FaaSConf: QoS-aware Hybrid Resources Configuration for Serverless Workflows

Yilun Wang
Anhui University, China
wangyilun@stu.ahu.edu.cn

Pengfei Chen
Sun Yat-sen University, China
chenpf7@mail.sysu.edu.cn

Hui Dou*, Yiwen Zhang
Anhui University, China
{douhui,zhangyiwen}@ahu.edu.cn

Guangba Yu, Zilong He, Haiyu Huang
Sun Yat-sen University, China
{yugb5,hezlong,huanghy95}@mail2.sysu.edu.cn

ABSTRACT

Serverless computing, also known as Function-as-a-Service (FaaS), is a significant development trend in modern software system architecture. The workflow composition of multiple short-lived functions has emerged as a prominent pattern in FaaS, exposing a considerable resources configuration challenge compared to individual independent serverless functions. This challenge unfolds in two ways. Firstly, workflows frequently encounter dynamic and concurrent user workloads, increasing the risk of QoS violations. Secondly, the performance of a function can be affected by the resource re-provision of other functions within the workflow.

With the popularity of the mode of concurrent processing in one single instance, concurrency limit as a critical configuration parameter imposes restrictions on the capacity of requests per instance. In this study, we present FaaSConf, a QoS-aware hybrid resource configuration approach that uses multi-agent reinforcement learning (MARL) to configure hybrid resources, including hardware resources and concurrency, thereby ensuring end-to-end QoS while minimizing resource costs. To enhance decision-making, we employ an attention technique in MARL to capture the complex performance dependencies between functions. We further propose a safe exploration strategy to mitigate QoS violations, resulting in a safer and efficient configuration exploration. The experimental results demonstrate that FaaSConf outperforms state-of-the-art approaches significantly. On average, it achieves a 26.5% cost reduction while exhibiting robustness to dynamic load changes.

CCS CONCEPTS

• **Computing methodologies** → **Distributed computing methodologies**.

KEYWORDS

Serverless Computing; Configuration Tuning; MARL

*Hui Dou is the corresponding author.

ASE '24, October 27-November 1, 2024, Sacramento, CA, USA

ACM ISBN 979-8-4007-1248-7/24/10…$15.00
https://doi.org/10.1145/3691620.3695477

ACM Reference Format:
Yilun Wang, Pengfei Chen, Hui Dou*, Yiwen Zhang, and Guangba Yu, Zilong He, Haiyu Huang. 2024. FaaSConf: QoS-aware Hybrid Resources Configuration for Serverless Workflows. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3691620.3695477

1 INTRODUCTION

Serverless computing becomes an important software system architecture, primarily due to its ability to eliminate an application's reliance on a fixed server infrastructure. This software system architecture decouples the application from concerns of resource allocation and elastic scaling. Due to the fine granularity and highly elastic nature, both commercial FaaS providers [16, 24, 25, 27, 39] and open-source FaaS platforms [59, 60] are witnessing a surge in function workflows, such as video processing pipelines [23], machine learning workflows [76, 85]. Nowadays, numerous software developers are migrating their services to the FaaS platforms [7, 42, 77]. The serverless application is composed of multiple loosely coupled functions, separating process logic and function execution, pursuing fine-granularity and high elasticity, as well as modular deployment [92, 96],which have been growing in popularity. Despite these benefits, guaranteeing end-to-end Quality of Service (QoS) and enhancing resource utilization efficiency remain pivotal challenges for both cloud vendors and users. Cloud vendors such as AWS Lambda [39], Google Cloud Functions (GCF) [27], and Azure Functions [25] typically offer resource configurations like CPU and memory. A lot of prior works have focused on resource management for serverless functions. For instance, [1, 8, 67, 78, 90, 91] employed black-box optimization like BO or RL to learn optimal configuration tuning, and [22, 44, 52] models relationships between configurations and performance. While they are effective, another crucial issue the system has to tackle is resource configuration in serverless workflows, which exhibits the characteristics of concurrency, dynamic loads and performance dependencies. Specifically, the primary challenges of resource configuration for serverless workflows are as follows.

C1: Trade-off between QoS and cost under concurrent workloads. A workflow comprises multiple functions, resulting in a longer execution path and posing greater challenges in ensuring QoS compared to individual functions, particularly when confronted with concurrent requests. However, blindly scaling out the resources can lead to significant increase on costs and resources wastage.

C2: Complex performance dependencies. There are complex dependencies between functions in complex workflows, which lead to cascading effects when reconfiguring resources. Moreover, the degree of the interactions between functions varies, which is hard to precisely capture.

C3: Difficulty on policy learning. The configuration space of workflow grows exponentially in relation to (i) the number of functions and (ii) the number of configuration knobs, discovering optimal policies requires extensive explorations, which significantly increases the training difficulty on learning-based methods[66, 67].

However, prior methods are not practical enough in serverless workflows for the two major reasons. Firstly, many prior approaches excel in optimizing resources of individual functions [1, 8, 66, 75, 90]. However, when applying them to workflows, two limitations are observed in our experiments: (i) they primarily focus on optimizing the trade-off between QoS and cost of individual functions while disregarding the overall end-to-end performance of function workflows. (ii) The resource reconfiguration of one function can cause non negligible impact on other functions within the workflow. Hence, it is imperative to adopt a holistic view to maintain end-to-end QoS while considering the interdependencies to balance the performance between functions.

Secondly, most prior methods [8, 22, 74, 78, 90, 96] find the optimal configuration in scenarios with a low and simple workloads, assuming a low invocation frequency of functions and only focus on optimizing hardware resources. However, serverless workflows often face dynamic and concurrent loads, with typical scenarios being substantial requests in online software services and high concurrency event-driven tasks [3, 7, 42, 48]. These optimal configurations that serve a single workload can cause QoS violations under concurrent requests and cannot easily adapt to dynamic workloads. Recent studies [45, 72] introduce concurrency-based autoscaling methods to determine the maximum number of requests that can be processed in parallel per container. Based on our observations on serverless functions and workflows in different FaaS platforms in Sec.2.2, the *concurrency* configuration significantly impacts the overall system performance under concurrent workloads. However, hardware resources reconfiguration can shift optimal concurrency to be sub-optimal[45]. Therefore, a promising solution lies in the joint optimization of hardware resources and concurrency in order to provide ideal amount of resources while avoiding overload on existing resources.

FaaSConf Approach. To fill the gaps of existing works, we propose FaaSConf, a QoS-aware and hybrid resource configuration approach that aims to automatically unearth **FaaS** workflows' optimal hybrid resources **Conf**iguration. Our key observation is that *concurrency* configuration greatly influences function performance in the scenario of concurrent user requests, and should be optimized along with hardware resources to find the optimal solution, which is ignored in previous works [1, 8, 67, 75, 90, 96]. Treating each function as an agent, FaaSConf leverages multi-agent reinforcement learning (MARL) to simultaneously optimize both vertical and horizontal resources, as well as concurrency to support cost-efficient resource configuration and ensure QoS (**solution to C1**). To achieve efficient MARL for hybrid resources configuration, we combine *attention mechanism*[79] and *mean-field* technique[87] in MARL to not only capture the dynamic performance dependencies of functions automatically (**solution to C2**) but also reduce the computational complexity of policy learning (**solution to C3**). Furthermore, we propose a *safe exploration strategy* to correlate the price of resources with the quality of configurations to reduce QoS violations during online resource configuration optimization process (**solution to C1**).

Table 1: Comparison of state-of-the-art serverless resource configuration approaches.

Approach	Vertical	Horizontal	Concurrency	Workflow
COSE [1]	✓	✗	✗	✗
SLAM [71]	✓	✗	✗	✓
Sizeless [22]	✓	✗	✗	✓
FaaSDeliver [90]	✓	✗	✗	✗
Q-Learning [72]	✗	✗	✓	✓
Kraken [7]	✗	✓	✗	✓
StepConf [82]	✓	✓	✗	✓
SIMPPO [67]	✓	✓	✗	✗
ENSURE [75]	✓	✓	✗	✗
Aquatope [96]	✓	✓	✗	✓
ORION [51]	✓	✗	✗	✓
FaaSConf	✓	✓	✓	✓

Generally, we make the following contributions.

- We conduct extensive studies to reveal that existing resource configuration optimization methods for FaaS workflows can result in sub-optimality and inefficiency due to concurrent requests and function dependencies, which are the typical characteristics of workflows.
- We propose a MARL-based resource configuration tuning approach, namely FaaSConf, which jointly optimizes hardware resources and concurrency configuration in an efficient manner. We combine ***mean field*** and ***attention mechanism*** to not only learn the varied dependencies between FaaS functions, but also reduce difficulty of policy learning.
- We implement FaaSConf based on *OpenFaaS*, we evaluate FaaSConf with four baselines using realistic serverless workloads, demonstrate that FaaSConf significantly reduces the average resource costs of 26.5% compared to state-of-the-art baselines and is robust to dynamic workload changes.

2 BACKGROUND AND MOTIVATION

2.1 Background

Serverless Workflows. In addition to single serverless functions, another important aspect of serverless computing is workflows. Serverless workflows implement business logic through a series of interdependent functions. A single function completes independent tasks and interacts with other functions through coordination mechanism provided by the platforms or third-parties, or through internal calls [14, 15, 92]. Despite the numerous benefits of serverless workflows, the complex applications make function management increasingly difficult. Moreover, concurrent requests in a FaaS environment can lead to increased response time, QoS violations and potential resource bottlenecks due to computing resource contention.

Hybrid Resource Configuration. Effective resource configuring in software systems, especially for microservice and FaaS, has

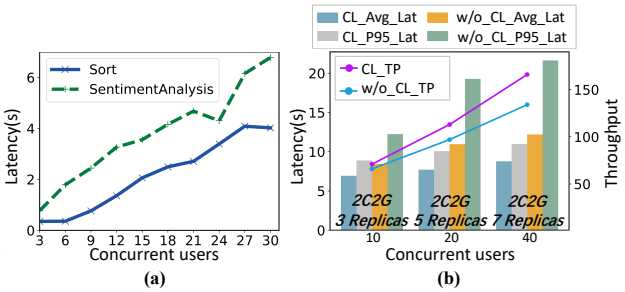
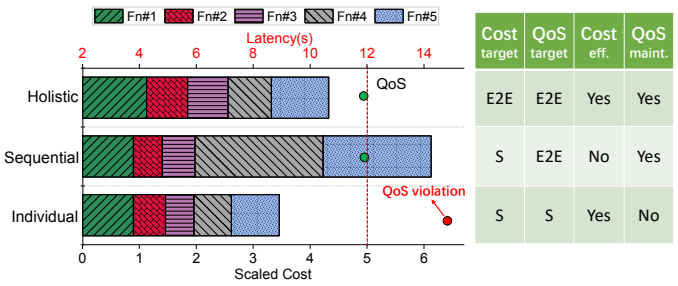


Figure 1: (a) shows the function latency increases with the concurrent workloads. (b) shows the average, P95 latency (Lat) and throughput (TP) with and without setting concurrency limit (CL) parameter in *Sequential* workflow under different workloads, we set CL=3 here.



Cost target	QoS target	Cost eff.	QoS maint.
E2E	E2E	Yes	Yes
S	E2E	No	Yes
S	S	Yes	No

Figure 2: Scaled cost and latency under three types of tuning methods to configure resources for a *Sequential* ML workflow. The bar chart represents the cost, while the dots represent the latency of each method. Table on the right shows the cost, QoS targets of optimization and whether cost efficiency and QoS maintaining.

gained a great attention among researchers. Unlike well-studied vertical resources (e.g., CPU, memory and network) and horizontal auto-scaling, *concurrency* is another important and configurable parameters in serverless platforms. However, in the traditional FaaS architecture, many frameworks initialize new containers for each request [17, 39], this approach encounters the cold start problem and low resource utilization. To mitigate this issue, the concurrent processing on single instance mode have been proposed as novel solutions in recent popular platforms [16, 25, 27, 37, 59, 60]. These methods allow for parallel processing of multiple requests within each container, e.g., OpenFaaS's watchdog forks a new process for each request within container[29, 59] and Knative's queue proxy allows a certain number of requests to enter the user-container simultaneously and queues the requests if necessary [37]. As highlighted in Tab.1, a significant portion of existing studies primarily focuses on vertical or horizontal resources allocation, only a small amount of prior work has noticed this issue. e.g., Schuler et al. adopt Q-Learning to decide this knob [72], Alibaba Function Compute platform [16] conducts offline stress testing to recommend the best concurrency value of a function under fixed hardware resources for user's functions, but it has poor generalization. Our study shows that optimizing the hybrid resources of vertical, horizontal resources and concurrency configuration effectively improves function performance and reduces resource costs.

Multi-Agent Reinforcement Learning. MARL is a popular field within machine learning that studies how multiple agents learn to make decisions in an environment. It combines principles from game theory such as strategic interactions, Nash equilibrium, and cooperative or competitive dynamics[5, 9, 95] with reinforcement learning to create systems where multiple decision-makers interact, and trying to maximize their long-term rewards. MARL is particularly relevant for scenarios where multiple agents must cooperate or compete with each other, such as in traffic light control [84] and robotic swarm coordination [62]. Recently, applying MARL to multi-functions resource allocation and scheduling is a promising way [41, 67, 70], agents exchange information or communicate with each other to consider the interdependencies and cross-functional impact of resource allocation.

2.2 Motivations

In this section, we show our empirical studies on hybrid resource configuration for serverless functions and workflows under concurrent requests and introduce our motivations.

Motivation 1: Concurrency significantly impacts function performance and should be optimized along with hardware resources to get optimal results. Excessive concurrent execution negatively impacts function performance due to competition for computation, memory, and network resources, thereby degrades user experience. We find this issue on different FaaS platforms. We implement a *SentimentAnalysis* function on *OpenFaaS*[59] and a *Sort* function on Alibaba Function Compute [16] with increasing concurrent users. As illustrated in the Fig.1, figure(a) demonstrates that function performance deteriorates with concurrent requests, with the average latency increased by 10.1× when the concurrent users increased from 3 to 30 on existing instances.

Fig.1(b) shows the performance of whether to apply concurrency limit parameter under different concurrent users. We have two key findings. First, applying concurrency parameter helps to achieve better performance under the same container-level hardware resources configuration (2 cores and 2GB memory for each replica), it reduces the average and the P95 end-to-end latency by 26.65% and 42.77%, respectively, while increasing throughput by 19.35% compared to not having concurrency control due to the rational setting of parallelism degree. Second, when facing autoscaling of dynamic workloads, we should take concurrency into full consideration as it is crucial for maintaining the stability of function performance and especially reducing tail latency.

However, tuning hybrid resources configuration including hardware resources and concurrency for FaaS applications can be complex due to (i) resource knobs interact with each other (e.g., changing CPU allocation can cause optimal concurrency setting change), (ii) there are complex performance dependencies between functions within workflows and an (iii) exponential growth in configuration space. A recent work[45] decouples hardware scaling and concurrency to bypass the issue, but it results in sub-optimal solution, because the end-to-end performance is mutually influenced by both hardware resources and concurrency setting in the serverless environment, which shown in Fig.3(a). Therefore, we aim to jointly optimize hardware resources and concurrency configuration to achieve optimal results.

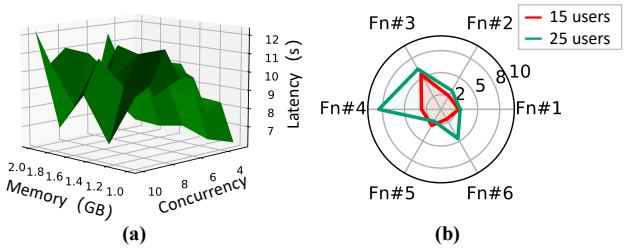


Figure 3: (a) shows the latency on different hardware resource and concurrency in *Sequential* workflow. (b) shows the performance change rate of the tested functions caused by resources reconfiguration of other functions in *Parallel* workflow under different concurrent users.

Motivation 2: A holistic view is necessary for maintaining end-to-end QoS. Many resource configuration tuning methods [1, 8, 90], which optimize single functions in the application independently, greedily seek QoS and cost trade-off of individual functions. Although configuring computing resources for a single serverless function in a narrow configuration space seems simple (e.g., 15 trails for BO to find optimal configurations in work[8]), but it can readily result in QoS violations because they neglect the end-to-end performance of the overall workflows. To support this point, we apply three methods to configure resources for a *Sequential* ML workflow consisting of five functions. (a) Individual tuning that optimizes the resource allocation of each function independently, the goal is to minimize the individual function resource cost while meeting the QoS requirements of each function. (b) Sequential tuning along the function workflow, with the goal of minimizing the individual function resource cost while meeting the QoS requirements of the overall workflow. (c) Holistic optimization generates the resource configurations for all functions, with the goal of optimizing the overall cost while meeting the end-to-end QoS of workflows. Fig.2's right table details tuning policy of cost and QoS, and whether it meets the requirements of cost efficiency and QoS maintaining, E2E for end-to-end view and S represents only considering single function once in the table.

Fig.2 shows that for individual tuning, each function is independently optimized in a greedy manner to pursue its own QoS-cost trade-off, tending to allocate fewer resources for cost-efficiency. While the QoS of a single function is satisfied, there is no assurance of aligning the upstream and downstream processing capabilities, resulting in an end-to-end latency that exceeds the cumulative latency of each function. Furthermore, selecting a QoS threshold at each stage is a major challenge, and most production services do not have per-stage targets[61, 96]. Conversely, sequential tuning defines QoS threshold based on end-to-end latency. In order to maintain the overall workflow's QoS, the functions in the later stages of the workflow prefer more resources to maintain QoS. Method (c) takes a holistic view to ensure end-to-end QoS and provides cost-efficient resources, automatically allocates appropriate resources to each stage, saving 29.34% of resource costs compared to sequential tuning.

Motivation 3: The degree of performance dependencies between functions varies. The resource configuration change of a function will affect the performance of adjacent functions in a serverless application, which we refer to as performance dependency. This situation occurs when functions of the next stages are under-provisioned to handle the invocations from the previous stages. To demonstrate this, we keep the resource configuration of the tested functions and then randomly reconfigure other functions 15 times under a *Parallel* ML workflow with 5 stages. Subsequently, we calculate the performance change rate (PCR) as the ratio of the maximum and minimum values of P75 latency observed in the tested functions among 15 runs. As shown in Fig.3(b), the resources reconfiguration of other functions in the application results in a large value of PCR of tested functions, ranging from 1.5× to 8.5× under different concurrent requests. This finding validates that resources reconfiguration of functions within a workflow can greatly influence other functions under concurrent loads, and the degree of performance dependencies varies across different functions.

The degree of performance dependencies among functions is influenced by various factors, including the characteristics of functions, and the invocation patterns between two functions in tightly-coupled applications is also an important consideration. However, a previous approache SIMPPO [67] assumes equal dependencies between functions, thus failing to capture the varied interdependencies. To address this limitation, we employ the *attention mechanism* to precisely capture the varying dependencies between functions, enabling better resource configuration.

3 FAASCONF OVERVIEW

Optimal Configuration Tuning On Hybrid Resources. We formalize a hybrid resource configuration problem, which jointly optimizes hardware resources and concurrency configuration. Consider a serverless workflow consisting of n functions $F = \{f_1, f_2, ..., f_n\}$ with multiple concurrent invocations, and the holistic resource configuration $C = \{C^1, C^2, ..., C^n\}$ consisting of all functions in the workflows F, the hybrid resources for function f_j is $C^j = \{C^j_{cpu}, C^j_{mem}, C^j_{instance}, C^j_{concurrency}\}$, include CPU, memory, number of function instances, and concurrency setting, respectively. We want to find the holistic optimal hybrid resource configuration C^* of workflow that minimizes the overall resource cost (y) while performance metrics (T) satisfying end-to-end QoS targets λ:

$$C^* = arg\min_{C} y(C),\ subject\ \ to\ \ T(C) < \lambda. \tag{1}$$

The resource cost is linearly correlated to CPU, memory, instance numbers and execution time, consistent with cost models in many production serverless platforms and prior works [8, 25, 27, 39, 96]:

$$y(C) = \sum_{j=1}^{N} ExecTime(\alpha C^j_{cpu} + \beta C^j_{mem}) * C^j_{instance}. \tag{2}$$

In our study, α and β is setting according to $\$0.173/vCPU * hour$ and $\$0.0123/GB * hour$ from *pay as you go* premium plan in Azure Functions[26].

Overview. We depict the overall architecture of FaaSConf in Fig.4. The core idea of FaaSConf is to apply *attention mean-filed* MARL to learn hybrid resource configuring policy for serverless workflows under concurrent and dynamic workloads. The hybrid resources configuration tuning system consists of the FaaS environment, Controller, Data Store, *attention mean-field* Multi-Agent

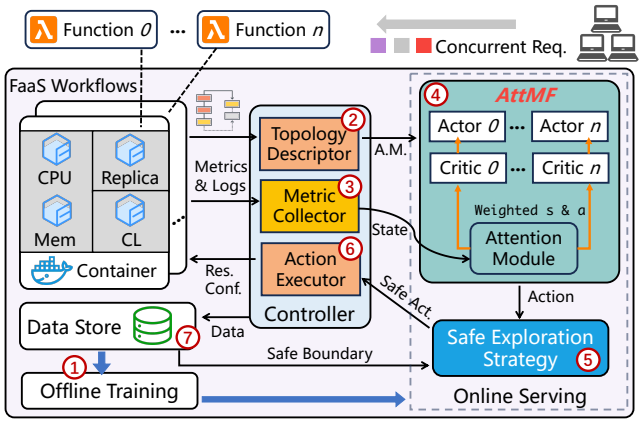


Figure 4: The architecture of FaaSConf.

Reinforcement Learning (AttMF) and Safe Exploration Strategy (SES).

Initially, we employ historical data from Data Store for offline training of the AttMF model ①. Once the online serving phase begins, the user-provided workflow topology is preloaded as adjacency matrix by Topology Descriptor into AttMF for identifying function dependencies ②. FaaSConf generates the optimal resources configuration based on the runtime states of the workflow in an iterative manner. The Metric Collector gathers runtime data such as resources utilization metrics, workloads, performance indicators, and latencies as the states of serverless system for AttMF ③. Within our AttMF, each agent manages hybrid resources configuration of a function, which including vertical resources (e.g., CPU and memory), horizontal resource (e.g. number of replicas), and concurrency. Agents collaborate together to maximize global reward, taking into account interactions through the weighted states and actions generated by the Attention Module ④. Original actions output by AttMF are fed into SES for further optimization ⑤, the role of the SES is to allow agents to explore within predefined safe boundaries by resources price, ensuring QoS while minimizing resource cost. These boundaries are autonomously generated based on historical data. Finally, the safe actions from SES are translated into hybrid resources configuration by the Action Executor and it schedules resources for FaaS workflows through updating YAML configuration files and the platform's command-line interface (CLI) ⑥.All hostorical data are stored in the Data Store for model training and safe boundaries analysis for SES ⑦. This iterative process continues until either the search budget is exhausted or an optimal action is found.

4 DEISGN OF FAASCONF

4.1 MARL-based Approach

Why MARL? We prefer MARL in resource configuring for large-scaled serverless workflows because MARL has superior feature representation and decision-making capabilities for high-dimensional and mutual-interactional environment of workflows compared to traditional ML based, single-RL based and heuristic based methods. In addition, it offers a remarkable adaptability that can transfer knowledge across dynamic workloads. We introduce some key components in MARL below.

Table 2: State-Action space of per agent in MARL.

State Space S_t
Resource Utilization($RU(t)$), Function Performance($FP(t)$), Concurrent Requests($CR(t)$), Resource Limits ($RLT(t)$), Number of Instances ($NI(t)$), Concurrency Limits ($CL(t)$)
Action Space A_t
Vertical Conf: Resource Limits ($RLT(t)$), Horizontal Conf: Number of Instances ($NI(t)$), Concurrency Conf: Concurrency Limits ($CL(t)$)

Action Space. Based on the analysis presented above, it is necessary to optimize the function's hybrid resources, as shown in the Tab.2, the vertical configuration action represents the scaling of the CPU and memory limits of a function container $RLT(t)$, while the horizontal action is the scaling of the number of function instances $NI(t)$. The concurrency limit $CL(t)$ is the max capacity of requests per instance.

State Space. We consider the telemetry data from system and resources configuration as the primary state for the MARL agents, as they provide a comprehensive representation of the system's information and can be easily obtained in real-time. Specifically, we collect the following data for each function at each time step: Resource Utilization $RU(t)$ includes CPU, memory and network. The function performance $FP(t)$ is the tail latency of a function executing multiple requests, Beside, concurrent requests $CR(t)$ and current resource configurations $RLT(t)$, $NI(t)$, $CL(t)$ are also kept as part of state.

Reward Function. Our goal is to learn a set of policies that result in fewer QoS violations while minimizing the resource cost $y(C)$ in Eq.2. Therefore, the reward is : $R_t = \alpha y(C) + \beta$. Where α is a negative regulatory factor that regulates the reward. If an action successfully satisfies the QoS, a positive β will be awarded to the agents, otherwise $\beta = 0$. We adopt sharing reward mechanism to encourage cooperative behavior and incentivize the collective maximization of rewards.

Policy Learning. We implement Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [47] to optimize hybrid resources allocation policies in serverless environment. MADDPG adopts the framework of centralized training with decentralized execution (CTDE). The global state $s = (s_1, \ldots, s_n)$ consists of local state of each agent and the joint action $a = (a_1, \ldots, a_n)$ consists of local action of each agent and is produced by jointly policies $\pi = \{\pi_{\theta^1}, \pi_{\theta^2}, ..., \pi_{\theta^n}\}$. It designs centralized critics and decentralized actors, where $Q_i(s_1, \ldots, s_n, a_1, \ldots, a_n)$ is the centralized action-value function that takes the states and actions of all agents as input to obtain the information of interactions between agents, and outputs a Q-value for agent i. In centralized training, Temporary difference (TD) algorithm is used to train every centralized critic networks, sampling joint tuples (s, a, R, s') of all agents from memory pool and the parameters of Q_i are updated with gradient descent to minimize the loss function ℓ:

$$\begin{aligned} \ell(\theta_i) &= \mathbb{E}\left[\left(Q_i^{\pi}(s_1, \ldots, s_n, a_1, \ldots, a_n) - y\right)^2\right], \\ y &= r_i + \gamma Q_i^{\pi'}(s_1', \ldots, s_n', a_1', \ldots, a_n')\Big|_{a_i' = \pi_i'(s_i)}. \end{aligned} \tag{3}$$

where π_i' is the target policy network, according to the feedback of centralized critic, actor updates policy with policy gradient:

$$\nabla_{\theta_i} J(\pi_i) = \mathbb{E}\left[\nabla_{\theta_i}\pi_i(a_i \mid s_i)\nabla_{a_i} Q_i^{\pi}(s_1,\ldots,s_n,a_1,\ldots,a_n)\big|_{a_i=\boldsymbol{\pi}_i(s_i)}\right]. \tag{4}$$

It is evident that as the number of agents increases, the concatenated state and action vector will have an extremely high dimensionality, which complicates the training process. To address this challenge, we turn to *attention mean-field* technique to enhance the effectiveness of MADDPG.

4.2 Attention Mean-Field MARL(AttMF)

Mean-Field for MARL. To effectively train agents and alleviate the curse of dimensionality introduced by increasing number of agents, *mean-field* (MF) techniques is proposed for MARL [87]. The key idea of *mean-field* MARL is to represent agents' interactions through an average field that captures the distribution of states and actions in the population. Under the *mean-field* approximation, instead of concatenating the global action vectors, the behavior of adjacent functions is modeled as their mean value of actions $\bar{a}_j$. the value function approximation of agent j is:

$$Q(s,a) \sim Q\left(s, a_j, \bar{a}_j\right), \quad \bar{a}_j = \frac{1}{N^j}\sum_{k\in N^j} a_k. \tag{5}$$

where N^j denotes the number of neighboring agents of agent j. This approach reduces the computational complexity of center training in CTDE, but does not lose much optimality in multi-agent scenarios, and the approximation error decreases as the number of agents increases, which is proved by existing work [53].

Weighted Mean-Field. However, the *mean-field* approach presumes equal influence among all agents, thus failing to discern their relative importance. As shown in our motivation 3, the performance dependency varies between functions. To address this issue, the accuracy of MF approximation can be enhanced by employing a weighted averaging, which captures different performance dependencies between functions. Moreover, we also reduce the state space with similar method in Eq.5 and assign neighboring agents with different weights. The value function approximation in weight MF is :

$$Q(s,a) \sim Q\left(s_j, \tilde{s}_j, a_j, \tilde{a}_j\right), \tag{6}$$

$$\tilde{a}_j = \frac{1}{W_j}\sum_{k\in N^j} w^{jk} a_k, \tilde{s}_j = \frac{1}{W_j}\sum_{k\in N^j} w^{jk} s_k, W_j = \sum_{k\in N^j} w^{jk}, \tag{7}$$

where w^{jk} is the weight between agent j and k. Similarly, the input of policy π incorporates both its own observation s_j and the weighted average observation of neighboring agents $\hat{s}_j$ to obtain the observations of other functions to effectively execute the actor-critic training process.

$$\pi(s) \sim \pi\left(s_j, \hat{s}_j\right), \quad \hat{s}_j = \frac{1}{U_j}\sum_{k\in N^j} u^{jk} s_k, \quad U_j = \sum_{k\in N^j} u^{jk}, \tag{8}$$

where u^{jk} is the weight between agent j and k from the perspective of the policy network. To assign different weight of agents, we adopt *attention mechanism* to automatically learn them.

Learning Attention Weight. We want agents to focus on those functions that have a greater impact on their performance when making decisions, rather than blindly believing that the performance impacts between functions are the same. Inspired by prior work[34], we adopt *attention mechanism* [79], which is widely used in NLP fields. The *attention mechanism* automatically explores the potential impact relationships between functions by learning the similarity between functions state vectors, which contain rich information about workloads, resource configurations, function performances and resources utilization, thereby achieving better collaboration and avoiding decision-making conflicts. In order to obtain the weights w^{jk} and u^{jk} that reflect the interaction strength between functions in a serverless application, we use a variant of *attention mechanism* that consider application *topology* to automatically learn them.

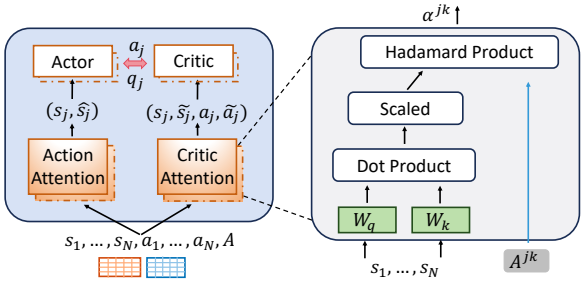


Figure 5: Implementation details of AttMF.

Fig.5 shows the detail components of AttMF. We assign a separate attention module for actor and critic of each agent with learnable parameters Query W_q and Key W_k. Consider the topology of a serverless application, in which each function represents a node and the invocation relationships between functions form edges, thereby creating an adjacency matrix A. In each time step, The attention weight α^{jk} between function j and k compares the state vectors s_j and s_k, using the Query-Key system and subsequently multiples the corresponding elements A^{jk} of the adjacency matrix at their respective positions (Hadamard product) to extract adjacency structure information:

$$\alpha^{jk} \propto exp(s_j W_q W_k^T s_k^T) A^{jk}, \tag{9}$$

where matrix W_q transforms s_j into a query and matrix W_k transforms s_k into a key, the matching is then scaled to prevent vanishing gradients. The weight w^{jk} and u^{jk} are calculated by critic attention and actor attention module, respectively.

Ultimately, we calculate the attention MF vector in Eq.7 and Eq.8, which represents the weighted state and action vectors of the neighboring functions. These vectors are used as inputs for critic and actor, respectively, and the attention modules are trained along with the critic and actor. Our AttMF algorithm enable the agents to consider the performance influence relationships and prioritize functions that significantly impact their performance for making better decisions.

4.3 Safe Exploration Strategy

The online configuration optimization process exploits the performance of workflows in production to make decisions, so it is necessary to reduce performance degradation and QoS violations caused by poor configurations. The above technologies enable FaaSConf to achieve better performance and unearth cost-efficient resource

configurations, yet it cannot guarantee that the online resource configuration optimization consistently meets the QoS requirements because the inherent noise and uncertainty of FaaS cloud platforms mislead the algorithm into making bad decisions [11, 96]. The purpose of SES is to allow the agent to explore within the safety price boundaries that derived from historical data. We hold opinion that resources price can reflect the amount of resources provisioning, which can infer the performance. i.e., a recommended resource configuration with a low price may fail to meet QoS. Conversely, an excessively priced configuration is likely not optimal. Consequently, during the online serving stage of FaaSConf, sub-optimal actions can be filtered out through safe boundary analysis by resources price before action execution.

Safe Boundary Analysis. We implement SES by employing a popular statistical approach k-σ rule [64] to capture a representative range of optimal resources and determine whether a recommended action is in sub-optimal status. It is essential for the agents to concentrate on high-performance areas that both guarantee QoS and provide cost-effective resources during the online serving phase. So we use history resource configurations that meet QoS with the most cost-efficient top 20% resource prices to form a set τ, and then compute the mean total price $P(\tau)$ and standard deviation $std(P(\tau))$ to establish a safe boundary using $P(\tau) \pm N \times std(P(\tau))$, denoted as $[lb, ub]$ (i.e., lower bound and upper bound), this range is likely to contain optimal solutions. We experimentally determined that $N = 2$.

$$A = \begin{cases} A_c, \ y(A) \in [lb, ub] \\ A_{safe} + \epsilon, \ y(A) \notin [lb, ub] \end{cases} \tag{10}$$

As described in Eq.10, in each iteration, MARL output a current action A_c, we then compute the resource configuration price y using Eq.2. If the resource price falls within the defined range $[lb, ub]$, we recommend this configuration and observe the performance. If it falls outside this range, algorithm selects a safe action A_{safe} randomly from the history and introduces a Gaussian random noise ϵ for exploration. By applying SES, FaaSConf can select the safe and cost-efficient resource configurations and reduce QoS violations in online optimization scenarios. We record the AppName, Workload, lb and ub as JSON files in the Data Store.

Boundary Adaptation. We further propose two practical policies to improve it's adaptability and effectiveness. First, due to varied performance, we regularly evaluate the confidence of the safe range. If the number of QoS violations reaches a threshold, we will re-determine the range based on the latest data. Second, during online resource optimization with dynamic and unseen workloads, records with boundary information may be missing, we retrieve the record with the similar workloads and set safety configuration boundaries to provide QoS assurance.

5 EXPERIMENTAL EVALUATION

In this section, we evaluate FaaSConf to answer three questions.

- **RQ1:** How effective is FaaSConf in finding optimal resource configuration while ensuring QoS?
- **RQ2:** How does FaaSConf adapts to dynamic workload?
- **RQ3:** What is the contribution of each design module?

5.1 Experimental Settings

5.1.1 *Experiment Setup.*

Hardware. We construct a distributed testbed that contains 8 virtual machines (VMs). Each VM has a 8-core 2.40GHz CPU, 16GB memory and runs with Ubuntu 18.04 OS. We guarantee that all the VMs are in the same local area network to reduce network jitters.

Serverless Platform. We evalute FaaSConf on *OpenFaaS* [59], a widely-used open-source serverless platform. *OpenFaaS* is deployed on top of *Kubernetes*[38], which acts as the principal container orchestrator. *OpenFaaS* provides an API gateway for invoking functions, serving as an external route to these functions. Function scaling decisions are facilitated by the AlertManager component, which scales functions by reading RPS metric from *Prometheus*[63], which is an open-source system monitoring toolkit and subsequently alerting the gateway. We disbale the AlertManager and use FaaSConf to carry out resource provision.

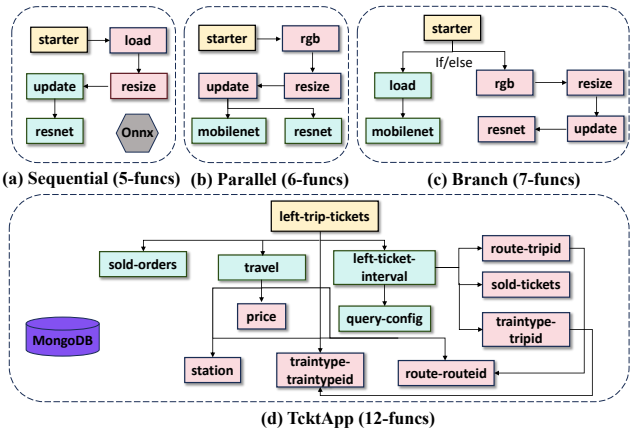


Figure 6: Architectures of used benchmark applications.

Benchmark Applications. The evaluation of FaaSConf utilizes two types of end-to-end interactive real-world benchmarks: one is machine learning (ML) workflows composed of multiple-functions, and the other is a popular web service, as shown in Fig.6. We employ ML workflows from prior work [80]. The individual serverless functions are adapted from publicly available examples that utilize TensorFlow with Azure Functions [25] and the models are sourced from the Onnx Model Zoo [58]. We have implemented three types of serverless workflows: *Sequential*, and *Parallel*, *Branch* with 5, 6 and 7 stateless functions, respectively.

Additionally, we employ an online serverless application with 12 functions: search ticket (*TcktApp*) from TrainTicket [30], a comprehensive microservices suite. *TcktApp*'s role is to retrieve information about remaining tickets between two locations, with the tickets information being stored in MongoDB[56]. These functions are more lightweight in logic and tightly-coupled than ML functions.

5.1.2 *Configurations and workloads.*

Resource Configuration Space. In our configuration space, we decouple CPU and memory to provide flexible and cost-efficient resource choices according to prior work [8]. In ML workflows, the memory configuration is set to a continuous value within the range of 256 to 2024 MB, while the CPU is allocated a continuous value

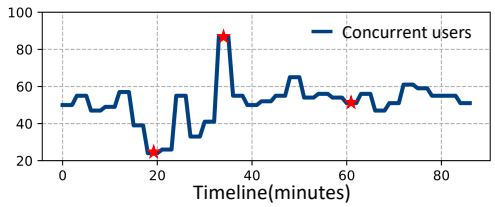


Figure 7: Dynamic workloads from Azure Functions trace. The pentagrams are utilized to represent relative low, middle and high workload in Sec.5.2.2.

between 250 and 2000 millicores. In web service, the maximum values for CPU and memory are set to 1000 and 1024, respectively. We adjust the function replicas number from 1 to 8 and establish a concurrency limit ranging from 1 to 10, which is specified in the function configuration file through the *max_inflight* field in *OpenFaaS*[59]. These settings are designed to adapt optimally to our experimental testbed and maximize the performance of cluster resources.

Load Generator. We conducted experiments under both static and dynamic loads which the dynamic workloads is from a real-word invocation traces from Azure Functions[74]. We use Locust [46] load generator to emulate users sending concurrent requests with 10 RPS mean arrival rate until the max concurrent users is reached. We set 30 seconds to be the length of one time window to continuously send concurrent requests according to [88]. The workload generator and functions are never physically co-located on a server.

5.1.3 Points of Comparisons. We compare FaaSConf with the following approaches.

- **Random Search (RS)** [6]randomly selects resource configurations from the search space. We treat RS as the basic approaches without any model.
- **Firm** [66] is a RL-based state-of-the-art method for multi-resources management. It uses Deep Deterministic Policy Gradient (DDPG) to dynamically adjust the resources of functions during runtime.
- **RAMBO** [40] holds a holistic view on the entire workflows and employs multi-objective Bayesian Optimization (BO) to allocate resources for applications, thereby achieving performance-cost tradeoff. RAMBO is coded based on BayesOpt [54].
- **Aquatope** [96] is a QoS- and uncertainty-aware resource management framework for serverless workflows, employs a customized BO approach. This approach integrates noise Gaussian Process (GP) models and noisy expected improvement (NEI) acquisition function to capture the uncertainty of FaaS platforms, while utilizing an independent GP model for predicting QoS. Aquatope is implemented based on BoTorch [4].

We first warm-up them through training samples, and then iterate 15 rounds to get results of QoS and costs. Our QoS targets are set as average latency and throughput in one time window. We linearly scale the resource costs in Eq.2 for better showcase. To offset performance difference under same resource configurations, our experimental results are the average of three repeated runs.

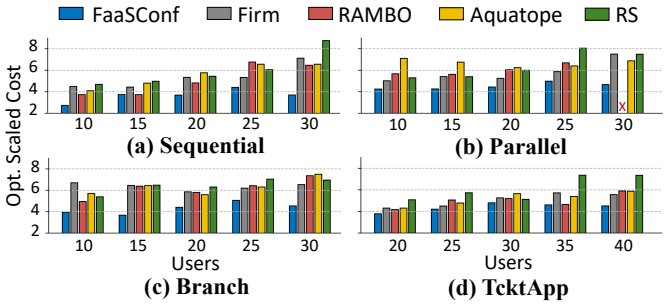


Figure 8: The optimal scaled cost under four serverless workflows with five different concurrent users (lower is better). 'X' means no feasible solution is found.

5.2 Results and Analysis

5.2.1 Effectiveness Validation (RQ1). We now evaluate the FaaSConf's ability to find cost-efficient resource configurations while meeting QoS. Fig.8 shows the optimal scaled cost results of different serverless workflows within the sampling budgets. For each workflow, we run 5 experiments with an increasing number of concurrent users. Under the same sampling budget for resource exploration, FaaSConf demonstrates superior performance compared to all other methods across the examined applications, and significantly reduce resource costs. Compared with Firm, RAMBO, Aquatope and RS, FasSConf can cut the resource cost by average 24%, 23%, 28% and 31%, respectively. Specifically, RAMBO[40] aims to strike a balance between workflow performance and cost, leading to over-provisioning of resources to meet QoS requirements, often getting trapped in local optima. Due to the dependencies of functions in workflows in the scenario of high concurrent requests, Aquatope[96] cannot accurately model the complex nonlinear relationship between resources and performance using Gaussian Process (GP) model. Firm[66] also results in sub-optimal cost because it solely optimizes the scaling-needed functions without considering the impact on the performance of other functions. Random search randomly selects a set of configurations to explore across all functions without leveraging knowledge from previous trials. Furthermore, neglecting to optimize concurrency of baseline methods can readily result in QoS violations under concurrent workloads, as requests indefinitely contend for resources. e.g., RAMBO cannot even find resources that meet QoS requirements within the sampling budgets under *Parallel*-30 users. In contrast, FaaSConf uses AttMF to capture the dynamic dependencies among functions, takes into account the performance implications of neighboring functions during decision-making, and improves resource efficiency.

We observe an interesting result of *TcktApp*, FaaSConf's performance benefits are less pronounced. Due to the application being implemented in a synchronous mode using *Java* language, requests are processed synchronously, rendering concurrency configuration ineffective in this scenario, which bares higher risk of request accumulation and increasing response time compared with concurrency pattern[19, 20, 65]. Even so, FaaSConf can still reduce the resource consumption of other baselines by average 16.6% and up to 38.8%. The above results prove that FaaSConf's ability for reducing resource consumption while meeting QoS for serverless workflows under concurrent requests.

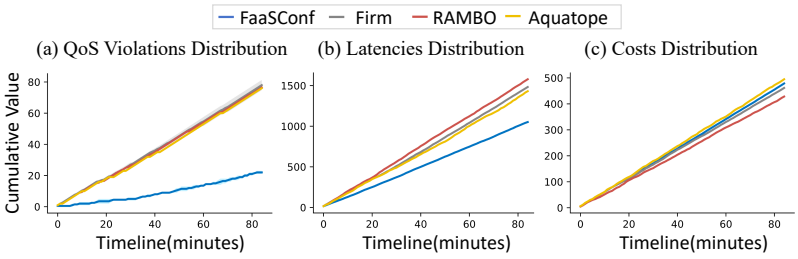


Figure 9: Performance comparison on QoS violations, latencies and costs under dynamic workloads during 85 minutes.

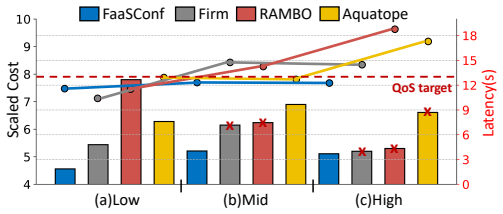


Figure 10: The scaled cost and latency under relative low, middle and high workload that specified in Fig.7 and 'X' means QoS violation.

5.2.2 Adaptability Analysis (RQ2). In real production, the workload always changes over time. Fig.7 shows a period of real-world serverless invocation trace from Azure Functions [74] in 85 minutes, emulating typical characteristics of serverless workloads such as stable, continuous fluctuations and burstly increases.

We leverage data under five workloads from Sec.5.2.1 experiment on *Parallel* workflow to train the model and investigate whether different approaches can rapidly adapt to dynamic loads. Fig.9 shows the cumulative distribution of QoS violations, latencies and scaled costs over time. We have two observations. First, FaaSConf quickly responds to changing loads, ensuring over 74.12% QoS compliance and reducing average 3.48× QoS violations compared to other methods. Second, FaaSConf can always find nearly optimal resource configurations and reduce 29.1%, 33.39%, 26.53% cumulative latencies than Firm[66], RAMBO[40] and Aquatope[96], with only 8.48% more resource costs than oracle, which is obtained by exhaustive offline search. This confirms that FaaSConf can find better resource configurations than other methods even with dynamic load changes. The reason is that our method considers runtime environment characteristics and workload changes, so the learned policy has a good transferability. In addition, SES helps agents make decisions within safety resource boundaries to reduce QoS violations. However, other methods unable to adapt to dynamic loads and make a lot of wrong decisions. The reason is that BO-based methods (e.g., RAMBO and Aquatope) focus on finding local optimal solutions in the offline scenarios. However, the changing environment causes inaccurate Gaussian Process modeling and requires a large amount of data for retraining to mitigate performance degradation, which is not practical in online scenarios. Firm also find it difficult to quickly update its policy under new environment in a short time. This demonstrates FaaSConf's adaptability to quickly recommend cost-efficient resource configurations in response to dynamic workloads, ensuring stable performance of FaaS applications.

In order to further facilitate the comparison of the performance of different methods in dynamic workloads, we select relative low, middle and high workload marked on Fig.7 for demonstration. Fig.10 shows the scaled cost and latency under three real-time workloads. It can be seen that FaaSConf consistently outperforms other methods, with an average improvement of 11.49%, 23.11%, 24.86% in reducing resource costs, and 8.26%, 18.92%, 16.01% in reducing end-to-end latency compared with Firm[66], RAMBO[40] and Aquatope[96], while always satisfying QoS. The other methods all caused QoS violations under different concurrent workloads beacause they can not scale out resource according dynamic workloads. The above results prove FaaSConf' s ability to reduce both QoS violations and costs in production environments where online continuous resource tuning is required.

5.2.3 Ablation Study (RQ3). In this subsection, we validate the effectiveness of each design in FaaSConf.

Contribution of Hybrid Resource Optimization. As shown in Motivation 1, concurrency configuration significantly impacts the performance of FaaS applications. Configuring a large concurrency limit can lead to resource contentions. Conversely, a lower one can lead to under-utilization of resources. We compare the optimal hybrid configuration found by FaaSConf with it's hardware resources configuration(i.e., CPU, memory and replicas) without concurrency configuration for each ML workflows. Fig. 11 shows that the hybrid resources achieves an average 25.58% and up to 34.76% end-to-end latency reduction and an average throughput improvement of 1.37× and up to 1.79× in *Sequential*, *Parallel* and *Branch* workflows. This result demonstrates the effectiveness of hybrid resource configuration tuning that considers both hardware resources and concurrency in enhancing the performance of FaaSConf.

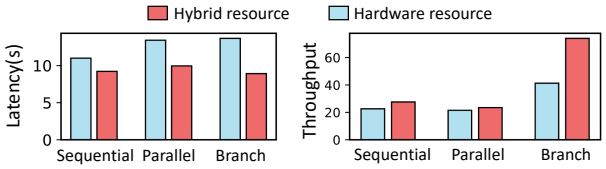


Figure 11: Latency and throughput of hybrid resources configuration and hardware resources configuration for ML workflows.

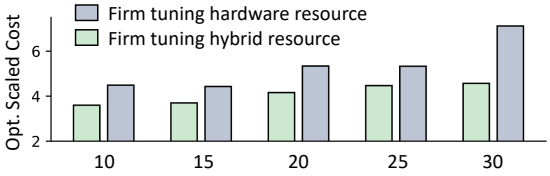


Figure 12: The scaled cost of optimal resources configuration of tuning hybrid resources and hardware resources in *Sequential* workflow under five concurrent users.

To further illustrate that optimizing hybrid configurations can likewise benefit other approaches, we apply Firm[66] to separately optimize hybrid resources and hardware resources in *Sequential* workflow. As shown in Fig. 12, under five different levels of concurrent user workloads, the optimal configuration derived from optimizing hybrid resources consistently exhibited lower costs than

solely optimizing hardware resources, with 19.89%, 16.48%, 22.1%, 16.14%, 35.81% resource costs reductions respectively and 22.08% on average. The results imply that optimizing hybrid resources can reduce resource costs while ensuring QoS and should be considered in production to ensure the performance stability of the workflows.

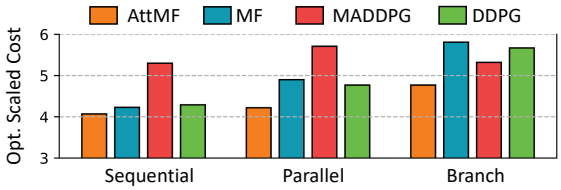


Figure 13: The scaled cost of optimal resources configuration found by different RL algorithms in *Sequential*, *Parallel* and *Branch* workflows(lower is better).

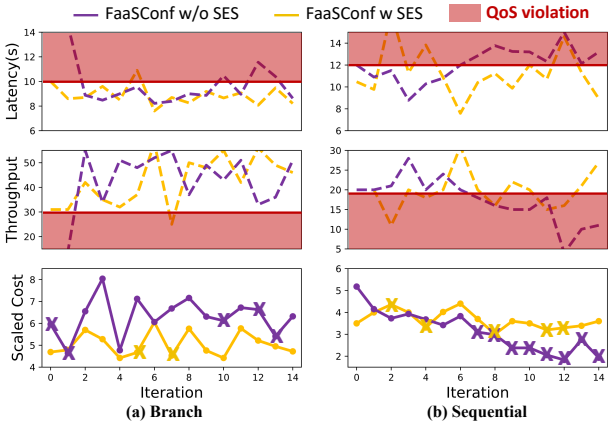


Figure 14: Impact of *safe exploration strategy* under two workflows. The dark red rectangle represents the area of QoS violations and 'X' means unavailable configuration.

Contribution of Attention Mean-Field. To study the effectiveness of attention mean-field mechanism, we compare our AttMF with (i) Mean-Field MARL(MF)[87] used in SIMPPO [67], (ii) vanilla Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [47] and (iii) DDPG algorithm[43] used in Firm[66]. Fig. 13 shows the most cost-efficient resource configuration during optimization process. AttMF achieve the best performance by reducing resource cost by 11.85%, 19.88% and 10.84% on average and up to 17.9%, 26.09% and 15.87% compared with MF, MADDPG and DDPG while satisfying QoS. The effectiveness of our AttMF stems from the utilization of *attention mechanism* to capture varying performance dependencies among functions, which allows for allocating different weights to adjacent functions, thereby facilitating more effective information sharing. On the other hand, As represented in Eq.3 and Eq.4, MADDPG and vanilla DDPG with global state-action space, suffering from dimension explosion, hindering effective policy learning. Although MF reduced computational complexity by using mean value of state vectors, it fails to reflect the true interaction among functions. In summary, AttMF not only considers the collaboration of resource allocation based on the varied dependencies among functions but also reduces difficulty of policy learning, thus generating better resource allocation decisions.

Contribution of Safe Exploration Strategy. An online optimization process may evaluate several under-performing resource configurations, however, such evaluations carry the risks of violating QoS. To reduce QoS violations, FaaSConf is augmented with a safe exploration strategy(SES). To demonstrate the effectiveness, we present the quality of candidate recommended points during an online resource optimization process. Fig.14 shows the end-to-end latency, throughput, and scaled resource costs of *branch* and *Sequential* over 15 resource configuration tuning iterations. We have two observations. First, the use of SES technique reduces 60% and 37.5% QoS violations during the online optimization period. It imples that SES can filter unsafe resource configurations and allow MARL agents to explore in the optimal solution area. Second, the best configuration is discovered to be 7.34% and 0.88% more cost-efficient compared to the baseline without using SES for *Branch* and *Sequential* workflows. This is because SES migrates the bad decisions to safe resource configurations learned form history. The results confirm that leveraging SES not only enhances QoS assurance of FaaSConf but also bring about performance improvements.

6 RELATED WORK

The commercial FaaS platforms provide users with rich configuration parameters to meet the needs of different application scenarios. These configuration parameters cover CPU, memory, I/O resources, regions, instance families, concurrency limit, timeout settings, etc.. Automatically resource configuration tuning is an important aspect of serverless systems to achieve various goals.

Resource Configuration on Serverless Computing. There are serveral studies work on optimizing resource cost on serverless platforms. COSE [1] uses the Bayesian Optimization (BO) to minimize the cost of execution by optimizing memory configuration. SLAM [71] estimates the execution time for the applications at different memory configurations and determines the optimal memory configuration. Aquatope [96] first predict future workloads to warm start the functions and then optimize resource configuration using noisy BO. Astra [36] and CE-scaling [83] optimizes configurations for data-intensive analytic and ML applications in serverless platforms, respectively. Fifer [31] and Kraken [7] resort to the concept of slack to reduce the containers. FaaSDeliver [90] finds optimal function delivery policy (FDP) for functions, including platform selection and resource allocation in a heterogeneous computing continuum. StepConf [82] dynamically configure memory and inter- and intra-function parallelism degrees on AWS Lambda, but these techniques do not consider the interaction between functions. A similar work to ours is SIMPPO [67] that uses MARL in resource management on multi-tenant environment. However, there is relative little researches on hybrid resources configuration tuning that consider concurrency to improve resource efficiency in serverless environment.

Resource Configuration in the Cloud. The field of resource management and configuration in cloud computing is of great significance, encompasses a wide array of research. VCONF [69], CherryPick [2] and OPTIMUSCLOUD[50] employ black-box optimization methods to determine optimal cloud configurations for wide-ranging applications, including VM type and resource size. Systems such as CLITE [61], PARTIES [13], and OLPART [12] are

designed to manage resources for the co-location of multiple interactive services within data centers. Another hot field is microservices auto-scaling to provision containers for dynamic workload changes [28, 35, 49, 55, 68, 81, 86, 89]. However, these systems fails to take into account the characteristics of serverless workflows.

Configuration Tuning for Software Systems. Configuration tuning is crucial for various software systems, especially in meeting user SLOs. Prior studies can be mainly divided into categories: search-based and learning-based methods. Search-based methods explore the configuration space according to specific rules and measure the configuration settings until finding satisfying results. e.g., random search [6, 57], genetic algorithm[73] and hill climbing algorithm[18] have been applied for finding configurations. Learning-based methods apply a surrogate model to predict the performance and then find the optimal configurations[1, 32, 33, 93], or apply RL to learn optimization policies[10, 21, 94]. However, they cannot effectively address the issue of resource configuration optimization in software runtime.

7 DISCUSSION

Overhead. We collected a total of 31K samples for offline training, each measurement takes around 2 minutes (including updating resource configurations, deploying functions, replaying workload, collecting runtime metrics and scaling containers to zero), so the total time spent on collecting samples is around 43 days. We train the MARL model on NVIDIA A100 GPU, and use the trained model in online test. Our model is a relatively compact structure, incorporating four DNNs with an actor-critic architecture, alongside the attention parameter matrix for calculating attention scores. The average time required to map a state to an action is approximately 1ms, which can be considered negligible.

Limitation. First, FaaSConf is a data-driven approach, faces limitations imposed by both the quality and quantity of the historical data, these restrictions can potentially impact the effectiveness and adaptability. Second, while FaasConf can effectively reduce QoS violations, the concurrency hard limit rejects some requests, if the request success rate continues to declines, it is necessary for FaaSConf to consider scaling up resources in the scenarios where the success rate of requests is a key indicator. Finally, we do not extensively consider cold start, acknowledging its potential yet limited impact in high-concurrency scenarios.

8 CONCLUSION

In this work, we propose FaaSConf, a QoS-aware hybrid resources configuration approach for serverless workflows under concurrent and dynamic workloads. We show that jointly optimizing hardware and concurrency configurations is crucial to meet QoS and cost-efficiency for workflows. The key novelty of FaaSConf is to leverages attention-based MARL to learn resource allocation intelligently, which considers performance interdependencies across functions. Experiments validate the effectiveness and adaptability of FaaSConf, while significantly reducing resource costs and avoiding QoS violations. The source code of FaasConf is available at https://github.com/wiluen/FaaSConf.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their valuable comments and suggestions. This work was supported by the National Natural Science Foundation of China under Grant No. 61902440 and No. 62272001. This work was also supported by the National Natural Science Foundation of China (No. 62272495), the Guangdong Basic and Applied Basic Research Foundation (No.2023B1515020054). The corresponding author is Hui Dou.

REFERENCES

[1] Nabeel Akhtar, Ali Raza, Vatche Ishakian, and Ibrahim Matta. 2020. Cose: Configuring serverless functions using statistical learning. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 129–138.
[2] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. 2017. {CherryPick}: Adaptively unearthing the best cloud configurations for big data analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 469–482.
[3] Aitor Arjona, Pedro García López, Josep Sampé, Aleksander Slominski, and Lionel Villard. 2021. Triggerflow: Trigger-based orchestration of serverless workflows. *Future Generation Computer Systems* 124 (2021), 215–229.
[4] Maximilian Balandat, Brian Karrer, Daniel Jiang, Samuel Daulton, Ben Letham, Andrew G Wilson, and Eytan Bakshy. 2020. BoTorch: A framework for efficient Monte-Carlo Bayesian optimization. *Advances in neural information processing systems* 33 (2020), 21524–21538.
[5] Emmanual N Barron. 2024. *Game theory: an introduction*. John Wiley & Sons.
[6] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of machine learning research* 13, 2 (2012).
[7] Vivek M Bhasi, Jashwant Raj Gunasekaran, Prashanth Thinakaran, Cyan Subhra Mishra, Mahmut Taylan Kandemir, and Chita Das. 2021. Kraken: Adaptive container provisioning for deploying dynamic dags in serverless platforms. In *Proceedings of the ACM Symposium on Cloud Computing*. 153–167.
[8] Muhammad Bilal, Marco Canini, Rodrigo Fonseca, and Rodrigo Rodrigues. 2023. With great freedom comes great opportunity: Rethinking resource allocation for serverless functions. In *Proceedings of the Eighteenth European Conference on Computer Systems*. 381–397.
[9] Rodica Branzei, Dinko Dimitrov, and Stef Tijs. 2008. *Models in cooperative game theory*. Vol. 556. Springer Science & Business Media.
[10] Xiangping Bu, Jia Rao, and Cheng-Zhong Xu. 2009. A reinforcement learning approach to online web systems auto-configuration. In *2009 29th IEEE International Conference on Distributed Computing Systems*. IEEE, 2–11.
[11] Baoqing Cai, Yu Liu, Ce Zhang, Guangyu Zhang, Ke Zhou, Li Liu, Chunhua Li, Bin Cheng, Jie Yang, and Jiashu Xing. 2022. HUNTER: an online cloud database hybrid tuning system for personalized requirements. In *Proceedings of the 2022 International Conference on Management of Data*. 646–659.
[12] Ruobing Chen, Haosen Shi, Yusen Li, Xiaoguang Liu, and Gang Wang. 2023. OLPart: Online Learning based Resource Partitioning for Colocating Multiple Latency-Critical Jobs on Commodity Computers. In *Proceedings of the Eighteenth European Conference on Computer Systems*. 347–364.
[13] Shuang Chen, Christina Delimitrou, and José F Martínez. 2019. Parties: Qos-aware resource partitioning for multiple interactive services. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 107–120.
[14] Apache OpenWhisk Composer. 2024. https://github.com/apache/openwhisk-composer/.
[15] Google Cloud Composer. 2024. https://cloud.google.com/composer.
[16] Alibaba Function Compute. 2024. https://www.alibabacloud.com/zh/product/function-compute.
[17] Nilanjan Daw, Umesh Bellur, and Purushottam Kulkarni. 2020. Xanadu: Mitigating cascading cold starts in serverless function chain deployments. In *Proceedings of the 21st International Middleware Conference*. 356–370.
[18] Xiaoan Ding, Yi Liu, and Depei Qian. 2015. Jellyfish: Online performance tuning with adaptive configuration and elastic container in hadoop yarn. In *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 831–836.
[19] Go Documentation. 2024. https://go.dev/doc/.
[20] Java Documentation. 2024. Concurrency. https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html.
[21] Hui Dou, Yilun Wang, Yiwen Zhang, and Pengfei Chen. 2022. DeepCAT: A Cost-Efficient Online Configuration Auto-Tuning Approach for Big Data Frameworks. In *Proceedings of the 51st International Conference on Parallel Processing*. 1–11.
[22] Simon Eismann, Long Bui, Johannes Grohmann, Cristina Abad, Nikolas Herbst, and Samuel Kounev. 2021. Sizeless: Predicting the optimal size of serverless functions. In *Proceedings of the 22nd International Middleware Conference*. 248–259.

[23] Sadjad Fouladi, Riad S Wahby, Brennan Shacklett, Karthikeyan Vasuki Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. 2017. 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)*. 363–376.

[24] IBM Cloud Function. 2024. https://cloud.ibm.com/functions.

[25] Azure Functions. 2024. https://azure.microsoft.com/en-us/services/functions.

[26] Azure Functions. 2024. Pricing. https://azure.microsoft.com/en-us/pricing/details/functions/.

[27] Google Cloud Functions. 2023. https://cloud.google.com/functions.

[28] Alim Ul Gias, Giuliano Casale, and Murray Woodside. 2019. ATOM: Model-driven autoscaling for microservices. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1994–2004.

[29] Github. 2024. OpenFaaS Watchdog. https://docs.openfaas.com/architecture/watchdog.

[30] Github. 2024. Serverless Train Ticket. https://github.com/FudanSELab/serverless-trainticket.

[31] Jashwant Raj Gunasekaran, Prashanth Thinakaran, Nachiappan C Nachiappan, Mahmut Taylan Kandemir, and Chita R Das. 2020. Fifer: Tackling resource underutilization in the serverless era. In *Proceedings of the 21st International Middleware Conference*. 280–295.

[32] Yijin Guo, Huasong Shan, Shixin Huang, Kai Hwang, Jianping Fan, and Zhibin Yu. 2021. Gml: efficiently auto-tuning flink's configurations via guided machine learning. *IEEE Transactions on Parallel and Distributed Systems* 32, 12 (2021), 2921–2935.

[33] Huong Ha and Hongyu Zhang. 2019. DeepPerf: Performance prediction for configurable software with deep sparse neural network. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1095–1106.

[34] Qianyue Hao, Wenzhen Huang, Tao Feng, Jian Yuan, and Yong Li. 2023. GAT-MF: Graph Attention Mean Field for Very Large Scale Multi-Agent Reinforcement Learning. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 685–697.

[35] Md Rajib Hossen, Mohammad A Islam, and Kishwar Ahmed. 2022. Practical efficient microservice autoscaling with QoS assurance. In *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*. 240–252.

[36] Jananie Jarachanthan, Li Chen, Fei Xu, and Bo Li. 2021. Astra: autonomous serverless analytics with cost-efficiency and QoS-awareness. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 756–765.

[37] Knative. 2024. Knative. https://knative.dev/docs/.

[38] Kubernetes. 2024. Production-grade container orchestration. https://kubernetes.io/.

[39] AWS Lambda. 2024. https://aws.amazon.com/lambda.

[40] Qian Li, Bin Li, Pietro Mercati, Ramesh Illikkal, Charlie Tai, Michael Kishinevsky, and Christos Kozyrakis. 2021. RAMBO: Resource allocation for microservices using Bayesian optimization. *IEEE Computer Architecture Letters* 20, 1 (2021), 46–49.

[41] Yihong Li, Tianyu Zeng, Xiaoxi Zhang, Jingpu Duan, and Chuan Wu. 2023. TapFinger: Task Placement and Fine-Grained Resource Allocation for Edge Machine Learning. In *IEEE INFOCOM*.

[42] Zijun Li, Quan Chen, Shuai Xue, Tao Ma, Yong Yang, Zhuo Song, and Minyi Guo. 2020. Amoeba: Qos-awareness and reduced resource usage of microservices with serverless computing. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 399–408.

[43] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[44] Changyuan Lin and Hamzeh Khazaei. 2020. Modeling and optimization of performance and cost of serverless applications. *IEEE Transactions on Parallel and Distributed Systems* 32, 3 (2020), 615–632.

[45] Jianshu Liu, Shungeng Zhang, and Qingyang Wang. 2023. μConAdapter: Reinforcement Learning-based Fast Concurrency Adaptation for Microservices in Cloud. In *Proceedings of the 2023 ACM Symposium on Cloud Computing*. 427–442.

[46] Locust. 2024. https://locust.io.

[47] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems* 30 (2017).

[48] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. 2021. Characterizing microservice dependency and performance: Alibaba trace analysis. In *Proceedings of the ACM Symposium on Cloud Computing*. 412–426.

[49] Shutian Luo, Huanle Xu, Kejiang Ye, Guoyao Xu, Liping Zhang, Guodong Yang, and Chengzhong Xu. 2022. The power of prediction: Microservice auto scaling via workload learning. In *Proceedings of the 13th Symposium on Cloud Computing*. 355–369.

[50] Ashraf Mahgoub, Alexander Michaelson Medoff, Rakesh Kumar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. 2020. {OPTIMUSCLOUD}: Heterogeneous configuration optimization for distributed databases in the cloud. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. 189–203.

[51] Ashraf Mahgoub, Edgardo Barsallo Yi, Karthick Shankar, Sameh Elnikety, Somali Chaterji, and Saurabh Bagchi. 2022. ORION and the Three Rights: Sizing, Bundling, and Prewarming for Serverless DAGs. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 303–320. https://www.usenix.org/conference/osdi22/presentation/mahgoub

[52] Nima Mahmoudi and Hamzeh Khazaei. 2020. Performance modeling of serverless computing platforms. *IEEE Transactions on Cloud Computing* 10, 4 (2020), 2834–2847.

[53] Weichao Mao, Haoran Qiu, Chen Wang, Hubertus Franke, Zbigniew Kalbarczyk, Ravishankar Iyer, and Tamer Basar. 2022. A mean-field game approach to cloud resource management with function approximation. *Advances in Neural Information Processing Systems* 35 (2022), 36243–36258.

[54] Ruben Martinez-Cantin. 2014. BayesOpt: a Bayesian optimization library for nonlinear optimization, experimental design and bandits. *J. Mach. Learn. Res.* 15, 1 (2014), 3735–3739.

[55] Chunyang Meng, Shijie Song, Haogang Tong, Maolin Pan, and Yang Yu. 2023. DeepScaler: Holistic Autoscaling for Microservices Based on Spatiotemporal GNN with Adaptive Graph Learning. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 53–65.

[56] MongoDB. 2024. https://www.mongodb.com/.

[57] Jeho Oh, Don Batory, Margaret Myers, and Norbert Siegmund. 2017. Finding near-optimal configurations in product lines by random sampling. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 61–71.

[58] Onnx. 2024. https://github.com/onnx/models.

[59] OpenFaaS. 2024. https://www.openfaas.com.

[60] Apache OpenWhisk. 2024. https://openwhisk.apache.org.

[61] Tirthak Patel and Devesh Tiwari. 2020. Clite: Efficient and qos-aware co-location of multiple latency-critical jobs for warehouse scale computers. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 193–206.

[62] Adolfo Perrusquía, Wen Yu, and Xiaoou Li. 2021. Multi-agent reinforcement learning for redundant robot control in task-space. *International Journal of Machine Learning and Cybernetics* 12 (2021), 231–241.

[63] Prometheus. 2024. https://prometheus.io/.

[64] Friedrich Pukelsheim. 1994. The three sigma rule. *The American Statistician* 48, 2 (1994), 88–91.

[65] Python. 2024. Concurrent Execution. https://docs.python.org/3/library/concurrency.html.

[66] Haoran Qiu, Subho S Banerjee, Saurabh Jha, Zbigniew T Kalbarczyk, and Ravishankar K Iyer. 2020. {FIRM}: An intelligent fine-grained resource management framework for {SLO-Oriented} microservices. In *14th USENIX symposium on operating systems design and implementation (OSDI 20)*. 805–825.

[67] Haoran Qiu, Weichao Mao, Archit Patke, Chen Wang, Hubertus Franke, Zbigniew T Kalbarczyk, Tamer Basar, and Ravishankar K Iyer. 2022. SIMPPO: A Scalable and Incremental Online Learning Framework for Serverless Resource Management. In *ACM Symposium on Cloud Computing*.

[68] Haoran Qiu, Weichao Mao, Chen Wang, Hubertus Franke, Alaa Youssef, Zbigniew T Kalbarczyk, Tamer Başar, and Ravishankar K Iyer. 2023. {AWARE}: Automate Workload Autoscaling with Reinforcement Learning in Production Cloud Systems. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. 387–402.

[69] Jia Rao, Xiangping Bu, Cheng-Zhong Xu, Leyi Wang, and George Yin. 2009. VCONF: a reinforcement learning approach to virtual machines auto-configuration. In *Proceedings of the 6th international conference on Autonomic computing*. 137–146.

[70] Fabiana Rossi, Valeria Cardellini, Francesco Lo Presti, and Matteo Nardelli. 2022. Dynamic multi-metric thresholds for scaling applications using reinforcement learning. *IEEE Transactions on Cloud Computing* (2022).

[71] Gor Safaryan, Anshul Jindal, Mohak Chadha, and Michael Gerndt. 2022. SLAM: SLO-aware memory optimization for serverless applications. In *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*. IEEE, 30–39.

[72] Lucia Schuler, Somaya Jamil, and Niklas Kühl. 2021. AI-based resource allocation: Reinforcement learning for adaptive auto-scaling in serverless environments. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 804–811.

[73] Arman Shahbazian, Suhrid Karthik, Yuriy Brun, and Nenad Medvidovic. 2020. eQual: informing early design decisions. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1039–1051.

[74] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *2020 USENIX annual technical conference (USENIX ATC 20)*. 205–218.

[75] Amoghavarsha Suresh, Gagan Somashekar, Anandh Varadarajan, Veerendra Ramesh Kakarla, Hima Upadhyay, and Anshul Gandhi. 2020. Ensure: Efficient

scheduling and autonomous resource management in serverless environments. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. IEEE, 1–10.

[76] John Thorpe, Yifan Qiao, Jonathan Eyolfson, Shen Teng, Guanzhou Hu, Zhihao Jia, Jinliang Wei, Keval Vora, Ravi Netravali, Miryung Kim, et al. 2021. Dorylus: Affordable, scalable, and accurate {GNN} training with distributed {CPU} servers and serverless threads. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. 495–514.

[77] Huangshi Tian, Suyi Li, Ao Wang, Wei Wang, Tianlong Wu, and Haoran Yang. 2022. Owl: Performance-aware scheduling for resource-efficient function-as-a-service cloud. In *Proceedings of the 13th Symposium on Cloud Computing*. 78–93.

[78] Parichehr Vahidinia, Bahar Farahani, and Fereidoon Shams Aliee. 2022. Mitigating cold start problem in serverless computing: a reinforcement learning approach. *IEEE Internet of Things Journal* 10, 5 (2022), 3917–3927.

[79] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[80] Runan Wang, Giuliano Casale, and Antonio Filieri. 2022. Enhancing performance modeling of serverless functions via static analysis. In *International Conference on Service-Oriented Computing*. Springer, 71–88.

[81] Ziliang Wang, Shiyi Zhu, Jianguo Li, Wei Jiang, KK Ramakrishnan, Yangfei Zheng, Meng Yan, Xiaohong Zhang, and Alex X Liu. 2022. DeepScaling: microservices autoscaling for stable CPU utilization in large scale cloud systems. In *Proceedings of the 13th Symposium on Cloud Computing*. 16–30.

[82] Zhaojie Wen, Yishuo Wang, and Fangming Liu. 2022. StepConf: Slo-aware dynamic resource configuration for serverless function workflows. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 1868–1877.

[83] Hao Wu, Junxiao Deng, Hao Fan, Shadi Ibrahim, Song Wu, and Hai Jin. 2023. QoS-Aware and Cost-Efficient Dynamic Resource Allocation for Serverless ML Workflows. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 886–896.

[84] Tong Wu, Pan Zhou, Kai Liu, Yali Yuan, Xiumin Wang, Huawei Huang, and Dapeng Oliver Wu. 2020. Multi-agent deep reinforcement learning for urban traffic light control in vehicular networks. *IEEE Transactions on Vehicular Technology* 69, 8 (2020), 8243–8256.

[85] Fei Xu, Yiling Qin, Li Chen, Zhi Zhou, and Fangming Liu. 2021. λdnn: Achieving predictable distributed DNN training with serverless architectures. *IEEE Trans. Comput.* 71, 2 (2021), 450–463.

[86] Siqiao Xue, Chao Qu, Xiaoming Shi, Cong Liao, Shiyi Zhu, Xiaoyu Tan, Lintao Ma, Shiyu Wang, Shijun Wang, Yun Hu, et al. 2022. A Meta Reinforcement Learning Approach for Predictive Autoscaling in the Cloud. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4290–4299.

[87] Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. 2018. Mean field multi-agent reinforcement learning. In *International conference on machine learning*. PMLR, 5571–5580.

[88] Zhe Yang, Phuong Nguyen, Haiming Jin, and Klara Nahrstedt. 2019. MIRAS: Model-based reinforcement learning for microservice resource allocation over scientific workflows. In *2019 IEEE 39th international conference on distributed computing systems (ICDCS)*. IEEE, 122–132.

[89] Guangba Yu, Pengfei Chen, and Zibin Zheng. 2019. Microscaler: Automatic scaling for microservices with an online learning approach. In *2019 IEEE International Conference on Web Services (ICWS)*. IEEE, 68–75.

[90] Guangba Yu, Pengfei Chen, Zibin Zheng, Jingrun Zhang, Xiaoyun Li, and Zilong He. 2023. FaaSDeliver: Cost-Efficient and QoS-Aware Function Delivery in Computing Continuum. *IEEE Transactions on Services Computing* (2023).

[91] Hanfei Yu, Athirai A Irissappane, Hao Wang, and Wes J Lloyd. 2021. Faasrank: Learning to schedule functions in serverless platforms. In *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. IEEE, 31–40.

[92] Tianyi Yu, Qingyuan Liu, Dong Du, Yubin Xia, Binyu Zang, Ziqian Lu, Pingchao Yang, Chenggang Qin, and Haibo Chen. 2020. Characterizing serverless platforms with serverlessbench. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 30–44.

[93] Zhibin Yu, Zhendong Bei, and Xuehai Qian. 2018. Datasize-aware high dimensional configurations auto-tuning of in-memory cluster computing. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 564–577.

[94] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, et al. 2019. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *Proceedings of the 2019 international conference on management of data*. 415–432.

[95] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. 2021. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control* (2021), 321–384.

[96] Zhuangzhuang Zhou, Yanqi Zhang, and Christina Delimitrou. 2022. Aquatope: Qos-and-uncertainty-aware resource management for multi-stage serverless workflows. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. 1–14.