FEO: Efficient Resource Allocation for FaaS at the Edge

Anirudh Sarma
asarma31@gatech.edu
Georgia Institute of Technology
Atlanta, USA

Jinsun Yoo
jinsun@gatech.edu
Georgia Institute of Technology
Atlanta, USA

Jithin Kallukalam Sojan
jsojan3@gatech.edu
Georgia Institute of Technology
Atlanta, USA

Umakishore Ramachandran
rama@gatech.edu
Georgia Institute of Technology
Atlanta, USA

Myungjin Lee
myungjle@cisco.com
Cisco Research
Bellevue, USA

Abstract

Geo-distributed Edge sites are expected to cater to the stringent demands of situation-aware applications like collaborative autonomous vehicles and drone swarms. While clients of such applications benefit from having network-proximal compute resources, an Edge site has limited resources compared to the traditional Cloud. Moreover, the load experienced by an Edge site depends on a client's mobility pattern, which may often be unpredictable. The Function-as-a-Service (FaaS) paradigm is poised aptly to handle the ephemeral nature of workload demand at Edge sites. In FaaS, applications are decomposed into containerized functions enabling fine-grained resource management. However, spatio-temporal variations in client mobility can still lead to rapid saturation of resources beyond the capacity of an Edge site.

To address this challenge, we develop *FEO (Federated Edge Orchestrator)*, a resource allocation scheme across the geo-distributed Edge infrastructure for FaaS. FEO employs a novel federated policy to offload function invocations to peer sites with spare resource capacity without the need to frequently share knowledge about available capacities among participating sites. Detailed experiments show that FEO's approach can reduce a site's P99 latency by almost 3x, while maintaining application service level objectives at all other sites.

DEBS '24, June 24–28, 2024, Villeurbanne, France

ACM ISBN 979-8-4007-0443-7/24/06
https://doi.org/10.1145/3629104.3666033

CCS Concepts

• **Computer systems organization** → **Peer-to-peer architectures**.

Keywords

Serverless Computing, FaaS, Edge Computing

ACM Reference Format:

Anirudh Sarma, Jinsun Yoo, Jithin Kallukalam Sojan, Umakishore Ramachandran, and Myungjin Lee. 2024. FEO: Efficient Resource Allocation for FaaS at the Edge. In *The 18th ACM International Conference on Distributed and Event-based Systems (DEBS '24), June 24–28, 2024, Villeurbanne, France.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3629104.3666033

1 Introduction

The recent proliferation of geo-distributed Edge micro-datacenters [22, 34] is well-timed to meet the need for situation-awareness applications like collaborative Autonomous Vehicles (AVs), Drone Swarm Navigation and Video Analytics. These applications are typically latency critical and bandwidth heavy and can benefit from resources at close proximity to the source of data. However, unlike Cloud datacenters, computing resources at the Edge are limited. Yet, we want to preserve the elasticity of the Cloud so that developers are oblivious of the resource scarcity at the Edge.

In recent years, Cloud providers have increasingly adopted the Function-as-a-Service (FaaS) model that provisions resources at the granularity of functions instead of dedicating Virtual Machines (VMs). On Cloud, the FaaS paradigm is attractive more for its ease in development, deployment and

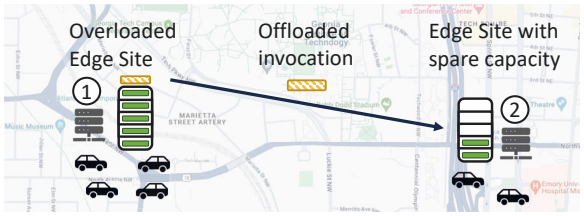


Figure 1: Spatio-Temporal variance in AV traffic can saturate proximal Edge clusters. Opportunistic offload of function invocations to peer clusters can help protect application SLO.

the economic incentive of not having to pay for unused services. On capacity-limited Edge, however, FaaS instead provides a suitable resource allocation model that enables function containers to be rapidly provisioned for an application commensurate with variances in the current workload.

There could be periods when an Edge site is overwhelmed by heavy workload while other sites in the vicinity are under utilized. This load imbalance will be commonplace for situation-aware applications that experience workloads with high variance both in space and time. The variance rises from the non-uniformity in the distribution of activity (e.g., vehicles or pedestrians) [33]. To illustrate, we conduct a trace-driven simulation of Edge-assisted AVs [2] (see §6.1 for setup details) wherein each vehicle generates function invocations to its network-proximal Edge site. Fig. 2 shows a 1-minute snapshot of the workload for three representative Edge sites. Such unpredictable traffic conditions can lead to rapid saturation of a specific site. Consider the scenario depicted in Fig. 1 where Site 1 is overloaded with requests. Intuitively, Site 1 can benefit from offloading excess function invocations (yellow/hatched) to a peer site (Site 2) to mitigate queuing delays and meet the application's latency constraints.

Devising an offloading scheme comes with several challenges. The system should be aware of the network proximity between clients and Edge-sites, as well as among Edge-sites. It should also be aware of the resource availability of peer sites and choose candidates that have spare capacity at that instant to offload excess requests.

We propose *FEO (Federated Edge Orchestrator)*, a resource allocator for efficiently offloading function requests in a fully decentralized manner at each Edge site. Each FEO instance dynamically discovers and utilizes spare resources at peer sites to serve function invocations.

There exists vast prior work in load balancing in Cloud computing [25], including Nginx [24] and HAProxy [14]. FEO is distinguished from such prior work since it is built for each Edge site to make autonomous offloading decisions concurrently to achieve a global load balance that meets an application's *Service Level Objectives (SLOs)*.

There is also prior art in resource allocation policies for geo-distributed Edge infrastructure. We first conduct a simulation-based study comparing such policies with FEO to showcase the performance of FEO relative to such prior art. We have also implemented an end-to-end distributed FaaS platform that incorporates FEO to validate the simulation results with both microbenchmarks and trace based application studies.

In this work, we make the following contributions:

- *FEO*, a fully decentralized resource orchestrator for function offloading with minimal state exchange among the participating Edge sites to achieve a global load balance while respecting application SLOs.
- A simulation-based study to compare FEO with other contemporary offloading policies.
- An end-to-end system implementation of a FaaS platform that incorporates FEO. The platform provides the flexibility to incorporate other offloading policies and is made available to the community as open-source software[1].
- An evaluation study of the end-to-end system using microbenchmarks and two exemplar situation-aware applications (Drone-Swarm Navigation and Vehicular Video Analytics). FEO's P99 latency reduction ranges from 2x-35x per-site depending on the application. Among the evaluated policies, FEO's *federated* policy alone helps meet application SLOs across all participating sites without dropping requests. An ancillary contribution is the conversion of these two applications from their monolithic versions into the FaaS paradigm.

2 Background

2.1 Serverless Computing and FaaS Platforms

Serverless in the Cloud. Serverless Computing enables developers to design, develop and deploy their applications as a composition of interconnected functions, such that resources are provisioned at the granularity of function invocations. Cloud service providers (such as Amazon and Microsoft) deal with auto-scaling function instances based on load, relieving auto-scaling burden from the clients.

FaaS Platforms. App developers may choose to use the FaaS facilities provided by the service providers (e.g., Azure functions and AWS Lambda), or use an open-source implementation like OpenWhisk[1] for deploying their functions. Typically, a function invocation arrives at a platform proxy, which is then routed to a resource orchestrator (such as Kubernetes [7]). The resource orchestrator deploys a container to host the function provided by the developer, which then awaits further function requests from the user.

[1]https://github.com/gt-epl/feo

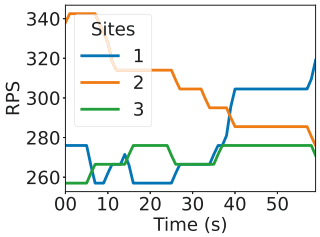


Figure 2: Aggregate requests at each Edge site varies spatio-temporally over a minute period due to mobility of clients in the SFCabs dataset.

Serverless at the Edge. Latency-sensitive and bandwidth-hungry applications from different application domains (e.g., gaming, Edge-assisted AV control, and Augmented/Virtual Reality) could benefit from extending the serverless paradigm from the Cloud to the Edge. Orchestration for serverless at the Edge has to necessarily be aware that resources are limited at individual Edge sites. Therefore, autoscaling decisions to deal with load fluctuations would need to consider peer Edge sites for function offloading. To date, the Edge offerings from the Cloud providers [22, 34] use orchestrators residing in the Cloud with little or no autonomy for the Edge in resource allocation decisions. There has been recent research work, namely, OneEdge [31], which is a hybrid scheme to give partial autonomy to the Edge for resource allocation decisions. It should be noted, however, that neither the Edge offerings from the Cloud providers nor OneEdge is specific for serverless computing at the Edge.

2.2 Resource Orchestration for FaaS at the Edge

FaaS platforms typically defer autoscaling decisions to the central orchestrators situated in the Cloud. Measurement studies [34] have shown that inter-Edge latencies are often shorter than the Edge-Cloud latencies. These studies suggest examination of offloading policies at the Edge that do not necessitate wide-area-network (WAN) traversals to the Cloud.

2.3 Prior art in Function Offloading

For efficient function offloading, each Edge site must decide on a policy for accomplishing two key objectives for optimally balancing the load across the Edge infrastructure. First, an Edge site (also referred to as an "offloader") should decide quickly and locally which peer node to offload to (also referred to as an "offloadee"). Second, the independent decisions taken by each Edge site should lead to a globally balanced schedule. Achieving the two key objectives requires policies to perform two tasks: 1) Become aware of the available resource capacities at their peer sites and 2) Decide when to offload functions.

There is limited prior art specific to function offloading in an Edge infrastructure. However, there is considerable prior art on the topic of load balancing in distributed systems. For example, load balancers such as Ngnix [24] and HAProxy [14] are centralized systems that aim to reduce load imbalance for application deployment in the Cloud. Such systems were not designed specifically for function offloading wherein multiple Edge sites make autonomous decisions. There is also prior art in resource orchestration for Edge infrastructures. For example, Fado [32] relies on a central orchestrator in the Cloud to perform function offloading decisions between Edge sites. Similarly, OneEdge [31] tries to execute requests locally and only offload when the site is saturated. However, it still relies on the Cloud for these requests and cannot handle FaaS requests.

We first describe policies *derived* from such prior art and their limitations that serve as motivations for the *Federated Edge Orchestrator (FEO)* to be described in §3. We use these derived policies as points of comparison to FEO in our simulation studies §4 and system evaluation §6.

Round Robin (*rr*): Round-robin is a time-tested scheduling policy and is the default for many Cloud-based load balancers such as HAProxy [14] and Nginx [24]. In adapting this policy to FaaS at the Edge, every Edge site autonomously offloads to its peer sites in a round-robin fashion. As this policy is oblivious to the resource availability of its peers, it does not need to track and/or disseminate this knowledge.

Centralized (*central*): This policy is derived from centralized in-Cloud orchestrators (such as Fado [32]). Resource availability of every site is periodically communicated to the centralized orchestrator that maintains the global state. Every Edge site consults the centralized orchestrator (by traversing the WAN) to determine an offloadee for handling function invocations. The *central* policy addresses both the challenges of estimating the resource availability at every Edge site and achieving a globally optimal schedule. Yet, centralizing offloading decisions for invocations incurs expensive WAN traversals on the critical path of decision-making and potentially inflate end-to-end latencies.

Hybrid (*hybrid*): This policy draws inspiration from OneEdge [31] wherein the Edge site is allowed to retain local autonomy for control decisions. This policy improves the *central* policy by restricting expensive WAN traversals to a central orchestrator to only those invocations that are likely to violate their SLO during busy periods at the offloader. In the function offloading context, *hybrid* prioritizes handling most function invocations locally, while offloading those invocations to the central orchestrator that are likely to violate their SLO. For such offloaded invocations, the central orchestrator recommends a suitable offloadee with spare capacity to service them. Like *central*, every Edge site locally monitors resource availability and reports it to the central authority to facilitate offloading decisions.

Response Time Based (*rtime*): This policy is based on the work of Cicconetti, et al. [4] and implements a variant of weighted round-robin and obviates the need for a centralized orchestrator. Every Edge site independently maintains its own table of weights for sites that can handle the function invocation. The offloader then selects the offloadee with the lowest weight to offload the function invocation. These weights are dynamically updated at each Edge site based on the response time for function execution at the offloadee.

Limitations: Although these policies derived from prior art address the objectives we identified, they come at a cost. For instance, both *rtime* and *rr* incur high data movement (shown in §4.2.1) to balance the load globally due to their eager approach to offloading. Both *central* and *hybrid* rely on the appropriate frequency of state exchange to minimize staleness and reduce data movement. However, both *central* and *hybrid* may still be vulnerable to erroneous offloading and expensive WAN traversals. The limitations of existing policies motivate a new design that avoids a centralized state, in-turn reducing WAN traversals and data staleness.

3 Federated Edge Orchestrator (FEO)

3.1 Key Assumptions

To coordinate resource allocation across the Edge infrastructure, FEO relies on five key assumptions:

Inter-Edge site Latency: An important assumption is that the inter-Edge site latency is considerably shorter and more predictable than WAN latency. This assumption is based on the fact that there typically is fiber-optic direct connectivity between central offices of telcos [10], and supported by published data from providers such as Alibaba [34].

Cumulative Resource Availability: The focus in our work is on alleviating site-local hotspots. We assume that the cumulative resource capacity across all Edge sites always exceeds the cumulative workload. With this assumption, an over-saturated site can find peer-sites with sufficient spare capacity to offload and preserve application SLO.

Availability of Warm Containers: There are orthogonal efforts in keeping containers warm [13, 23] that could be applied to resource orchestration for FaaS at the Edge to mitigate performance loss due to cold starts. To ensure our study of policies is predictable and repeatable, FEO assumes the availability of warm containers at the target offloadee Edge sites deduced via the per-function queue-depths obtained from peers (described in §3.3.2).

Per-Function Offload Granularity: An application can be decomposed into a Directed Acyclic Graph (DAG) of multiple function components. The application developer must specify the target latency. FEO operates with the promise

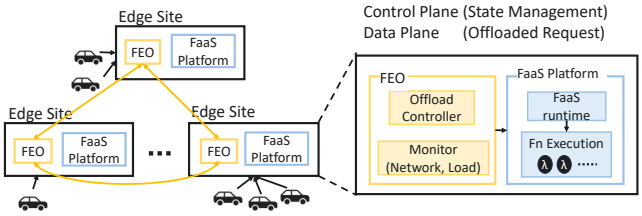


Figure 3: FEO (in yellow) seamlessly integrates with the FaaS Platform (in blue) at every edge site.

of achieving per-function SLOs by monitoring load at a per-function per-invocation granularity. This fine-grained orchestration results in efficient use of scarce Edge resources.

Offloading Stateless Invocations: In this work, we focus on addressing the challenges of efficient offloading policies for stateless functions. Orchestrating stateful functions requires additional machinery for application state maintenance and is beyond the scope of this paper.

3.2 The *federated* approach in FEO

FEO's ability to effectively offload function invocations is reliant on the novel *federated* policy. It achieves the two objectives identified in §2.3 in a manner that avoids WAN traversals on the critical path.

Under this policy, Edge sites minimize the amount of state messages needed to be exchanged to learn about the resources at its peer sites while meeting the application-specified SLOs. A key insight inspiring the *federated* approach is that offloaders can "*fail fast*" while attempting to offload and in the process learn the offloadee's resource availability.

The federated offload policy works as follows independently at each Edge site:

- Upon the need to offload, an Edge site initiates an offload request to a *random* peer Edge site.
- Upon receiving the request, the offloadee may either choose to *accept* or *decline* the request.
- A declined request is returned immediately to the offloader and is scheduled for execution locally to ensure timeliness of the function execution.
- On the other hand, an accepted request is executed and the response is returned to the offloader, allowing the continuation of the downstream functions in the application's DAG.
- The response (either acceptance or denial) from the offload request contains the resource availability of the offloadee (including the availability of warm containers for the desired function) and is used by the offloader to construct a local view of resource availability at peer Edge sites.

3.3 Architecture of FEO

Fig. 3 presents an overview of the FEO architecture within an Edge site and its interplay with its peers. FEO runs alongside a FaaS platform and forwards function invocations to the FaaS platform instance of the appropriate Edge site(§5). It consists of two main entities: a *Monitor* and a *Controller*. The Monitor consists of network and load monitoring components. The network component conducts network proximity estimation to determine inter-Edge site latency; the load component keeps track of the local resource usage as well as resource information gleaned from peers. The Controller implements the federated policy that we outlined earlier. It utilizes the monitored load information to make offloading decisions. At each Edge site, the Controller intercepts all incoming function requests. It then leverages the information learned during monitoring to determine a target offloadee site to forward the invocation.

3.3.1 Monitor

Load Monitor. The load monitor at an Edge site is responsible for bookkeeping per-function capacities of peers in the same latency equivalence class. FEO expresses capacity in terms of queue-depth of the outstanding invocations, which was empirically(§6.2) found to be more effective at handling function invocation bursts. This is because the queue-depth metric inherently captures the processing capacity of a site by approximating the containers available to handle function invocations. A deep queue likely indicates a saturated site. The Monitor keeps track of per-function queue depths across the peer sites. FEO's *federated* policy relies on this knowledge to select the peer site with the least enqueued requests for the function to be offloaded.

Network Monitor. The network monitor at each Edge site maintains a list of peer sites that are in the same latency equivalence class. We use Network Coordinates [8, 18], which is a low overhead method of estimating latency proximal peers. The Network Coordinate algorithm maps sites to synthetic coordinates such that their Euclidean distance approximates the latency between the sites. Every site thus maintains its own peer list for offloading.

3.3.2 Controller

The Controller is the heart of FEO and is designed to be independent of the associated FaaS platform.

Application Deployment. When an application represented as a DAG is deployed at an Edge site, that Edge site becomes the *home* for that application. The schema associated with the DAG specifies the data and control flow between the components of the application. Akin to the classic dataflow architecture, the controller is responsible for launching a function when its inputs are ready. The client that deployed the application sends requests to the entry point function

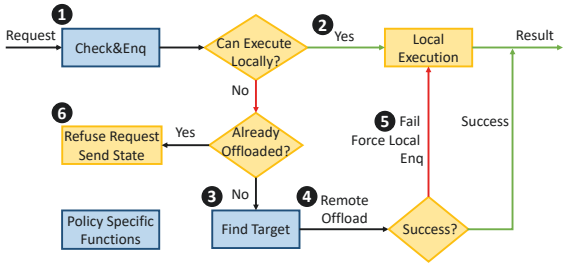


Figure 4: FEO Controller. Plug-and-play design allows the *Policy Specific* components (in blue) to be changed without affecting the overall FEO architecture. The control loop keeps track of per-function queue depths to aid the policy component make offload decisions.

of the DAG always to the *home* Edge site. FEO has machinery for migration of the *home* for an application to a peer Edge site based on the mobility of the client. Description of the DAG migration machinery is outside the scope of this paper. We only describe the actions at an Edge site when a function becomes ready to be launched. The Controller at an Edge site uses the associated FaaS platform to launch a function if the decision is made to execute the function locally. The Controller shields the client from the details of the underlying FaaS platform used in the system.

Function Launch. The Controller is designed to be *plug and play* with respect to the function offload policy. Fig. 4 depicts the *control loop* of the FEO Controller.

❶ When a function is ready to be launched the policy determines if the request can be enqueued at the local site. The default policy of FEO uses the load monitor component to decide if the function can be launched locally.
❷ If the function can run locally, the request is forwarded to the local FaaS platform.
❸ On the other hand, the policy is consulted to determine an offload target in light of site saturation. The default policy of FEO is to use queue-depths at network proximal peer Edge sites (as determined by the load monitor component) and the availability of warm containers for that function at the target site.
❹ The Controller then proxies the request to the offloadee and awaits the status of the invocation.
❺ Upon offload failure, the default policy decides to locally enqueue the request regardless of the current site capacity. In FEO, we disallow nested offloads to prevent perpetual offloading.
❻ At the offloadee, if the request cannot be processed, a negative acknowledgment is sent back to the offloader. Note that the state of the offloadee is piggy-backed with the acknowledgment regardless of the status of the offload request,

and the offloader uses this to update its own state about its peers.

Function Completion. Upon successful completion of the function execution by the FaaS platform, the Controller would either send the result back to the offloader if it was an offloaded request. The Controller at the *home* site for the DAG would collect the result of the function execution and decide on the next steps for downstream execution based on the schema associated with the DAG.

Offload Predicate. The default policy engine in the Controller uses the queue-depth for a function at an Edge site and the availability of warm containers for that function in its offload decision. The queue-depth parameter is better equipped to handle invocation bursts than coarse-grained metrics like CPU utilization and observed request rates. The request inter-arrivals are often bursty and unpredictable, leading to rapidly fluctuating queue lengths. Reacting to the instantaneous queue-length can lead to erroneous offloads at an offloader site and corresponding denials at an offloadee site. The FEO Controller instead considers the hysteresis of fluctuating queue-depths by calculating the smoothed moving average of queue lengths over a fixed timed window. We show in §6.2.3 that for bursty inter-arrival behavior of invocations belonging to mobile clients, a short window of 1 second with a 70% weight on historical queue length suffices to use as a signal to offload.

4 Simulation Studies

Using a discrete-event simulator, we conduct studies to understand the efficacy of FEO's *federated* policy(§3.2) in comparison to the policies we derived from prior art (§2.3). The primary figure of merit used in the study is end-to-end latency.

4.1 Simulation Setup

We simulate the offload policies using `faas-sim` [28], a trace driven simulator powered by `SimPy` python simulation framework. `faas-sim` is designed to simulate invocation traces on a single FaaS deployment. We modify the framework to extend it to handle resource orchestration across multiple FaaS deployments (i.e., multiple Edge sites).

`SimPy`'s clock tick is assumed to be 1 ms long. Our analyses focus on function invocations that incur around 100 ms of simulated processing latency on average similar to end-to-end latencies observed in AV [35] and drone navigation [16]. Finally, we source inter-site latencies from [34] that observe 5 ms round trip time (RTT) for Edge-Edge and around 20 ms RTT for Edge-Cloud connections.

Workload Generation. We devise a synthetic workload generator per-site based on a mobile user's behavior and incorporate the following characteristics: 1) A user's initial connection to their proximal Edge is dictated by Poisson arrivals with a mean rate of λ; 2) The user may stay connected to an edge node for some time t, which is exponentially distributed around mean t_{mean}; and 3) The user typically generates requests at a fixed rate r which corresponds to the rate of sensor streams. We then vary λ and t_{mean} by randomly choosing values from the ranges specified in Table 1 to obtain request arrival traces per-site. The traces for all Edge sites are collectively referred to as a profile.

Parameter	values
λ	$0.007 - 0.013s^{-1}$
t_{mean}	$281 - 485s$

Table 1: Workload Parameters. Extracted from the mock-up simulation of cabs in SFCabs dataset [26] over a 1 hour period (see details in §6.1).

Baselines. We consider two baselines: 1) *no-offload*: Edge sites are unaware of spare resources of its peers and hence do not offload any request. Observing per-site and aggregate end-to-end latencies under this scenario will help us examine the pitfalls of local hotspots. 2) *state-aware*: The simulation setup allows us to evaluate the policies against a state-aware offload policy (*state-aware*) in which every Edge site is aware of the instantaneous resource capacity of all their peers and incurs no cost in learning this information. A saturated site relies on the already available information on peer capacities to select an offloadee with the most available capacity. Note that the site's foresight does not extend to the decisions taken by peers and can result in a sub-optimal offloading decision. Nevertheless, the *state-aware* policy obviates staleness in state. Its performance gap over the other policies helps compare policies with respect to their costs associated with maintaining latest information about resource availability of peers.

4.2 Simulation Results

We structure our simulation experiments to determine the optimal policy to use for cross-site orchestration by answering the following questions:

(1) What impact does the choice of policy have on per-site (local) and global latencies of function invocations?
(2) Is the performance of the policy sensitive to the number of peer-sites in the system and to the inter-site latencies?

4.2.1 Policy Performance

End-to-End Latency. We setup 10 Edge sites where for each site, we generate function invocations according to the site's inter-arrival trace in the profile. This setup is kept identical across runs while we employ different offload policies. Fig. 5 summarizes the results from this experiment where we see the end-to-end latencies observed across all of the Edge sites for a function with average execution time of 100

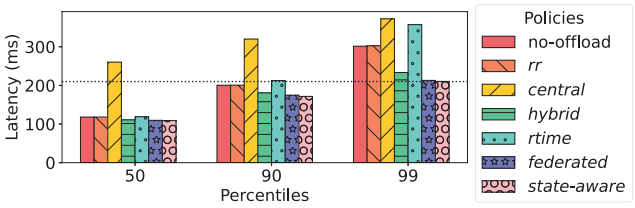


Figure 5: Impact of various load profiles on End-to-End latency.

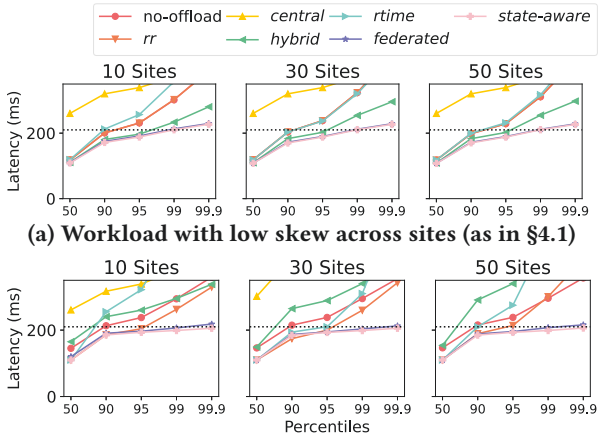
ms. First, the baseline *no-offload* policy indicates that only a small fraction of requests suffer from site-local saturation. We observe a 1.7x and 2.5x inflation in P90 and P99 over median (P50). The goal of offloading should be to ameliorate higher percentiles. The *state-aware* offload policy (in pink) helps identify our improvement potential. It reduces the inflation in P90 and P99 to 1.4x and 1.7x respectively. Interestingly, P50 latencies are also reduced by 10% as the policy strives to globally balance the load. The naive *rr* shows a similar performance to *no-offload* as each site in *rr* is oblivious to the load on the target site and aggregate load experienced by a site post offloading is similar to *no-offload*. Next, on one end of the spectrum, *federated* closely tracks the latencies of *state-aware* as a result of agile target determination. On the other end, the *central* policy is not viable as a candidate because of high latencies (even at the P50) due to WAN traversals to a centralized load balancer lying on the critical path for every request. Moreover, this is exacerbated by incorrect offload decisions arising from staleness in state. *hybrid* alleviates the latency inflation for median requests, yet suffers the same inflation as *central* for higher percentiles. This is because the fraction of requests offloaded in *hybrid* are susceptible to the same pitfalls observed in *central*. *rtime* is similarly unable to perform well because it awaits the actual execution of the function.

Data Movement. Among the policies, both *rr* and *rtime* tend to de-prioritize site-local executions to ensure a globally balanced utilization of resources at the cost of ferrying large volumes of offload requests amounting to 70-80% of all requests on average for the given workload profile. The offloaded requests are ferried either on horizontal communication links between sites.

The capacity-aware policies (*hybrid* and *federated*) minimize data movement costs over these links by prioritizing local execution that yields a lower fraction of offload requests amounting to 5-10% of all requests. The *federated* approach however, has a lower footprint of control plane data movement as it piggybacks state information on the offloadee acknowledgments instead of regular periodic updates.

4.2.2 Sensitivity Analyses

Scaling across sites. Pervasive deployment of geo-distributed sites can result in a large number of peers for a site. Therefore, this experiment assesses the ability of every policy to scale gracefully with an increasing number of participating sites. The workload for each additional site is generated similar to §4.1. Fig. 6a shows the impact on the end-to-end latency as the number of sites is scaled to 50. Most policies' performance does not suffer with increases in site counts (e.g., the *hybrid* approach sees a mere 9% increase in P99 when moving from 10 to 50 sites.). *rtime* in particular observes a latency decrease with increasing site-count. However, its absolute latencies still remain high. *federated* tracks *state-aware* independent of the site-count similar to the results seen in §4.2.1. We note that the insensitivity to site count is tied to the small fraction of requests that are offloaded for the specific workload profile.



(a) Workload with low skew across sites (as in §4.1)

(b) Workload with high skew across sites.

Figure 6: Impact of number of Edge sites on End-to-End latency.

A workload profile with sufficiently high skew across sites can force more requests to be offloaded, thus amplifying differences in policies as we vary the site count. To test this principle, Fig. 6b uses a manually constructed profile consisting of a high skew in workload among Edge sites that results in a larger fraction (~20-25%) of the requests being offloaded. Under this scenario, we observe a noticeable impact on performance as we increase the number of participating sites. However, each policy is differently impacted by increasing site-counts. For example, the poor scalability for *central* and *hybrid* is attributed to the increased search space for an offload candidate coupled with the staleness in global state that leads to erroneous offloading. With increasing site-count, additional sites are similarly susceptible to becoming hotspots. The *federated* approach adopts a state-minimal approach and does not rely on periodic updates and yields a similar performance to *state-aware* in both scenarios. These

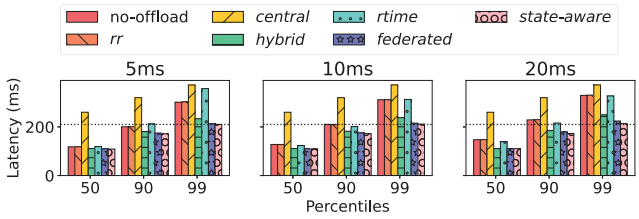


Figure 7: Impact of varying Inter-Edge-site latencies on the policy performance. Capped at 250ms to highlight marginal differences across graphs.

advantages make *federated* scalable with the site-count and generalize well across workload profiles.

Impact of peer latencies. Although the latencies were sourced from existing datasets on real-world public Edge-Cloud measurements [34], we also examine the impact of inter-site latencies on the efficacy of the various policies. To this end, Fig. 7 summarizes results for the various policies as we vary the inter-site latencies. For the most part, inter-site latencies do not appear to have a significant impact on the policy's behavior already captured by the previous experiments, other than the universal but marginal increase in end-to-end latency with the increasing peer latencies. We note that it is not viable to offload to peer sites that are greater than 20 ms away or take longer to reach than reaching the nearest Cloud that has abundant resources. However, a large difference between Edge-Edge and Edge-Cloud is not essential to demonstrate the efficacy of the *federated* approach.

4.2.3 Summary

The policy experiments simulate the orchestration of resources across multiple Edge sites by offloading function invocations across the deployed FaaS clusters. The end-to-end latency observations and sensitivity analyses suggest that a *federated* approach that actively learns peer capacities through offload-request acknowledgments is expected to scale and generalize better compared to other policies. Alternatively, it highlights the inherent weaknesses with policies that require centralized state (*central*) or are capacity-oblivious (*rr*, *rtime*). These weaknesses are exacerbated in the presence of real systems effects (network, memory, CPU, runtime, etc.) resulting in further performance degradation. Findings from the simulated study help narrow our focus towards the prototype implementation and a real-world evaluation of a cross-site orchestrator equipped with a practical and scalable policy variant in the next section.

5 Implementation

The components described in §3 are all implemented in Go. FEO runs as a standalone binary and is provisioned on resources allocated for the site's control plane. FEO implements a proxy HTTP server to intercept requests similarly to current FaaS platforms. Client communications upto the frontend gateway are encrypted while backend connections are assumed to be unencrypted. FEO then forwards requests to either the site-local FaaS platform, or offloads the invocation to a FEO instance on a peer site. We use an HTTP header (or lack thereof) to convey to the site if a request is an offloaded one. The target site then sets another header in the response to indicate its acknowledgment. We use an in-house FaaS platform that disables cold starts by pre-provisioning containers in line with the key assumptions in §3.1.

Monitor. For Network, we leverage Serf's [15] implementation of the Network Coordinate protocol. The monitor estimates inter-site RTTs from the network coordinates and updates the per-function peer lists. For Load, we periodically monitor the queue-depths as recorded by the control-loop at a per-function basis.

5.1 Policy Implementation Details

The study conducted in §4.2.3 suggests the infeasibility of *central* and *rr* approaches to offloading even in the absence of real world systems effects. We thus limit our real system implementation to 2 policies and discuss the details below:

federated. This is the policy used by FEO. As discussed in §3, we rely on smoothed average queue lengths to determine whether a site is saturated. To offload, a saturated site's offloader bootstraps the site's state by first issuing offloading requests at random to its peers. The recipient site sets a response header to indicate the status of the execution and the queue lengths observed in the recent past for the function. The offloader uses this to update its state about the offloadee. The per-site queue-lengths are used as weights to perform a weighted random selection of the offloadee for subsequent offload requests.

hybrid. The *hybrid* implementation consists of two components: 1) A central state aggregator that implements a gRPC server that periodically receives per-site updates for per-deployed-function queue lengths. The central aggregator provides an API for each site to query for the optimal target to handle a function invocation. 2) A site-local offload component that leverages the smoothed-average queue length to determine if a request is to be offloaded. It then queries the central state aggregator, obtains the target site, and offloads the request. The target site services the request and returns the response to the source site. However, it may also choose to deny the offload request upon which the offloader must locally handle the invocation request.

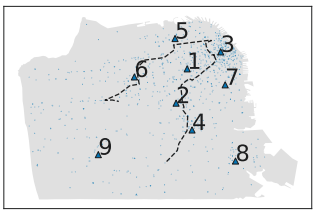


Figure 8: The map of San Francisco shows 9 sites (triangles) obtained by clustering cellular towers (dots) and a sample trajectory of a vehicle from SFCabs dataset.

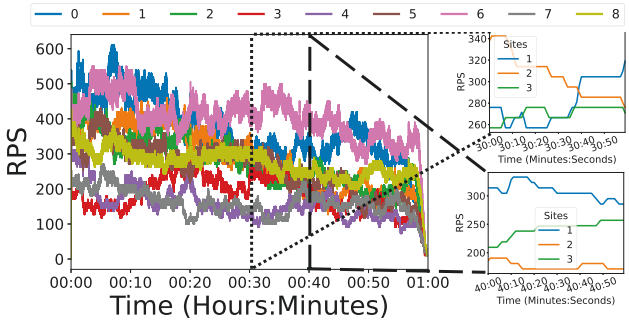


Figure 9: Space-time Varying Load at Edge Sites. Vehicles generate requests at 10RPS to their respective proximal Edge sites along their routes specified in the SFCabs dataset.

6 Evaluation

In this section, we evaluate FEO on the policies we implemented in §5. We first discuss microbenchmark evaluations to answer the following questions.

(1) How well do different policies employed by FEO fare in meeting a synthetic function's SLO?
(2) What facets of a policy can help in meeting the application SLO?

We then demonstrate benefits of FEO on two real world applications that have been adapted for a FaaS deployment.

Experimental Platform. We provision Intel(R) Xeon(R) Silver 4114 nodes on Cloudlab [6], each with 20 cores (10 per socket) and 192 GB RAM that are inter-connected over a switch using 10G NICs.

Edge Sites. We emulate Edge Sites on the Xeon Servers, where each site runs an independent instance of the in-house FaaS platform on socket0.

Clients. Clients are instantiated and generate invocations on the same server as its network-proximal Edge site.

6.1 Workload Characterization

The SFCabs dataset contains timestamped coordinates of taxis moving within San Francisco. We examine a snapshot of a random hour to emulate clients (vehicles, drones) for a reliable workload representing client mobility across Edge sites. We note that the observations from this setup are similar to other hourly snapshots in the complete SFCabs dataset. Cellular towers [5] are clustered via k-means to obtain 12 clusters. The cluster centroids were further grouped according to the density of cellular towers within each cluster into 9 representative Edge sites (shown in Fig. 8). This grouping is done to ensure that the load on sites is not heavily skewed in favor of offloading.

Next, we emulate vehicles generating requests to their proximal Edge site and model the aggregate requests observed at each Edge site. Fig. 9 shows the aggregate requests at 10 RPS at 9 Edge sites with spatio-temporal variations. Zooming in at minute granularity (see subgraphs) highlights non-determinism over time in RPS load across Sites 1,2 and 3. The mock-up yields an inter-arrival trace that is used both for microbenchmark and real-world application study. Specifically, we randomly sample timestamps in the complete trace to obtain minute-long traces (similar to subgraphs in Fig. 9) to evaluate the latency performance of different policies.

6.2 Microbenchmarks

The microbenchmark evaluations use a cpu-intensive function [17] with an average service time of 100ms and an SLO of 200 ms.

6.2.1 Policy Performance

We sample 4 timestamps across the hour-long trace as described in §6.1 and examine the efficacy of different offloading policies. Fig. 10 shows the P99 latency across the 9 Edge sites. Unsurprisingly, any amount of offloading can help alleviate saturation at specific sites as seen by decrease in P99 for Site 1 at t=2, and Site 7 at t=30 below 300 ms. In comparing different offload policies, *federated* fares better than all other policies as it ensures that a site's P99 latency does not cross 200 ms. *federated*'s P99 characteristics remain similar across the sampled timestamps and shows the performance generalizability of the approach. Next, while *hybrid* helps minimize site-saturation, the cost of doing so is greater than *federated* as seen by universal increase in P99 across all sites.

6.2.2 Analyzing Offloads

To understand the P99 results, we look at the cumulative percentage of requests arriving on all sites that were offloaded under each policy in Fig. 11. The translucent portion of the bars depict the offloads that were rejected. If we consider the 2nd minute, *federated* successfully offloads 7.8% of the requests yet only 0.12% (barely visible in the plot) of the requests are rejected. Alternatively, *hybrid* only offloads 6% of the requests while experiencing a rejection of 2% on average. The increased fraction of rejections negatively affects

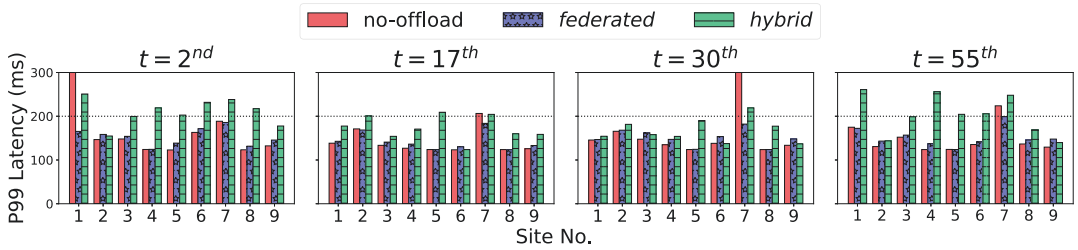


Figure 10: The *federated* policy alone meets the SLO at 200ms (black dotted line) across all sites at various minute-timestamps in the SFCabs trace.

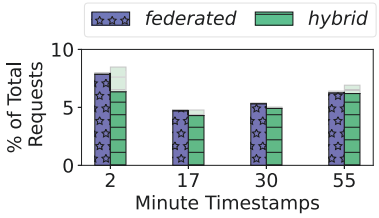


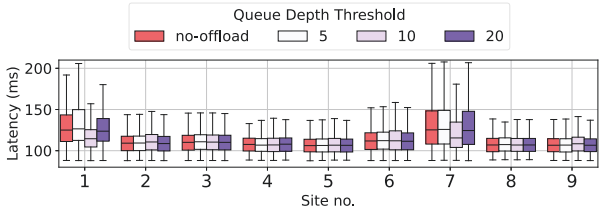
Figure 11: Percentage of total requests offloaded across different timestamps under different policies. *hybrid* rejects a larger portion of offloaded requests as shown by the translucent portions.

the P99 latency. Increased rejections can arise due to the staleness in the resource capacity information as they are only periodically exchanged.

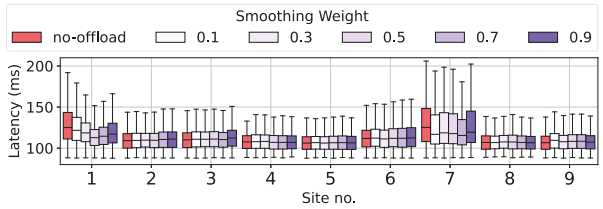
6.2.3 Sensitivity to *federated* parameters

Next, we conduct a sensitivity study (Fig. 12) for the parameters that determine the performance for the *federated* policy. We first analyze the queue-depth threshold required for the best latency profile on a site with 10 warm containers. Fig. 12a shows that a threshold of 10 (same as number of warm containers) offers a good tradeoff between suffering too many rejections vs. offloading too late.

Similarly, the optimal smoothing weight for capturing a site's capacity incorporates only 30% of the new instantaneous queue-depth as seen by lower latencies for site 1 and 7 in Fig. 12b.



(a) Maximum queue-depth threshold



(b) Queue-depth smoothing weight

Figure 12: The sensitivity study for the *federated* policy at $t = 2^{nd}$ minute in terms of latency impact shows the optimal depth threshold and smoothing weight to be 10 and 0.7 respectively (lowest whisker).

6.3 Real World Evaluations

In this subsection, we make real-world situation-aware applications FaaS-compliant and evaluate FEO's ability to honor per-function SLO requirements under a realistic scenario incorporating client mobility.

6.3.1 Video Analytics Pipeline

Video analytics applications on the edge take and provide meaningful insight. We take a publicly available monolithic application [11] and decompose it into a sequence of multiple functions. The overall pipeline could be represented as a DAG, where each vertex is the function and the edge represents data dependency between functions. Handling this application requires FEO to maintain the queue information of multiple applications.

Fig. 13 depicts the P99 latency for two of the functions (filter, detect) at timestamp t = 2. Site1 is overloaded with requests beyond its local resource availability and cannot handle the requests without offloading. Compared to other policies, FEO is able to distribute the load most evenly.

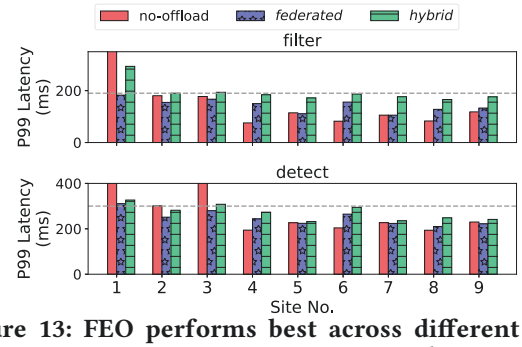


Figure 13: FEO performs best across different functions of the vehicle application at $t = 2^{nd}$ minute.

6.3.2 Drone Navigation

We modify a drone navigation application [16] to be FaaS-compliant by decomposing it into two parts: stateful and stateless. Since the stateful portion of the app cannot be offloaded, we focus primarily on benefits accrued by offloading the stateless part of the drone application.

We model a cab from the SFCabs dataset as a single drone. Along the drone's route, it requests resources at its proximal site. These resources at a site are apportioned to account for fluctuations in drone arrivals. The maximum number of drones that can be supported by an Edge site is typically capped by resources available for both stateful and stateless components. However, by offloading the stateless components, FEO makes room for for the deployment of additional stateful components (and by extension additional drones). We demonstrate this in Fig. 14 which shows the P99 latencies observed for one representative timestamp. With FEO's offloading, more drones can meet the application SLO of 110ms.

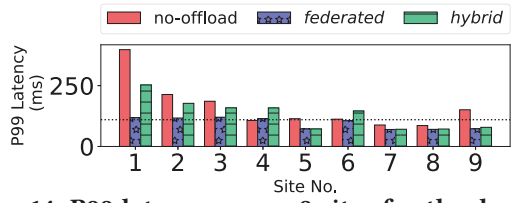


Figure 14: P99 latency across 9 sites for the drone application at $t = 2^{nd}$ minute.

7 Related Work

Resource Orchestration for Geo-Distributed Edge Sites. Prior work propose agile and flexible control planes and adopt a hybrid Cloud-Edge approach to orchestration [3, 31] for long running services on the Edge. In FEO, autonomous sites are viewed as peers which participate in tandem for resource orchestration for FaaS.

Serverless Computing. A wide variety of work has been carried out in the field of serverless computing which improve video analytics [30], optimize machine learning inference tasks [29], and improve serverless DAG scheduling [21]. Several others propose improvements for resource scheduling and systems optimizations within a FaaS platform deployment [19, 20, 27]. FEO extends the problem scope to geo-distributed environments consisting of multiple Edge sites which independently run FaaS platforms.

Load Balancers. In this work, we examined policies commonly incorporated in state-of-the-art load balancers [14, 24]. These load balancers are typically solitary deployments and therefore do not have the necessary mechanisms to interact with geo-distributed peers. FEO coordinates across multiple sites to evaluate the peer node to offload requests.

FaaS Orchestrators. Prior serverless systems have explored invocation balancing across sites using a centralized load balancer [32], and examined policies in a single-tenant environment [4]. FEO instead explores the broad applicability of various policies in past work and proposes a federated load-aware orchestrator for multi-tenant deployments.

8 Discussion and Future Work

Enhancements to *federated* Policy. FEO adopts the *federated* approach that helps meet SLOs across a range of applications. The same policy can be embellished further to reduce the impact of failed offload requests. One option is to hedge function invocations [9] to multiple offloadees, ignoring redundant results from stragglers as long as they are idempotent. Alternatively, byzantine infrastructure failures can be handled by incorporating transactions [12]. Unsuccessful offloads can be retried to other candidate offloadees as a future optimization. We leave the evaluation of such enhancements to future work.

Configurable Policies. The facility to configure new policies in a plug-and-play manner is a key consideration in the design of FEO. Adding a new policy is simple. One only needs to implement the 'Check&Enq' and 'FindTarget' functions in Fig. 4. Such flexibility allows us to consider other policy variants before settling in the *federated* approach. For example, in our analysis of spatio-temporal traffic patterns, we found short periods of time (termed "epoch") over which resource availability of a site is relatively stable. Exchanging state just at the start of such epochs could enable autonomous sites to perform state-aware offloads during the epoch. We have open-sourced our implementation for researchers to explore other policies in a similar manner.

9 Conclusion

We presented FEO, a scalable orchestrator to mitigate workload imbalance across geo-distributed Edge sites for applications using the FaaS paradigm. FEO meets application SLOs via its *federated* policy which *fails-fast* to learn about a peer's resource availability. We studied the efficacy of FEO relative to other policies via discrete-event simulation, and followed the simulation studies with a distributed systems implementation to evaluate system effects beyond policy effects using both microbenchmarks and application workloads.

10 Acknowledgements

We thank our anonymous reviewers and members of the Embedded Pervasive Lab at Georgia Tech for their insightful feedback and suggestions, which substantially improved the content of this paper. This work was funded in part by NSF CNS-1909346, Cisco, and a gift from Microsoft Corp.

References

[1] Apache OpenWhisk. Open Source Serverless Cloud Platform, 2023. https://openwhisk.apache.org/.

[2] Joy Arulraj, Abhijit Chatterjee, Alexandros Daglis, Ashutosh Dhekne, and Umakishore Ramachandran. eCloud: A Vision for the Evolution of the Edge-Cloud Continuum. *Computer*, 54(5):24–33, 2021.

[3] Giovanni Bartolomeo, Mehdi Yosofie, Simon Bäurle, Oliver Haluszczynski, Nitinder Mohan, and Jörg Ott. Oakestra: A Lightweight Hierarchical Orchestration Framework for Edge Computing. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 215–231, Boston, MA, July 2023. USENIX Association.

[4] Claudio Cicconetti, Marco Conti, and Andrea Passarella. A Decentralized Framework for Serverless Edge Computing in the Internet of Things. *IEEE Trans. Netw. Serv. Manag.*, 18(2):2166–2180, 2021.

[5] City and County of San Francisco Planning Department. Existing Commercial Wireless Telecommunication Services Facilities in San Francisco, 2021.

[6] CloudLab. https://www.cloudlab.us/, (accessed Feb, 2024).

[7] CNCF. Kubernetes. https://kubernetes.io/, 2024.

[8] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A Decentralized Network Coordinate System. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04, page 15–26, New York, NY, USA, 2004. Association for Computing Machinery.

[9] Jeffrey Dean and Luiz André Barroso. The Tail at Scale. *Communications of the ACM*, 56:74–80, 2013.

[10] FOA. FOA Reference Guide: Fiber In Communications. https://www.thefoa.org/tech/ref/OSP/nets.html, 2018.

[11] Frank Schmitz. Object Detection with Yolo, OpenCV and Python via Real Time Streaming Protocol (RTSP), 2020. https://github.com/foschmitz/yolo-python-rstp.

[12] Suyash Gupta, Sajjad Rahnama, Erik Linsenmayer, Faisal Nawab, and Mohammad Sadoghi. Reliable transactions in serverless-edge architecture. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 301–314, 2023.

[13] Adam Hall and Umakishore Ramachandran. Opportunities for Optimizing the Container Runtime. In *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*, pages 265–276, 2022.

[14] Haproxy. HAProxy: The Reliable, High Performance TCP/HTTP Load Balancer, 2023. https://www.haproxy.org/.

[15] Hashicorp. Serf: Network Coordinates, 2024. https://www.serf.io/docs/internals/coordinates.html.

[16] Samira Hayat, Roland Jung, Hermann Hellwagner, Christian Bettstetter, Driton Emini, and Dominik Schnieders. Edge Computing in 5G for Drone Navigation: What to Offload? *IEEE Robotics and Automation Letters*, 6(2):2571–2578, April 2021.

[17] Indeed. Fibtest, 2024. https://github.com/indeedeng/fibtest.

[18] Jonathan Ledlie, Paul Gardner, and Margo Seltzer. Network Coordinates in the Wild. In *4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 07)*, NSDI'07, page 22, USA, 2007. USENIX Association.

[19] Suyi Li, Wei Wang, Jun Yang, Guangzhen Chen, and Daohe Lu. Golgi: Performance-Aware, Resource-Efficient Function Scheduling for Serverless Computing. In *Proceedings of the 2023 ACM Symposium on Cloud Computing*, SoCC '23, page 32–47, New York, NY, USA, 2023. Association for Computing Machinery.

[20] David H. Liu, Amit Levy, Shadi Noghabi, and Sebastian Burckhardt. Doing More with Less: Orchestrating Serverless Applications without an Orchestrator. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1505–1519, Boston, MA, April 2023. USENIX Association.

[21] Ashraf Mahgoub, Edgardo Barsallo Yi, Karthick Shankar, Sameh Elnikety, Somali Chaterji, and Saurabh Bagchi. ORION and the Three Rights: Sizing, Bundling, and Prewarming for Serverless DAGs. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 303–320, Carlsbad, CA, July 2022. USENIX Association.

[22] Microsoft. Azure private multi-access edge compute, 2023. https://azure.microsoft.com/en-us/solutions/private-multi-access-edge-compute-mec.

[23] Anup Mohan, Harshad Sane, Kshitij Doshi, Saikrishna Edupuganti, Naren Nayak, and Vadim Sukhomlinov. Agile Cold Starts for Scalable Serverless. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, Renton, WA, July 2019. USENIX Association.

[24] NGINX. NGINX: Advanced Load Balancer, Web Server, & Reverse Proxy, 2023. https://www.nginx.com/.

[25] Klaithem Al Nuaimi, Nader Mohamed, Mariam Al Nuaimi, and Jameela Al-Jaroodi. A Survey of Load Balancing in Cloud Computing: Challenges and Algorithms. In *2012 Second Symposium on Network Cloud Computing and Applications*, pages 137–142, 2012.

[26] Michal Piorkowski, Natasa Sarafijanovic-Djukic, and Matthias Grossglauser. CRAWDAD epfl/mobility. https://dx.doi.org/10.15783/C7J010, 2022.

[27] Shixiong Qi, Leslie Monis, Ziteng Zeng, Ian-chin Wang, and K. K. Ramakrishnan. SPRIGHT: Extracting the Server from Serverless Computing! High-Performance EBPF-Based Event-Driven, Shared-Memory Processing. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 780–794, New York, NY, USA, 2022. Association for Computing Machinery.

[28] Philipp Raith, Thomas Rausch, Alireza Furutanpey, and Schahram Dustdar. faas-sim: A trace-driven simulation framework for serverless edge computing platforms. *Software: Practice and Experience*, 53(12):2327–2361, 2023.

[29] Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. INFaaS: Automated Model-less Inference Serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 397–411. USENIX Association, July 2021.

[30] Francisco Romero, Mark Zhao, Neeraja J. Yadwadkar, and Christos Kozyrakis. Llama: A Heterogeneous & Serverless Framework for Auto-Tuning Video Analytics Pipelines. In *Proceedings of the ACM Symposium on Cloud Computing*, page 1–17, New York, NY, USA, 2021. Association for Computing Machinery.

[31] Enrique Saurez, Harshit Gupta, Alexandros Daglis, and Umakishore Ramachandran. OneEdge: An Efficient Control Plane for Geo-Distributed Infrastructures. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 182–196, 2021.

[32] Christopher Peter Smith, Anshul Jindal, Mohak Chadha, Michael Gerndt, and Shajulin Benedict. FaDO: FaaS Functions and Data Orchestrator for Multiple Serverless Edge-Cloud Clusters. In *2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC)*, pages 17–25, 2022.

[33] A. Stathopoulos and M. Karlaftis. Temporal and Spatial Variations of Real-Time Traffic Data in Urban Areas. *Transportation Research Record*, 1768(1):135–140, 2001.

[34] Mengwei Xu, Zhe Fu, Xiao Ma, Li Zhang, Yanan Li, Feng Qian, Shangguang Wang, Ke Li, Jingyu Yang, and Xuanzhe Liu. From cloud to edge: a first look at public edge platforms. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 37–53, 2021.

[35] Xumiao Zhang, Anlan Zhang, Jiachen Sun, Xiao Zhu, Y. Ethan Guo, Feng Qian, and Z. Morley Mao. EMP: Edge-Assisted Multi-Vehicle Perception. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pages 545–558, New Orleans Louisiana, October 2021. ACM.