaS with IaaS.

We then analyze the cost of serverless and IaaS from two aspects: expense and resource utilization. The expense is referred to as the fee for executing functions or renting VMs, while the resource utilization is defined as how many resources paid by the user are utilized, indicating the cost-effectiveness of FaaS or IaaS.

3.1 Cost of Using Serverless

Expense of serverless. As shown in Figure 1, the scale of functions can automatically adapt to the scale of user demand. The expense of serverless during the t^{th} time slot

$K_S(t)$ is the sum of the FaaS and BaaS expenses, which can be denoted as follows.

$$K_S(t) = P_B a_t + P_F R^{Mem} \sum_{i=1}^{a_t} l_i^S,$$

where P_F is the duration price of FaaS, l_i^S is the total execution time of request i (including initialization, computing, and BaaS), and P_B is the sum of request prices of FaaS and BaaS.

Remark. The expense of serverless incurs only when the functions are active. As soon as the functions are completed, the serverless provider does not charge the user immediately. *The serverless users essentially pay for the usage of functions*.

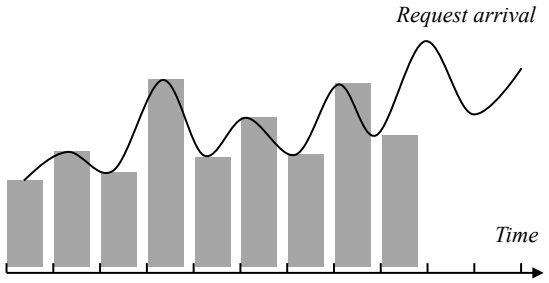


Figure 1: Expense of serverless is calculated based on the number of requests for functions and the execution duration of functions.

Resource utilization of FaaS. The vCPU utilization V_F^{vCPU} and memory utilization V_F^{Mem} of FaaS are calculated by dividing the resource demand by the resource capacity allocated to the user, which are defined as follows, respectively.

$$V_F^{vCPU}(t) = \frac{\sum_{i=1}^{a_t} l_i^S r_i^{vCPU}}{R^{vCPU} \sum_{i=1}^{a_t} l_i^S}, V_F^{Mem}(t) = \frac{\sum_{i=1}^{a_t} l_i^S r_i^{Mem}}{R^{Mem} \sum_{i=1}^{a_t} l_i^S}.$$

Remark. The resource utilization of FaaS, e.g., CPU utilization, is denoted as $(\sum_{i=1}^{a_t} l_i^S r_i^{vCPU})/(R^{vCPU} \sum_{i=1}^{a_t} l_i^S) = (\sum_{i=1}^{a_t} \frac{r_i^{vCPU}}{R^{vCPU}} \cdot l_i^S)/(\sum_{i=1}^{a_t} l_i^S)$, which is the weighted arithmetic mean of the resource utilization of functions ($\frac{r_i^{vCPU}}{R^{vCPU}}$). The weights are the processing time of functions (l_i^S). *This means there is no waste of resources caused by idleness*.

3.2 Cost of Renting IaaS

Typically, the VMs that an IaaS user rents are divided into base and on-demand groups. VMs in the base group are reserved instances for serving ordinary workload demand and ensuring high availability, whose scale remains unchanged. These VMs are rented for a long-term duration with all-upfront (e.g., 1-year all-upfront [6]) to save cost [7]. Meanwhile, to deal with unexpected bursty workload demand, on-demand VMs are employed to improve scalability. With the bursty workload demand dying away, the on-demand VMs are shut down to save cost.

Expense of IaaS. The expense of IaaS is calculated based on the scale of VMs and the duration of renting VMs. As shown in Figure 2, we assume that VMs the user rents are homogeneous, where the scale of reserved and on-demand

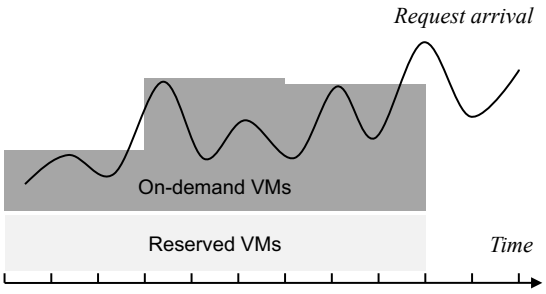


Figure 2: Typical solution to deploying applications to IaaS. The resource utilization is updated at regular time intervals. During the time interval, the scale of on-demand VMs remains unchanged.

VMs are denoted as N_R and $N_O(t)$, respectively. Hence, the expense of IaaS during the t^{th} time slot $K_I(t)$ is calculated by summing up the expense of renting reserved and on-demand VMs, which is calculated as follows.

$$K_I(t) = P_R N_R + P_O N_O(t),$$

where P_R and P_O are the prices of the reserved and on-demand VMs.

Resource utilization of IaaS. The vCPU utilization V_I^{vCPU} and memory utilization V_I^{Mem} are derived by dividing the resource demand by the resource capacity rented by the user, which are denoted as follows.

$$V_I^{vCPU}(t) = \frac{\sum_{i=1}^{a_t} l_i^I r_i^{vCPU}}{R^{vCPU} \Theta N(t)}, V_I^{Mem}(t) = \frac{\sum_{i=1}^{a_t} l_i^I r_i^{Mem}}{R^{Mem} \Theta N(t)},$$

where l_i^I is the computing latency of request i running in a computing unit of IaaS (the initialization is omitted and no BaaS latency). $N(t)$ is the total number of VMs during the t^{th} time slot and it equals $N_R + N_O(t)$. Θ is the total number of computing units a VM has.

Remark. The IaaS expense $K_I(t)$ monotonically increases with the scale of the on-demand VMs $N_O(t)$, where $N_O(t)$ is determined by the scaling strategy of the IaaS user. Typically IaaS users leverage auto-scaling services to adjust the scale of VMs: if the overall utilization of one particular resource is above/below a threshold, the scale of VMs grows/falls. However, the resource utilization cannot be timely updated (AWS CloudWatch updates the load statistics every 5 minutes at no charge and every minute for an additional charge). As a result, when the bursty workload demand dies out, the VMs of the user may be over-provisioned. Worse, the scale of reserved VMs keeps unchanged, incurring extra expense regardless of the workload demand. Above all, *the IaaS users basically pay for renting VMs, no matter whether the VMs are fully utilized (i.e., busy or idle)*.

3.3 Is Serverless More Economical than IaaS?

Although the on-demand charging of serverless prevents users from paying for idle resources, there exist potential cost issues. Based on the formulation of expense in Sec. 3.1, the expense of serverless monotonically increases with the request arrival rate. When the applications are running long

enough, the expense of serverless will grow continuously, possibly resulting in an expense explosion. Furthermore, the duration price of functions is fixed and $3.5\times$–$5.6\times$ higher than VMs, which makes it impossible to alleviate the expense explosion. Concerning the lack of discounts, users would hesitate to adopt serverless.

4 Revenue of Serverless Providers

In this section, to identify the revenue crisis of serverless providers, we develop a cost model to analyze why serverless is more profitable than IaaS and how to increase profit.

4.1 Serverless is Resource-conserving

Functions are more fine-grained and light-weighted than VMs, there is a huge benefit from selling short-lived slices of physical machines that might otherwise go unused because of the difficulties of bin-packing VMs and the relative slowness of VM scaling up/down.

Chances of over-committing resource. More importantly, as analyzed in Sec. 3, serverless sells function execution, while IaaS rents resources. Hence, serverless inherently allows higher resource over-commitment.

As shown in Figure 3, since the IaaS providers have to ensure that every user's resources are available at any time, two computing units (as well as the underlying physical resources) are allocated to user i and j, respectively. Although the computing units are idle, e.g., during $[t_i, t+1]$, resources allocated to user i cannot be shared with user j. Conversely, FaaS users only submit code to the provider and do not own any resources. Functions become active only when they are triggered, and shut down when they are complete. Hence, users i and j can share one computing unit on FaaS. There is no waste of resources when the demand of users is low, making FaaS providers over-commit more resources.

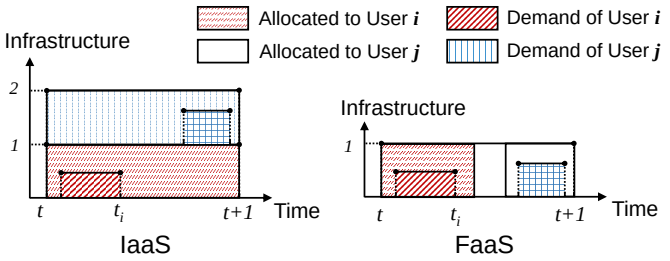


Figure 3: For IaaS, VMs are rented to users who hence temporarily occupy the virtual and underlying physical resources. The physical resources cannot be shared among IaaS users even when they are idle, since it is necessary to ensure that the virtual resource is available at any time.

Profiting from over-commitment. Typically, providers can over-commit resources to improve infrastructure utilization and raise revenue. The over-commitment (OC) ratio of resource r is defined as the ratio of the allocated virtual resources to the total physical resources, which can be denoted as $OC_r = \frac{C'_r}{C_r}$. Although raising the OC ratio can help improve resource utilization and make more profit, the high OC ratio of the CPU inevitably incurs frequent context switches, resulting in performance degradation. Suppose that OC^{CPU} vCPU are bound to each CPU core. The overhead of processes gets scheduled (Φ) can be denoted as $\phi \cdot OC^{CPU}$, where ϕ is the CPU utilization of a process for getting scheduled on the CPU core.

On a cloud platform, the resources are multiplexed among users. This can take the form of time-multiplexing, where the users take turns (e.g., the processor resource), or space-multiplexing, where each user gets a part of the resource (e.g., memory). Since the capacity of memory is space-multiplexed, techniques of over-committing memory basically adopt a strategy of saving space, e.g., virtio-balloon, kernel-samepage-merging, and memory swap. These techniques not only introduce significant overhead but also harm the stability of the cloud system. However, compared with memory, the CPU inherently allows more over-commitment, since it is time-multiplexed in OS. Given that the memory-to-CPU ratio is fixed, memory is the critical resource that determines how much CPU is over-committed.

4.2 Cost of Providers

We take revenue and resource utilization into consideration to examine whether serverless can profit from over-commitment. The revenue is referred to be the total expense of cloud users and the resource utilization means the utilization of the underlying physical infrastructure. Similarly, IaaS is selected as the baseline as well.

When modeling the resource utilization, we only consider the computing services, i.e., FaaS and IaaS, excluding backend services. To compare the revenue and resource utilization between IaaS and FaaS, we make some necessary setups: 1) The scale of the underlying infrastructure that supports IaaS and FaaS is fixed; 2) The over-committed capacity provisioned by IaaS and FaaS can be totally sold to cloud users.

Revenue and resource utilization. The revenue of FaaS or IaaS is calculated by summing up the expense of all the users. The total number of computing units allocated to the users during the t^{th} time slot is denoted as g_t. For each computing unit g, the vCPU and memory utilization required by requests during the t^{th} time are denoted as $r_{g;t}^{vCPU}$ and $r_{g;t}^{Mem}$, respectively. The CPU and memory utilization ($U^{CPU}(t)$ and $U^{Mem}(t)$) of FaaS or IaaS can be calculated as $\frac{\sum_{g=1}^{g_t}(r_{g;t}^{vCPU} + \ g;t)l_{g;t}}{C^{CPU}}$ and $\frac{\sum_{g=1}^{g_t} r_{g;t}^{Mem} l_{g;t}}{C^{Mem}}$, respectively.

Remark. The total number of computing units g monotonically increases with the memory over-commitment ratio. Since FaaS platforms support higher OC ratios, the revenue and resource utilization are higher than IaaS. $l_{g;t}$ is the latency of the resource unit g being active during the t^{th} time slot. FaaS is more profitable than IaaS. Nevertheless, such a conclusion is based on an assumption, i.e., all of the over-committed capacity provisioned by FaaS can be totally sold to cloud users. Consequently, in practice, FaaS may not be as profitable as analyzed.

5 Insight on Cost of Serverless

Based on the cost model, we conduct extensive simulations. The detailed setup and results of simulations can be found in Section 6. By analyzing the simulation results, we provide multiple insights on the cost of serverless as follows.

Perspective of users. Generally, serverless is not cost-saving as claimed from the cost perspective. Whether serverless is economical depends on the arrival pattern and scale of requests. If the user demand fluctuates drastically (e.g., the bursty and drastic patterns), FaaS is more cost-saving. Particularly, the more wildly the user demand fluctuates, the better FaaS performs. However, if the scale of requests continuously increases, the expense of the bursty and drastic patterns will surpass that of IaaS like the regular case, even leading to expense explosion.

Perspective of providers. We produce multiple insights on how to exploit FaaS and profit more from the perspective of providers. Because of the high duration price of functions, the revenue of FaaS is much higher than that of IaaS. Since FaaS platforms inherently allow a higher memory OC ratio, providers can raise the OC ratio to improve resource utilization and increase profits.

The cost barrier of serverless. We can conclude that serverless is conditionally cost-saving due to the prevailing pricing. The potential expense explosion brought by the pricing greatly limits the use scenarios of serverless and hinders users from adopting serverless. As a result, although the serverless providers have a chance to make a big profit from over-committing more resources, the serverless providers are now facing the possibility of revenue loss due to the decrease in the number of users. The pricing of serverless, which fails to neither offer discounts for users nor fully exploit resources for providers, now become the cost barrier between the user and providers against a win-win deal.

6 Simulation of Serverless Cost

In this section, we demonstrate the trace-driven simulation results of the cost model to verify the cost issues of serverless.

6.1 General Setup

Real-world traces. The function invocation counts of the Azure Functions Trace 2019 [8] are selected to simulate the request arrival. The resource utilization of containers for online services in the Alibaba Cluster Trace 2018 [9] is selected as the resource demand of functions.

Spec of VMs and functions. The functions are assumed to be homogeneous as well, the memory is set as 1769MB configured with 1 vCPU. We assume that all the VMs are homogeneous and the spec of the VMs is set as 32 vCPU and 56608MB. The price of functions are based on AWS Lambda, while that of VMs are based on `c6g.8xlarge` EC2.

6.2 Evaluation of User's Cost

Trace. As shown in Figure 4, we select three typical traces, i.e., *regular* (the request arrival follows a daily recursive pattern), *drastic* (the scale of requests fluctuates wildly), and *bursty* (the scale of requests is generally low except multiple sudden spikes). The total number of requests in the three cases is the same.

Setup. The time intervals are assumed in units of minutes. The duration price of FaaS and IaaS is in units of $/GB·second. The users pay for on-demand EC2 instances by the minute[4] and the load statistics are updated every minute[5]. The time interval of adjusting the scale of on-demand VMs is set as 1 minute.

Baselines. The cost of IaaS depends on the scaling strategy of VMs. Here we set two scaling strategies as baselines: auto-scaling (auto) and 80%-reserved (p80) strategies. The auto-scaling strategy only prepares a minimal scale of reserved VMs for high availability and base workload demand while the scale of on-demand VMs automatically increases or decreases based on the workload demand. The 80%-reserved strategy prepares a fixed scale of reserved VMs whose capacity can satisfy the workload demand during 80% of the whole time slots. The exceeding workloads are handled by the auto-scaling on-demand VMs. The threshold of scaling is the average CPU utilization of the user demand.

Comparison of resource utilization. We plot Figure 5 to investigate how the user demand affects the memory utilization of FaaS and IaaS under 10% of full load.

In the bursty case, as plotted in Figure 5(a), the memory utilization of IaaS is mostly lower than 30% and fluctuates wildly due to the slowness of scaling up/down.

In the drastic case, as demonstrated in Figure 5(b), although the dynamics of the user demand is wild, the memory utilization of FaaS remains stable. However, due to the coarse-grained scaling strategy, the memory utilization of IaaS-p80 fluctuates drastically, indicating frequent over-provision and under-provision.

In the regular case, as illustrated in Figure 5(c), for the IaaS-auto case, the memory utilization becomes more stable than other cases, indicating that the VMs are in a state of busy and barely scale.

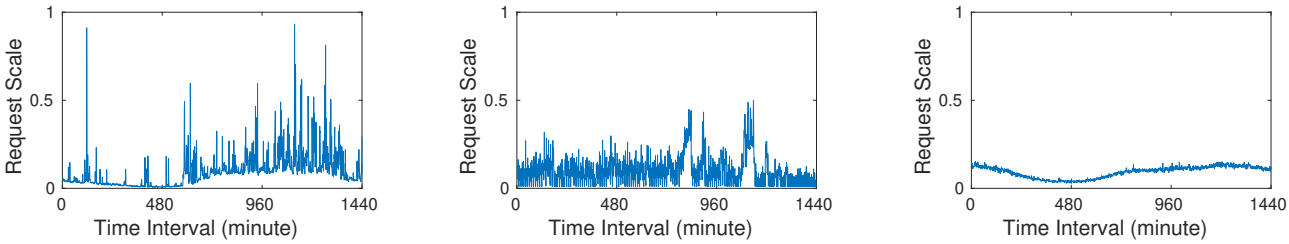
Above all, resources paid by the serverless user can be more efficiently utilized while the resource efficiency of IaaS depends on the scaling strategy and how the request arrival pattern is. The memory utilization of FaaS is stably high for all the cases, since the memory utilization of FaaS is the weighted arithmetic mean of the memory utilization of functions, implying that resource paid by the FaaS user remains busy, i.e., no waste of resources caused by idleness.

Comparison of expense. We plot Figure 6 to investigate how the user demand affects the expense of serverless under full load. When the scale of requests is low, as shown in Figure 6(a), 6(b), and 6(c), the expense of serverless is lower than IaaS in each case, especially in the bursty case.

Whether the expense of serverless is less than that of IaaS depends on the request arrival patterns and duration. As shown in Figure 6(a), 6(b), and 6(c), the expense of serverless is lower than that of IaaS at first. However, the expense of all the cases finally surpasses that of IaaS. It is because that the price of resource for functions are $3.5\times$–$5.6\times$ higher than that of VMs. With the number of requests increasing, the free quota is used up. Furthermore, the expense increases with the scale of requests monotonically.

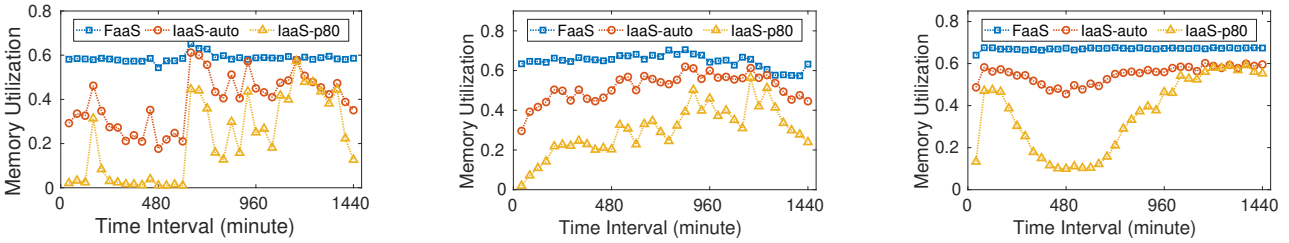
However, the turning points of all the cases differentiate from each other. The turning point for the bursty case appears the latest, while the turning point of the regular

4. https://aws.amazon.com/ec2/pricing/on-demand/
5. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-cloudwatch-new.html



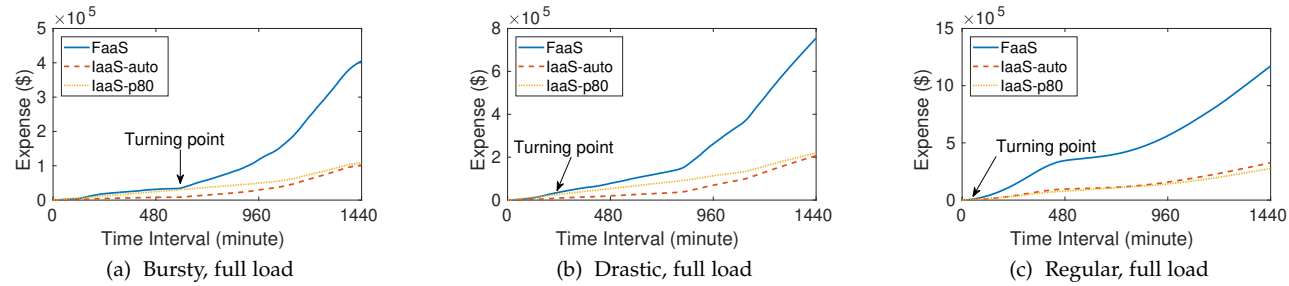
(a) Bursty (b) Drastic (c) Regular

Figure 4: Three typical workload traces selected from Azure Functions Trace 2019.



(a) Bursty, 10% load (b) Drastic, 10% load (c) Regular, 10% load

Figure 5: The memory utilization of IaaS and FaaS.



(a) Bursty, full load (b) Drastic, full load (c) Regular, full load

Figure 6: The expense of IaaS and FaaS.

case shows up the earliest. Such a phenomenon indicates that the bursty case is the most suitable.

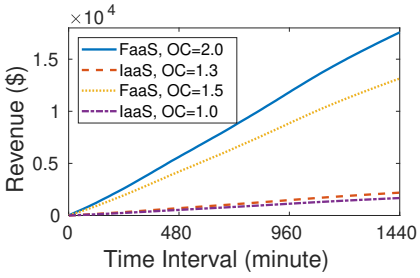


Figure 7: The revenue of FaaS and IaaS under different memory OC ratios.

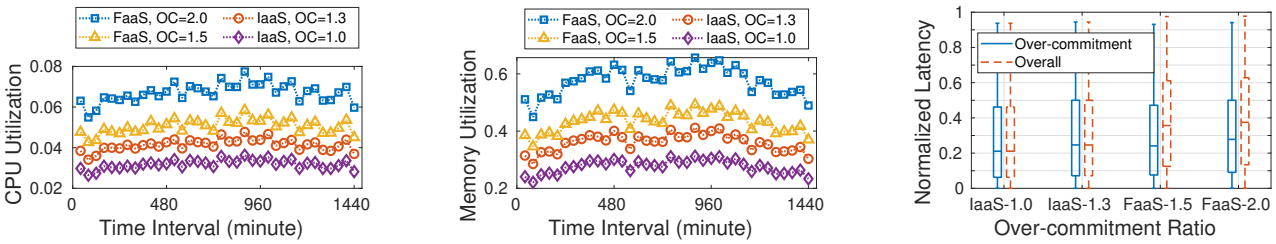
6.3 Evaluation of Provider's Cost

To evaluate the revenue and resource utilization of FaaS, we select the function invocation counts of a day to simulate the user demand of an IaaS/FaaS from the Azure Functions Trace 2019 [8].

Setup. To investigate how the OC ratio affects the resource utilization and revenue of providers, we conduct a series of experiments under various values of the OC ratio. As analyzed in Section 4, the memory OC ratio of FaaS can be higher than that of IaaS. Based on the practical experience of IaaS from IBM and OpenStack, the memory over-commitment ratio should be set as 1.5 [10], [11]. Hence, we tune the memory OC ratio of IaaS from 1.0 to 1.3 and tune that of FaaS from 1.5 to 2.0, respectively.

Comparison of revenue. As plotted in Figure 7, compared with IaaS, since the duration price of FaaS is higher than that of IaaS, the revenue of FaaS is 6.0×–10.5× higher than that of IaaS. Meanwhile, with the OC ratio increasing, the revenue of FaaS and IaaS increase by 1.3× for selling resources to more users.

Comparison of resource utilization. In terms of resource utilization, as illustrated in Figure 8(a) and Figure 8(b), a larger OC ratio improves resource utilization of both IaaS and FaaS. As analyzed in Section 4, IaaS fails to over-commit more memory nor CPU, making the resource utilization of IaaS lower than that of FaaS.



(a) CPU Utilization (b) Memory Utilization (c) Overhead

Figure 8: The resource utilization and overhead of FaaS and IaaS under different memory OC ratios.

Comparison of overhead. Figure 8(c) illustrates the overhead of FaaS and IaaS, including the overhead brought by over-commitment, BaaS services, and cold start effect. These factors increase the tail latency, which contributes to the revenue as well. We observe that the overhead of IaaS is generally less severe than FaaS. Although raising the OC ratio can improve resource utilization and increase revenue, with the CPU OC ratio increasing, it introduces frequent context switches, which may degrade performance and harm stability.

7 Pricing of Future Function

In this section, based on the analysis of the cost model, we identify the disadvantages of the widely adopted pricing strategy of FaaS and propose a dynamic pricing strategy for FaaS.

7.1 Primitive Pricing Strategy of FaaS

We identify three disadvantages of the prevailing pricing policy, including unfairness, risk of expense explosion for users, and lack of incentive scheme for providers. The pricing of functions contains two components: request and duration. The duration price is the price of computing resources that dominates the expense/revenue of users/providers, so we here only discuss the pricing strategy of the duration price.

Unfairness. The duration expense of functions is calculated by multiplying a fixed duration price by the resource capacity allocated to a function and the execution duration of the function. Such static pricing causes severe unfairness for two reasons: the heterogeneity of cloud platforms and the fluctuating demand of cloud users.

The infrastructure of cloud providers is a heterogeneous system where servers have various types of hardware and architectures of software, resulting in different computing performances. Consequently, some functions may be scheduled to high-end servers while some are not, leading to unfairness across FaaS users. From the perspective of providers, it is also unfair to charge users a fixed price regardless of the heterogeneity of infrastructure. Even if the underlying infrastructure is homogeneous, the performance of cloud platforms is volatile due to fluctuating user demand. Hence, FaaS users are paying for different qualities of services but at the same price. To eliminate the unfairness, the price of functions is supposed to be dynamic, instead of remaining unchanged.

Risk of expense explosion for users. Based on the real-world trace released by Azure [8], the regular request arrival pattern is ubiquitous. However, with the prevailing static pricing strategy, regular patterns will incur expense explosions. For the drastic and bursty cases, with the volume of requests increasing, FaaS users face the risk of expense explosions as well. For example, a start-up burnt $72,000 when internally testing Cloud Run within a few hours [12]. As a result, serverless users may have concerns about causing expense explosion.

Lack of incentive scheme for providers. On the one hand, based on the best practices advised by FaaS providers, the most typical use cases of FaaS are online applications. The report of SPEC also reveals that 86% of the functions are triggered on-demand as a direct result of a user interacting with applications [13].

On the other hand, many academic works and FaaS platforms extend use cases of FaaS to batch jobs, such as big data analytics, video processing, and model training [1], [14], [15], [16], [17], [18], [19]. Most batch jobs do not require extreme scalability and interactivity but expect a cheap price.

Due to lacking delay-tolerant use cases, the use scenarios of serverless are limited. The providers are required to offer discounts for the users as incentives to widen the use scenarios and attract more users. Furthermore, serverless providers can over-commit more resources, implying that they need to deal with a more severe waste of spare capacity than IaaS. However, the widely adopted serverless pricing strategy provides little incentive for FaaS users to cooperate with the provider in increasing user demand and improving overall resource utilization.

Summary. Today's serverless pricing strategy charges users the same price, which is static, coarse-grained, and unfair. As a result, FaaS providers are required to develop a dynamic, fine-grained, and transparent pricing strategy, not only meeting requirements of users for fairness and reducing expense but also helping providers exploit spare capacity and raise revenue.

7.2 How to Design a Dynamic Pricing?

To design such a dynamic pricing strategy, providers face a key challenge: *how to offer discounts to users as well as avoid the loss of revenue?* One promising solution is to attract extra workloads to offset the loss brought by discounts. Typically, applications outsourced to the cloud can be classified into

online services and batch jobs. Different from online services, batch jobs run without end-user interaction, hence choosing when to submit batch jobs is mostly flexible. As a result, FaaS providers can influence the timing and amount of batch job submissions with dynamic (lower) prices.

As analyzed in Sec. 7.1, concerning most use cases of FaaS are on-demand services, attracting extra batch jobs meets the requirements of widening the use scenarios of FaaS. Furthermore, extra user demand of batch jobs can not only offset the loss of discounts but also shape the overall resource utilization.

To this end, the providers face the other challenge: *how to determine the dynamic prices based on platform capacity and user demand?* Concerning the capacity being limited and multiple users expecting to gain discounts, an *auction-based* strategy is appropriate to enable the negotiation between the users and the provider.

Is Spot Instance directly applicable? In terms of dynamic pricing, AWS Spot Instance [20] is already released to the cloud market, where users can pay a spot price that is in effect for the period their instances are running. Can the pricing strategy of Spot Instance be directly applied to FaaS? Compared with Spot Instance, the challenges of designing an auction-based pricing strategy for FaaS mainly are threefold.

First, IaaS users bid for occupying resources, while FaaS users bid for executing functions. When IaaS users are rejected in the auction, their spot instances will be suspended. However, for FaaS, it is challenging to save states of functions for the short life-cycle, high concurrency, and statelessness.

Second, for IaaS, Spot Instance is mainly designed for batch jobs, which is supplementary to on-demand and reserved instances. However, a universal auction-based pricing strategy that covers both online services and batch jobs is required for FaaS.

Third, for Spot Instance, all the users who win the instances share the same payments; this is reasonable since cloud resources are identical. However, FaaS basically provisions function execution services, thus the performance of every execution is volatile due to user demand, underlying infrastructure, platform capacity, etc. As a result, users accepted in the auction should pay different prices.

To deal with the challenges, we propose *future function* and design auction-based pricing.

7.3 Future Function Auction

We propose an auction where users bid for *future function execution*. Specifically, at the beginning of each time slot, the FaaS provider determines how many resources to be prepared for discounted functions during the future time slot. All the users are allowed to bid for the discounted functions with expected prices (namely, the bidding prices, which are lower than the original price). For the user whose bid is accepted, functions invoked during the future time slot can be paid based on the payment price (lower than the respective bidding price) determined by the FaaS provider. Otherwise, users who are rejected in the auction will pay for function execution with the original price.

Algorithm 1: Winner selection

Input : Bids submitted by users, $B = \{B_i, \forall i \in N\}$
Resource capacity, $U = \{U_i, \forall j \in M\}$
Output: Bids accepted in the auction, W
Bids rejected in the auction, F

```
Sort B based on virtual value w in descend order;
for B_i ∈ B do
    if d_i < U then
        W ∪ i;
        Update resource capacity U;
    else
        F ∪ i;
    end
end
```

Table 3: The notations of future function.

Notation	Description
M	The number of resource types.
N	The number of users who participate in an auction.
B_i^u	The bid submitted by user i for future functions.
λ_i	The estimated total request scale.
b_i	The expected (bidding) price of user i.
B_i	The transformed bid of user i.
$\vec{d_i}$	The total resources consumed by user i.
d_i^j	The resource j's consumption of user i's functions.
r_i	The total scale of resources consumed by user i.
p_i	The discounted price of future functions of user i.

Bidding for future function. Suppose that the FaaS has M types of resources and the number of users who participate in the auction is denoted as N.

User interface. Each bid submitted by user i for future functions is denoted as $B_i^u = (\lambda_i, b_i)$, where λ_i and b_i are the estimated total request scale and the expected bidding price during the future time slot, respectively. To estimate a precise bidding price b_i, FaaS users need to develop bidding schemes based on historical request scale and resource usage. These metrics can be easily accessed via RESTful API which is typically provided by commercial and open-source FaaS platforms, including invocation times, CPU total time, memory utilization, and so on [21], [22]. To relieve FaaS users of developing bidding schemes, FaaS providers can set basic schemes for users. If FaaS users choose to apply more precise bidding schemes for more discounts, it is feasible for FaaS users to utilize related techniques as IaaS users do in AWS Spot Instance market [23], [24], [25].

Provider transformation. Each bid of user i is transformed to $B_i = (\vec{d_i}, r_i, b_i)$ by the FaaS provider. $\vec{d_i}$ is the total resources consumed by functions of user i, which is evaluated based on λ and historical statistics. Here $\vec{d_i} = (d_i^1, ..., d_i^M)$ is a demand vector with d_i^j representing how much resource j user i's functions consume. r_i is the total scale of resources the provider charges user i, depending on λ_i and l_i, where units of r_i are $\$/(GB \cdot second)$.

Given the bids from the users $\{B_i | i \in N\}$, the FaaS provider selects winners and determines corresponding payment vectors, which are denoted as $\vec{x} = (x_1, ..., x_N)$ and $\vec{p} = (p_1, ..., p_N)$, respectively. If $x_i = 1$, it means user i

wins the future functions and it pays p_i for them. The utility obtained by user i is denoted as $(v_i x_i - p_i) r_i$, where v_i is the true value of the functions. Note that if the user, who wins r_i of resources, yet fails to use them up, i.e., the scale of resources allocated to future functions is less than r_i, it still has to pay for r_i of resources with price p_i. If the scale of resources allocated to functions exceeds r_i, the user has to be charged at the original price for the exceeding part. Otherwise, if $x_i = 0$, it means user i is rejected and it has to be charged with the original price for function invocations.

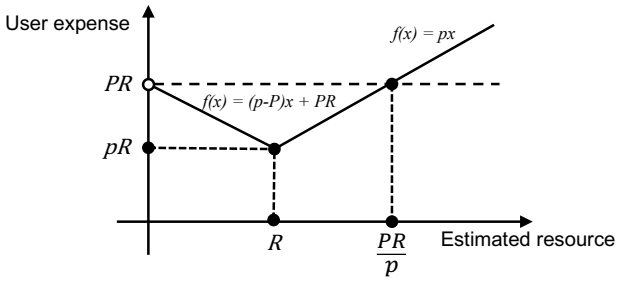


Figure 9: The impact of estimation error on expense.

Figure 9 plots the gap between the original expense and the discounted expense if a user's bid is accepted. As shown in the figure, p is the discounted price and P is the original price. R is the real resource scale charged by the provider, mainly depending on λ. x is the estimated value of the resource scale, so the expense of a winner can be denoted as follows.

$$f(x) = \begin{cases} (p-P)x + PR, & x < R \\ px, & x \geq R \end{cases}$$

Based on the above function, the estimated value should be as accurate as possible so that the user can obtain the largest benefit. Furthermore, the estimated value should not exceed $\frac{PR}{p}$, or the user will suffer from loss.

It is common to predict future request scales based on historical data, such as LSTM, ARIMA, and so on. Although it is challenging to predict request spikes, existing work can achieve pretty high accuracy. For example, the overall prediction accuracy of the number of calls per minute can reach as high as 92.3% on average [26].

Definition 1. *A future function auction is truthful, if and only if each user is to report its true valuation, i.e., $b_i = v_i$, which always maximizes its utility.*

Definition 2. *The social welfare in future function auction is calculated as the sum of the user utility $\sum_{i \in N}(v_i x_i - p_i) r_i$ and the FaaS provider's utility $\sum_{i \in N} p_i r_i$. The social welfare hence becomes $\sum_{i \in N} v_i r_i x_i$.*

The social welfare maximization problem in the function auction can be formulated into an integer linear problem:

$$\begin{aligned} \max \quad & \sum_{i=1}^{N} v_i r_i x_i \\ \text{s.t.} \quad & f_j(d_1^j x_1, ..., d_N^j x_N) \leq U_j, \forall j \in M \end{aligned} \tag{1}$$

where the constraint enforces the resource demand within the supply. The auction is truthful if each bidder's best strategy is always to reveal her true valuation, regardless of the other bidders' valuations, and regardless of how bids are decided. The social welfare is the sum of the values of both bidders and providers.

Algorithm 2: Payment determination

Input : Bids accepted in the auction, W
Bids rejected in the auction, F
Output: Payment of bids accepted in the auction, $\vec{p}$

```
for B_i ∈ W do
    Update U⃗_temp = U⃗ − Σ_{j=1}^{N_ni} d⃗_j;
    for B_j ∈ F do
        if U⃗_temp + d⃗_j < U⃗ then
            U⃗_temp = U⃗_temp + d⃗_j;
            if U⃗_temp + d⃗_i < U⃗ then
                U⃗_temp = U⃗_temp + d⃗_i;
                Record the less critical bid lc = j;
            else
                Calculate payment p_i = (b_j r_j D_i) / (r_i D_j);
            end
        end
    end
    if j == length(F) then
        Calculate payment p_i = (b_lc r_lc D_i) / (r_i D_lc);
    end
end
```

7.4 Determining the Price for Future Functions

Theoretically, the Vickrey–Clarke–Groves (VCG) mechanism can be employed to generate maximum social welfare while enforcing truthfulness. However, a VCG mechanism is computationally infeasible since Problem (1) is proved to be NP-hard in [27], whose computation complexity is too high for the future function auction.

As a result, we adopt a sub-optimal algorithm to solving the future function auction, which is proved to be truthful [28], [29], ensuring that $b_i = v_i$. The solution is described as follows.

Step 1: winner selection. Calculate the virtual value of each bid B_i, which is defined as $w(i) = \frac{b_i r_i}{D_i}$, where D_i is the weighted arithmetic mean of $\vec{d_i}$, i.e., $D_i = \sum_{m=1}^{M} \omega_i^m \cdot d_i^m$. The virtual value indicates how much user i pays for the resources its functions consume. Then based on the virtual value, sort all the bids in descending order and apply a greedy strategy to select accepted bids until the resource supplied to discounted functions is used up. The detailed pseudo code is shown in Algorithm 1.

Step 2: payment determination. For any bid B_i of accepted bids W, we select a critical bid from the rejected bids F and determine the payment of the accepted bid. We first update W by excluding B_i as W^0. Then for any bid B_j in F, we select the first rejected bid B_j which makes B_i fail to be selected in W^0 (the total resource demand exceeds the supply), i.e., the first rejected bid B_j whose resource demand is less than that of the accepted bid B_i, as the critical bid. Finally, the payment p_i is calculated as $\frac{b_j r_j D_i}{r_i D_j}$. In other words, the payment is derived based on the virtual

value of the critical bid B_j. The detailed pseudo code is shown in Algorithm 2.

The complexity of the algorithms is $O(MN^2 + N \log N)$, where N is the total number of bids submitted by users and M is the total number of resource types. The complexity of Algorithm 1 and Algorithm 2 are $O(N \log N)$ and $O(MN^2)$, respectively.

Announcement. After the payment of users is determined, the announcement is required. In future function pricing, the announcement contains two parts: discount and virtual value. The discount announcement is for accepted bids, while the other is for rejected bids. The discount announcement sorts discounts in ascending order, every individual winner can see the ranks in the announcement. For the rejected users, they can see their ranks in virtual value, so as to adjust the bidding strategy.

Security. A reverse inference process that investigates how the provider set prices will not harm the security of bidding systems. For example, Zheng et. al [23] study providers' setting of the bidding prices and answer the question of how to bid for cloud resources. In terms of preventing malicious bids, the truthfulness of future function auction can achieve that.

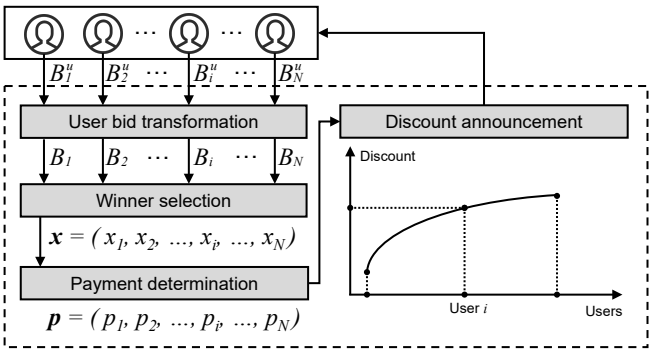


Figure 10: Expense of serverless is calculated based on the number of requests for functions and the execution duration of functions.

Summary. By now, serverless users can bid for future functions with dynamic prices yet without suspending them. Then we seek to design a pricing mechanism to cover both online services and batch jobs. As shown in Figure 10, users submit bids for future functions, then the FaaS provider transforms the user bids into candidate bids. After winner selection and payment determination, the FaaS provider can determine which bids are accepted or rejected.

7.5 Weighing Loss and Profit

From the perspective of providers, for online services, since the users are unable to control the request arrival, the users have to pay for function execution whatever the price is. If the future function auction which covers both online and batch users is introduced, FaaS providers have to offer future functions (i.e., discounted functions) to online services, resulting in a loss of profit. However, the spare capacity can be exploited to attract extra batch jobs to offset the loss.

As shown in Figure 12, resources consumed by online services are referred to as base capacity, denoted as $\vec{U}_o$. Resources consumed by extra batch jobs are referred to as spare capacity, denoted as $\vec{U}_b$. Hence, with the future function pricing, the sum of the base and spare capacity may approach the max capacity of the platform, namely $\vec{C}$. Furthermore, the profit brought by extra batch jobs can offset the loss incurred by discounts offered for online services. To reduce the loss, the serverless provider can decide how many resources to support future functions of the online users by tuning the supply ratio θ (a ratio of the capacity consumed by online future functions to the base capacity). The FaaS provider can choose to prefer profit or discount.

The problem of weighing the loss of online services and the profit of batch jobs can be formulated as a bilevel programming problem.

$$\begin{aligned} \max_{;\vec{x}^b;\vec{x}^o} \quad & \mathcal{F}(\theta, \vec{x}^b, \vec{x}^o) \\ \text{s.t.} \quad & \vec{U}_b + \theta\vec{U}_o \le \vec{C} \\ & \vec{x}^o \in \max \ \mathcal{A}(\vec{x}, \theta\vec{U}_o) \\ & \vec{x}^b \in \max \ \mathcal{A}(\vec{x}, \vec{U}_b) \end{aligned}$$

where $\max \mathcal{A}(\vec{x}, \vec{U})$ is the future function auction problem (1) and $\mathcal{F}(\theta, \vec{x}^b, \vec{x}^o)$ is defined as the sum of loss of online services and profit of batch jobs.

Pricing scheme of future functions. The future function pricing strategy is developed as follows. At the beginning of each time slot, the spare capacity is supplied to discounted functions of batch jobs. Then by evaluating the profit brought by the spare capacity, the FaaS provider determines how many resources to be supplied to discounted functions of online services by tuning the supply ratio. Then FaaS users can bid for discounted functions and the FaaS provider selects specific ones to accept their bids and determines their payments.

Table 4: Discount and acceptance ratio of online services and batch jobs.

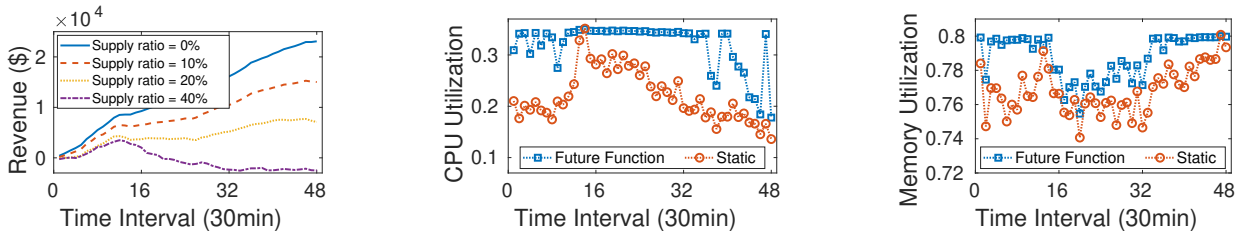
	Supply	Pay/Bid	Pay/Price	Acceptance
Online	10%	54.73%	46.08%	11.42%
	20%	54.47%	40.54%	9.90%
	40%	58.19%	43.51%	8.14%
Batch	-	71.90%	39.89%	24.5%

Summary. Basically, future function provides a bilateral mechanism where both users and providers can collaborate to gain benefit. To reduce the risk of expense explosion, users should estimate resource usage and request scale as precisely as possible. For providers, to improve resource utilization and attract user workloads, they are required to select winners as many as possible.

7.6 Evaluation of Future Function Pricing

We now evaluate the future function pricing to answer the following questions: 1) Can future function exploit the spare capacity; 2) How much profit can future function gain; 3) How much discount can future function offer?

Setup. We select the resource utilization statistics during 24 hours from Alibaba Cluster Trace [9] which includes about 4000 machines in a period of 8 days to evaluate the future function pricing strategy. We set the expected max



(a) Revenue (b) CPU utilization (c) Memory utilization

Figure 11: Revenue, CPU, and memory utilization of the FaaS platform under the future function pricing compared to the static pricing.

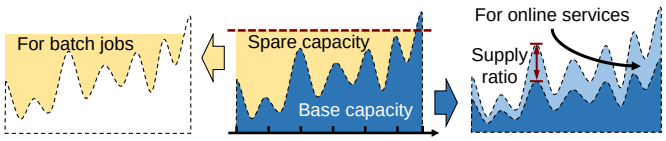


Figure 12: Spare capacity is exploited to offset the loss incurred by discounted functions of online services.

CPU and memory utilization as 35% and 80%, respectively. If the CPU and memory utilization are below the expected thresholds, the spare capacity will be used to supply discounted functions for batch jobs. For the online services, we set the supply ratio as 10%, 20%, and 40%. The price and spec of functions and VMs are the same as those in Section 6. In the simulation, we randomly generate a bidding price ranging from 0.5 to 0.99 of the original price. Then, the average values of the historical request scale and resource usage during the last time slot are selected as estimated values.

The future function involves two players: the provider and the users. The revenue and discounts depend on winner selection and payment determination strategies of providers, which are required to ensure the future function auction is truthful. Meanwhile, the expense of users is determined by the estimation of resource scale. The estimation techniques should be as accurate as possible so that the bidding price approaches the true valuation.

In this section, we aim to evaluate the performance of the future function auction, especially the winner selection algorithm (Algorithm 1) and the payment determination algorithm (Algorithm 2). Hence, for bidding strategies of users, we randomly generate a bidding price ranging from 0.5 to 0.99 of the original price. Since prediction techniques are not the main contribution of the future function auction, we use average values of the historical data as estimated values and omit evaluating user benefits.

Resource utilization. As shown in Figure 11, the CPU and memory utilization increase and approach 35% and 80%, respectively. With the future function pricing, the discounted functions for batch jobs improve the resource utilization of the FaaS platform, implying that the FaaS provider can exploit the spare capacity of infrastructure as fully as possible.

Revenue. As shown in Figure 11(a), with supplying more resources to discounted functions of online services, the revenue of the FaaS provider decreases. The 0% supply ratio case means the FaaS platform does not offer discounts to online services at all, so the revenue is the profit brought by the extra batch jobs. When the supply ratio is 40%, it incurs a loss of $2544.2. Otherwise, the profit brought by the spare capacity can offset the loss and make more profit. By tuning the supply ratio, future function pricing allows the FaaS provider to choose which is preferred, discount or profit.

Discount. Table 4 demonstrates the discount rate and the acceptance ratio of future function pricing. The acceptance ratio is derived by dividing the number of accepted bids by the total number of bids. Among all the online users who participate in bidding future functions, 8.1%–11.4% of the users can win the future functions. Generally, winners can enjoy a 40.5%–46.1% discount and their payments are 54.4%–58.1% cheaper than their bids. Meanwhile, 24.5% of bids for the discounted batch functions are accepted and the price reduces by 57.5% on average.

8 Related Work

Regarding serverless, [30] and [31] conduct a comprehensive review of the pros and cons of serverless. Existing works broadly can be classified into two categories. One is trying to optimize the serverless platform to improve performance and efficiency [32], [33], [34], [35], [36], [37]. The other aims to outsource various applications to serverless platforms, to acquire high performance with low cost [1], [15], [16], [17], [18], [19], [30], [31], [38], [39], [40], [41], [42], [43], [44], [45], [46]. However, none of the works justifies whether serverless is cost-saving or profitable.

Performance of serverless. ServerlessBench is an open-source benchmark suite for characterizing the performance of platforms [47]. Wang *et al.* conduct a measurement on mainstream serverless platforms from the perspective of users [48]. Li *et al.* identifies the critical factors of serverless platforms based on several popular open-source platforms [49]. SAAF is developed to profile performance and resource utilization of functions, so as to accurately predict performance of FaaS-based applications [50]. Based on developers' bid on resources, Bermbach *et al.* propose auction-based approaches that determine what functions should be offloaded from edge (fog) to cloud, aiming for improving resource efficiency [51], [52]. However, none of the studies conducts a review or quantitative analysis of the cost of serverless. Maissen *et al.* develop FaaSdom, a benchmark for FaaS platforms [53]. One of the test cases examines the

cost of the current mainstream serverless cloud providers but fails to compare the cost to IaaS platforms.

Cost of serverless. Lin *et al.* develop an economic model to analyze the incentives of users and providers to adopt serverless [54]. Different from [54], our model systematically examines the cost of serverless, to reveal the best use scenarios of serverless and identify the drawbacks of the prevailing serverless pricing. Wang *et al.* conduct a simple measurement of the cost of FaaS and IaaS by running machine learning training jobs on AWS Lambda and EC2, respectively [18]. Driven by machine learning interference workloads, Spock [55] evaluates the cost of serverless and IaaS. Costless [56] optimizes the cost by fusing and placing functions based on application workflows. Tayal *et al.* study two real cases to compare the cost of serverless and serverful applications and show that serverless is more expensive than traditional cloud [57]. Each of these works provides a case study of cost for a particular scenario, failing to provide a comprehensive understanding of serverless cost. Our cost model covers general request arrival patterns, aiming at understanding and removing the cost barriers of adopting serverless from the perspectives of users and providers. A cost prediction of serverless workflows is developed based on a Monte Carlo simulation of an abstract workflow model [58]. However, such a cost prediction methodology can not be used to compare the cost of serverless with IaaS.

Our work is one of the first studies that develops an analytical model for serverless cost from the perspectives of users and providers, by comparing it to IaaS cloud. Our cost model mainly focuses on cloud scenarios, where a massive scale of infrastructure is located in one region. We try to analyze the cost when homogeneous infrastructure is installed as IaaS and FaaS cloud, respectively. Hence, benefiting from high bandwidth within a datacenter, the impact of function placement can be omitted. Our cost model does not cover the case where VMs/functions are placed in different regions. Recently, edge computing has been evolving into a serverless fashion. As a result, to adapt edge-cloud scenarios, the model should be extended by taking multiple factors relating to the cost of placement into consideration, such as network bandwidth between edge and cloud, the volume of data to be transferred among functions, hardware specifications, and so on [52], [59]. How to analyze the cost of IaaS and FaaS in edge-cloud scenarios, where VMs and functions are located in geo-distributed regions, requires further study yet is out of the scope of our work.

Cost and performance of serverless. Lin *et al.* propose a heuristic algorithm to balance the performance and cost of serverless applications, ensuring quality of service as well as controlling cost [60]. AMPS-Inf [61] is designed to save cost and guarantee the Service Level Objective (SLO) of model inference. These works mainly deal with the trade-off between the cost and performance of serverless applications, yet fail in answering when serverless is more cost-saving and more profitable than IaaS.

Pricing of serverless. Concerning pricing, most serverless platforms adopt a static pricing strategy that charges users a fixed price. To the best of our knowledge, the future function is the first auction-based pricing plan for FaaS, which provides a bilateral negotiation for the future price of functions between the provider and the users. Mahajan *et al.* propose to deploy applications to both FaaS and IaaS simultaneously to minimize cost. Then cloud providers can set different prices for three solutions, i.e., VM, VM+Function, and Function [62]. Yet, the price schemes in [62] are static, which cannot provide discounts for users and increase profits for providers.

9 Conclusion

We develop a cost model for serverless from the perspectives of users and providers to answer how to use and operate serverless computing services. Extensive trace-driven simulations reveal that serverless is not always cost-saving due to potential cost issues caused by the prevailing pricing. From the perspective of providers, serverless is more profitable than IaaS. However, today's serverless pricing prevents providers from profiting more. To remove the cost barrier brought by the pricing, we propose the future function, an auction-based pricing strategy, to reduce the risk of expense explosion for users as well as increase profits for providers. Experimental results show that the duration price of winners can be reduced by 57.5% on average and the revenue of the provider barely shrinks.

References

[1] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the Cloud: Distributed Computing for the 99%," in *Proceedings of ACM SoCC*, 2017.

[2] AWS, "AWS Lambda," [Online], https://aws.amazon.com/lambda/.

[3] Microsoft Azure, "Azure Functions," [Online], https://azure.microsoft.com/en-us/services/functions/.

[4] Google Cloud Platform, "Cloud Functions," [Online], https://cloud.google.com/functions.

[5] Alibaba Cloud, "Function Compute," [Online], https://www.alibabacloud.com/product/function-compute.

[6] Amazon, "Amazon EC2 Reserved Instances Pricing," [Online], https://aws.amazon.com/ec2/pricing/reserved-instances/pricing/.

[7] Amazon, "AWS EC2," [Online], https://aws.amazon.com/ec2/.

[8] Microsoft Azure, "Azure Functions Trace 2019," [Online], https://github.com/Azure/AzurePublicDataset/blob/master/AzureFunctionsDataset2019.md.

[9] Alibaba, "Alibaba Cluster Trace Program 2018," [Online], https://github.com/alibaba/clusterdata.

[10] IBM, "Resource Overcommit Allocation Ratios," [Online], https://www.ibm.com/docs/en/prs/2.4.0?topic=administering-resource-overcommit-allocation-ratios.

[11] OpenStack, "Overcommitting CPU and RAM," [Online], https://docs.openstack.org/arch-design/design-compute/design-compute-overcommit.html.

[12] Sudeep Chauhan, "We Burnt $72K testing Firebase - Cloud Run and almost went Bankrupt," [Online], https://blog.tomilkieway.com/72k-1/ & https://blog.tomilkieway.com/72k-2/.

[13] S. Eismann, J. Scheuner, E. Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. Abad, and A. Iosup, "A Review of Serverless Use Cases and their Characteristics," in *arXiv*, 2021.

[14] V. Shankar, K. Krauth, K. Vodrahalli, Q. Pu, B. Recht, I. Stoica, J. Ragan-Kelley, E. Jonas, and S. Venkataraman, "Serverless Linear Algebra," in *Proceedings of ACM SoCC*, 2020.

[15] J. Sampé, G. Vernik, M. Sánchez-Artigas, and P. García-López, "Serverless Data Analytics in the IBM Cloud," in *Proceedings of ACM/IFIP Middleware*, 2018.

[16] L. Ao, L. Izhikevich, G. M. Voelker, and G. Porter, "Sprocket: A Serverless Video Processing Framework," in *Proceedings of ACM SoCC*, 2018.

[17] S. Fouladi, R. S. Wahby, B. Shacklett, K. V. Balasubramaniam, W. Zeng, R. Bhalerao, A. Sivaraman, G. Porter, and K. Winstein, "Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads," in *Proceedings of USENIX NSDI*, 2017.

[18] H. Wang, D. Niu, and B. Li, "Distributed Machine Learning with a Serverless Architecture," in *Proceedings of IEEE INFOCOM*, 2019.

[19] J. Carreira, P. Fonseca, A. Tumanov, A. Zhang, and R. Katz, "Cirrus: A Serverless Framework for End-to-End ML Workflows," in *Proceedings of ACM SoCC*, 2019.

[20] Amazon, "AWS EC2 Spot Instances," [Online], https://aws.amazon.com/cn/ec2/spot/.

[21] AWS Lambda, "Working with Lambda function metrics," [Online], https://docs.aws.amazon.com/lambda/latest/dg/monitoring-metrics.html.

[22] Amazon CloudWatch, "Metrics collected by Lambda Insights," [Online], https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/Lambda-Insights-metrics.html.

[23] L. Zheng, C. Joe-Wong, C. W. Tan, M. Chiang, and X. Wang, "How to Bid the Cloud," in *Proceedings of ACM SIGCOMM*, 2015.

[24] Y. Song, M. Zafer, and K.-W. Lee, "Optimal bidding in spot instance market," in *Proceedings of IEEE INFOCOM*, 2012.

[25] M. Khodak, L. Zheng, A. S. Lan, C. Joe-Wong, and M. Chiang, "Learning Cloud Dynamics to Optimize Spot Instance Bidding Strategies," in *Proceedings of IEEE INFOCOM*, 2018.

[26] S. Luo, H. Xu, K. Ye, G. Xu, L. Zhang, G. Yang, and C. Xu, "The power of prediction: Microservice auto scaling via workload learning," in *Proceedings of ACM SoCC*, 2022.

[27] W. Vickrey, "Counters peculation, auctions, and competitive sealed tenders," *Journal of Finance*, vol. 16, pp. 8–27, 1961.

[28] D. Lehmann, L. I. Oćallaghan, and Y. Shoham, "Truth Revelation in Approximately Efficient Combinatorial Auctions," *J. ACM*, vol. 49, no. 5, p. 577–602, Sep. 2002.

[29] J. Jia, Q. Zhang, Q. Zhang, and M. Liu, "Revenue Generation for Truthful Spectrum Auction in Dynamic Spectrum Access," in *Proceedings of ACM MobiHoc*, 2009.

[30] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Menezes Carreira, K. Krauth, N. Yadwadkar, J. Gonzalez, R. A. Popa, I. Stoica, and D. A. Patterson, "Cloud Programming Simplified: A Berkeley View on Serverless Computing," EECS Department, University of California, Berkeley, Tech. Rep., Feb 2019.

[31] J. M. Hellerstein, J. M. Faleiro, J. E. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu, "Serverless Computing: One Step Forward, Two Steps Back," in *Proceedings of CIDR*, 2019.

[32] E. Oakes, L. Yang, D. Zhou, K. Houck, T. Harter, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "SOCK: Rapid Task Provisioning with Serverless-Optimized Containers," in *Proceedings of USENIX ATC*, 2018.

[33] I. E. Akkus, R. Chen, I. Rimac, M. Stein, K. Satzke, A. Beck, P. Aditya, and V. Hilt, "SAND: Towards High-Performance Serverless Computing," in *Proceedings of USENIX ATC*, 2018.

[34] S. Shillaker and P. Pietzuch, "Faasm: Lightweight Isolation for Efficient Stateful Serverless Computing," in *Proceedings of USENIX ATC*, 2020.

[35] Z. Shen, Z. Sun, G.-E. Sela, E. Bagdasaryan, C. Delimitrou, R. Van Renesse, and H. Weatherspoon, "X-Containers: Breaking Down Barriers to Improve Performance and Isolation of Cloud-Native Containers," in *Proceedings of ACM ASPLOS*, 2019.

[36] D. Du, T. Yu, Y. Xia, B. Zang, G. Yan, C. Qin, Q. Wu, and H. Chen, "Catalyzer: Sub-Millisecond Startup for Serverless Computing with Initialization-Less Booting," in *Proceedings of ACM ASPLOS*, 2020.

[37] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider," in *Proceedings of USENIX ATC*, 2020.

[38] Q. Pu, S. Venkataraman, and I. Stoica, "Shuffling, Fast and Slow: Scalable Analytics on Serverless Infrastructure," in *Proceedings of USENIX NSDI*, 2019.

[39] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica, "Ray: A Distributed Framework for Emerging AI Applications," in *Proceedings of USENIX OSDI*, 2018.

[40] A. Klimovic, Y. Wang, P. Stuedi, A. Trivedi, J. Pfefferle, and C. Kozyrakis, "Pocket: Elastic Ephemeral Storage for Serverless Analytics," in *Proceedings of USENIX OSDI*, 2018.

[41] S. Fouladi, F. Romero, D. Iter, Q. Li, S. Chatterjee, C. Kozyrakis, M. Zaharia, and K. Winstein, "From Laptop to Lambda: Outsourcing Everyday Jobs to Thousands of Transient Functional Containers," in *Proceedings of USENIX ATC*, 2019.

[42] F. Bakir, R. Wolski, C. Krintz, and G. S. Ramachandran, "Devices-as-Services: Rethinking Scalable Service Architectures for the Internet of Things," in *Proceedings of USENIX HotEdge*, 2019.

[43] A. Wang, J. Zhang, X. Ma, A. Anwar, L. Rupprecht, D. Skourtis, V. Tarasov, F. Yan, and Y. Cheng, "InfiniCache: Exploiting Ephemeral Serverless Functions to Build a Cost-Effective Memory Cache," in *Proceedings of USENIX FAST*, 2020.

[44] F. Xu, Y. Qin, L. Chen, Z. Zhou, and F. Liu, "λDNN: Achieving Predictable Distributed DNN Training With Serverless Architectures," *IEEE Transactions on Computers*, vol. 71, no. 2, pp. 450–463, 2022.

[45] Z. Wen, Y. Wang, and F. Liu, "StepConf: SLO-Aware Dynamic Resource Configuration for Serverless Function Workflows," in *Proceedings of IEEE INFOCOM*, 2022.

[46] L. Pan, L. Wang, S. Chen, and F. Liu, "Retention-Aware Container Caching for Serverless Edge Computing," in *Proceedings of IEEE INFOCOM*, 2022.

[47] T. Yu, Q. Liu, D. Du, Y. Xia, B. Zang, Z. Lu, P. Yang, C. Qin, and H. Chen, "Characterizing Serverless Platforms with Serverlessbench," in *Proceedings of ACM SoCC*, 2020.

[48] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking behind the Curtains of Serverless Platforms," in *Proceedings of USENIX ATC*, 2018.

[49] J. Li, S. G. Kulkarni, K. K. Ramakrishnan, and D. Li, "Understanding Open Source Serverless Platforms: Design Considerations and Performance," in *Proceedings of International Workshop on Serverless Computing*, 2019.

[50] R. Cordingly, W. Shu, and W. J. Lloyd, "Predicting Performance and Cost of Serverless Computing Functions with SAAF," in *Proceedings of IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress*, 2020.

[51] D. Bermbach, S. Maghsudi, J. Hasenburg, and T. Pfandzelter, "Towards Auction-Based Function Placement in Serverless Fog Platforms," in *Proceedings of IEEE International Conference on Fog Computing*, 2020.

[52] D. Bermbach, J. Bader, J. Hasenburg, T. Pfandzelter, and L. Thamsen, "Auctionwhisk: Using an auction-inspired approach for function placement in serverless fog platforms," *Software: Practice and Experience*, vol. 52, no. 5, pp. 1143–1169, 2022.

[53] P. Maissen, P. Felber, P. Kropf, and V. Schiavoni, "FaaSdom: A Benchmark Suite for Serverless Computing," in *Proceedings of ACM International Conference on Distributed and Event-Based Systems*, 2020.

[54] X. C. Lin, J. E. Gonzalez, and J. M. Hellerstein, "Serverless Boom or Bust? An Analysis of Economic Incentives," in *Proceedings of USENIX HotCloud*, 2020.

[55] J. R. Gunasekaran, P. Thinakaran, M. T. Kandemir, B. Urgaonkar, G. Kesidis, and C. Das, "Spock: Exploiting Serverless Functions for SLO and Cost Aware Resource Procurement in Public Cloud," in *Proceedings of IEEE CLOUD*, 2019.

[56] T. Elgamal, "Costless: Optimizing Cost of Serverless Computing through Function Fusion and Placement," in *Proceedings of IEEE/ACM Symposium on Edge Computing*, 2018.

[57] A. Tayal, E. Lam, D. Choudhury, M. Dickerson, G. Moovera, and G. Arora, "Determining the Total Cost of Ownership of Serverless Technologies when compared to Traditional Cloud," Deloitte Consulting, Tech. Rep., September 2019.

[58] S. Eismann, J. Grohmann, E. van Eyk, N. Herbst, and S. Kounev, "Predicting the Costs of Serverless Workflows," in *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, 2020.

[59] A. Sandur, C. Park, S. Volos, G. Agha, and M. Jeon, "Jarvis: Large-scale server monitoring with adaptive near-data processing," in *Proceedings of IEEE ICDE*, 2022.

[60] C. Lin and H. Khazaei, "Modeling and Optimization of Performance and Cost of Serverless Applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 615–632, 2021.

[61] J. Jarachanthan, L. Chen, F. Xu, and B. Li, "AMPS-Inf: Automatic Model Partitioning for Serverless Inference with Cost Efficiency," in *Proceedings of ACM ICPP*, 2021.

[62] K. Mahajan, D. Figueiredo, V. Misra, and D. Rubenstein, "Optimal Pricing for Serverless Computing," in *Proceedings of IEEE GLOBECOM*, 2019.