Towards Elasticity in Heterogeneous Edge-dense Environments

Lei Huang, Zhiying Liang, Nikhil Sreekumar, Sumanth Kaushik, Abhishek Chandra and Jon Weissman

University of Minnesota, Twin Cities.
Email: {huan1397,liang772,sreek012,kaush047, chandra}@umn.edu, jon@cs.umn.edu

***Abstract*—Edge computing has enabled a large set of emerging edge applications by exploiting data proximity and offloading computation-intensive workloads to nearby edge servers. However, supporting edge application users at scale poses challenges due to limited point-of-presence edge sites and constrained elasticity. In this paper, we introduce a densely-distributed edge resource model that leverages capacity-constrained volunteer edge nodes to support elastic computation offloading. Our model also enables the use of geo-distributed edge nodes to further support elasticity. Collectively, these features raise the issue of edge selection. We present a distributed edge selection approach that relies on client-centric views of available edge nodes to optimize average end-to-end latency, with considerations of system heterogeneity, resource contention and node churn. Elasticity is achieved by fine-grained performance probing, dynamic load balancing, and proactive multi-edge node connections per client. Evaluations are conducted in both real-world volunteer environments and emulated platforms to show how a common edge application, namely AR-based cognitive assistance, can benefit from our approach and deliver low-latency responses to distributed users at scale.**

***Index Terms*—edge computing, edge elasticity, heterogeneity, volunteer computing**

I. Introduction

Edge computing, a computing paradigm that brings computation closer to data sources and end-users, has enabled the deployment of emerging edge applications such as AR/VR, cognitive assistance, and autonomous vehicles. Offloading workload from devices to powerful edge servers that can run complex machine learning algorithms is necessary to resolve the device-side limitation. The demand for these latency-sensitive and compute-intensive applications is starting to increase rapidly and requires the edge to be highly available and scalable.

However, elasticity is a well-known limitation of edge resources. Edge applications are typically pinned to a single pre-deployed edge site. A burst of incoming workload can easily overwhelm an edge site causing service performance degradation. While resource allocation optimizations [1], [2] and workload adaptation strategies [3], [4], [5] can mitigate the contention and improve utilization, edge resource capacity is still a bottleneck without hardware scaling. Furthermore, geo-distributed users require wide edge availability to provide low-latency edge access. Existing public edge infrastructures [6], [7], [8] only serve major metropolitan areas, and on-premise edge deployment solutions [9], [10], [11] rely on costly/private resources to only serve certain users at specific geo-locations. Today's edge presence density [12] is far from delivering the expected geographical coverage.

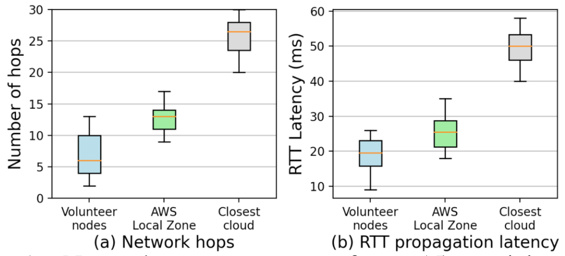

Fig. 1: Network measurements from 15 participants in Minneapolis-Saint Paul metropolitan area using home wifi network to (1) five volunteer-based edge nodes we configured with heterogeneous network access, (2) AWS Local Zone/us-east-1-msp, (3) Closest cloud/AWS us-east-2.

With the prevalanence of powerful personal PCs/laptops, we argue that sufficient compute power with low latency is already in the vicinity of end users. Volunteer resources are widely distributed with unlimited potential to scale cost-efficiently under appropriate incentive models. They tend to reside in the same local ISP networks with nearby end users, which minimizes the number of routing hops and network jitter. Figure 1 shows that nearby volunteer edge nodes can deliver lower RTT propagation delay compared to the latest AWS Local Zone infrastructure in the specified area.

In this paper, we propose using volunteer resources and geo-distributed edge resources as an enabler of elastic edge computing, and introduce the concept of a fine-grained edge resource model, namely *geo-distributed heterogeneous edge-dense environments*. This model relies on heterogeneous, unreliable yet densely-distributed commodity machines on the edge to provision latency-sensitive services. We then formulate a latency optimization problem, and present a distributed edge selection approach that leverages client-centric views to locate the best-performing edge node for each user and optimize average end-to-end latency. Specifically, we answer the following questions:

- How to identify both networking proximity and processing performance of volunteer edge nodes, considering heterogeneity, geo-location, hardware capacity and resource contention?
- How to achieve low end-to-end latency for end users in

the presence of system dynamics?
- How to guarantee continuous service in the presence of high node churn and failure rate?

Our contributions are as follows: First, we present the notion of *geo-distributed heterogeneous edge-dense environments*, a novel edge resource model, using (but not limited to) volunteer resources and non-local resources (as appropriate) as an enabler. Second, we design and implement a 2-step distributed edge selection approach that optimizes global average end-to-end latency through a simple local selection heuristic. We use a user-side performance probing strategy as a key mechanism to accurately evaluate edge candidates, with a focus on the environment heterogeneity and workload interference. Further, our approach proactively establishes multiple client-to-edge connections to provide: (i) fault tolerance by enabling an immediate connection switch to alternate edge nodes upon failure, and (ii) improved performance by switching to a better performing edge node in the presence of system dynamics.

The evaluation is conducted using an AR-based edge application in both real-world environments with 9 volunteer edge nodes, and emulation environments with 18 AWS EC2 instances in controlled node churn. The real-world results show that our approach can deliver near-optimal average end-to-end latency in a dynamic environment, and achieve 18%-46% latency reduction compared to resource-aware, locality-based and dedicated-edge-only approaches under high user demand.

II. Background and Related Work

A. Edge Computing Infrastructure

To cater to geo-proximity, strong QoS, and privacy requirements of targeting users/customers at specific areas, enterprises and organizations tend to deploy their own on-premise hardware on the edge. The logical proximity, defined as low-latency high-bandwidth communication channels between users and edge servers, is usually provided by a LAN or dedicated networking infrastructure. This category of special-purpose deployments can deliver stable latency performance, but is extremely expensive to maintain and lacks scaling capacity.

Today, major cloud providers are aggressively rolling out their public edge infrastructures [6], [7], [8]. Smaller-scale data centers are directly deployed in major metropolitan areas, providing generic VM-based computing capacity and general access to the public. The point-of-presence (PoP) of this category of edge resources is largely limited to metro areas with their numbers being only one order of magnitude higher than those for a traditional cloud. Our observations on AWS Local Zone (Figure 1) also show that its deliverable latency is much higher than the claimed single-digit millisecond level to end users due to the networking overhead within the local ISP network.

We categorize today's on-premise and public edge infrastructures into a coarse-grained edge resource model such that they are sparsely-distributed geographically with hundreds of PoPs worldwide. These edge data centers are smaller footprint extensions of the cloud relying on the same technology stack, and users at specific locations only have one (sometimes zero) or very few nearby options of edge access points to satisfy their QoS requirements.

B. Edge Elasticity and User Allocation

Edge elasticity has been improved by efficient resource allocation and service scheduling [1], [2] using workload prediction, fine-grained provisioning and proactive service deployment as common strategies. Application-specific adaptive streaming [3], [4], [5] is another technique to reduce resource contention while satisfying performance requirements by dynamically adjusting requesting rate and QoS level. However, these works cannot enable the hardware scaling capacity on the edge which is the fundamental problem constraining the edge elasticity.

In mobile edge cloud (MEC) environments with densely distributed edge nodes, studies [13], [14] have converted the elasticity problem to a user allocation problem optimizing user-to-edge assignments towards resource utilization and latency performance. However, these works take server-centric views without examining the deliverable latency performance on the user side. A distributed iterative algorithm [15] is proposed to optimize user allocations by examining user-side QoE, however, environment heterogeneity is not taken into consideration. A resource allocation scheme is applied in these works to strictly avoid workload interference and performance degradation, however, an overloaded but well-connected edge node can still provide better performance if an idle edge node suffers from the networking overhead.

We are unaware of any edge system that combines both edge-dense volunteer and dedicated nodes and geo-distributed resources simultaneously.

C. Volunteer Computing

While volunteer computing has mostly been applied to scientific computation [16], [17], it has also been proposed to enable low-latency services on the edge [18]. There have been several works [19], [20], [21] bringing conceptual designs of volunteer-based or volunteer-assist systems for latency-sensitive workloads over emulation environments. However, they lack quantitative analysis for volunteer resource characteristics, addressing networking/capacity heterogeneity, availability and reliability in real-world environments.

Multi-provider edge clouds have been proposed in both industry and academia [22], [23], [24], as a similar paradigm related to volunteer resources. It is based on a coarse-grained resource model with edge data centers as contributors, in contrast to individual devices in the volunteer computing landscape. Nebula [25] leverages geo-proximate volunteer workers to optimize MapReduce workloads, considering data locality, resource availability and bandwidth heterogeneity. It targets data-intensive batch jobs where client latency is not a concern.

D. Performance Prediction on the Edge

Edge-native applications [26] have been proposed to take advantage of edge servers, and [3] uses static profiling to obtain their processing performance before optimizing resource allocation. Proactive profiling, however, introduces significant overhead before edge nodes join the system acting as workers. Wagner and Gedeon et. al introduce a learning-based assessment framework [27] to estimate the workload performance on heterogeneous resources, and reduce profiling overhead by only performing a few runs before the prediction. However, a separate profiling process still exists with compromised accuracy. Several studies [28], [29] propose methods based on live workloads instead of test workloads to profile applications on heterogeneous hardware. However, these studies focused more on the data flow and streaming applications and are limited to data centers.

III. Problem Definition

A. Geo-distributed Heterogeneous Edge-Dense Environment

We present a fine-grained edge resource model called *geo-distributed heterogeneous edge-dense environments* that supplements deployed edge infrastructure with volunteer edge computing resources. The main benefit of this model is to substantially scale the availability of edge resources, both in terms of their geographic spread (e.g., in rural areas), as well as proximity to users (to reduce network latency). Compared to hundreds of edge data centers inherited from cloud technologies, the number of volunteer edge nodes can be orders of magnitude greater than cloud PoPs. Potentially millions of volunteer-based commodity machines are densely distributed around users contributing computing power and providing low-latency edge access, as greener complements of existing edge infrastructures discussed in Section II-A. This edge resource model has the following characteristics:

- **Heterogeneous client-to-edge networks:** Without dedicated communication channels between users and edge nodes, the number of routing hops and forwarding/propagation delays can be diverse. They are affected by distance, underlying networking infrastructures of local ISPs, protocol configurations, and real-time network condition.
- **Heterogeneous edge nodes:** Edge nodes have heterogeneous underlying hardware presenting diverse processing performance for the same workload. Compared to data centers with clusters of tightly-coupled homogeneous workers, edge nodes can be highly compute-constrained and sensitive to performance degradation due to resource contention.
- **Dense and geo-distributed resource distribution:** Massive number of edge nodes are loosely coupled yet densely distributed around users. The coverage areas of adjacent edge nodes are heavily overlapped, and users at specific areas have many nearby options towards edge access point to satisfy QoS requirements.
- **Unreliable edge node:** Volunteer edge nodes are assumed to be unreliable and unpredictable with high node churn. They can join and leave the system anytime without notifications. In addition, edge nodes can run unexpected higher priority host workloads competing with existing edge services.

We believe our model is richer and more elastic than emerging edge-dense environments such as in MEC models, where edge compute capacity co-locates with densely distributed 5G base stations. In MEC environments, dedicated edge servers, fully managed by telecom companies, tend to be highly reliable and homogeneous. The user-to-edge connectivity is mainly affected by first wireless hop characteristics [30], while the user-to-edge connectivity in our proposed model is determined by local ISP infrastructures and unpredictable networking conditions. Our model has more flexibility in deployment as it does not require resource co-location with 5G base stations.

B. Edge Application Model

We adopt a widely used edge application model consisting of the pre-deployment of application servers (e.g. image processing) to a set of edge nodes. The user selects an edge node for their requests to be out-sourced. For simplicity, we consider a single application server type in this paper, but our model can be extended to support any number of application server types. An application manager manages each application service type in the system.

C. Problem Formulation

Based on the environment definition above, we formulate a latency optimization problem. We first define the problem in a static manner without the churn of application users and edge nodes, and then discuss its generalization in a dynamic system.

Consider a geo-distributed heterogeneous edge-dense environment with n users and m edge nodes in a specified area, an *Edge Assignment (EA)* refers to a users-to-edge match that assigns each user u_i ($1 \leq i \leq n$) an edge node e_j ($1 \leq j \leq m$) to offload computation. An EA contains n pairs as shown below:

$$EA = \{< u_1, e_{j1} >, < u_2, e_{j2} >, \ldots, < u_n, e_{jn} >\}$$

We can also express the EA above from the view of edge nodes. For an edge node e_j, we use S_j to denote the set of users attached to it, namely $S_j = \{u_{i1}, u_{i2}, \ldots, u_{ik_j}\}$ where $0 \leq k_j \leq n$ and $k_1 + k_2 + \ldots + k_m = n$. Note that each user only offloads computation to one edge node while one edge node can serve multiple users concurrently. Therefore, it's equivalent to express the EA as follows:

$$EA = \{S_1, S_2, \ldots, S_m\}$$

Assuming the user u_i selects the edge node e_j under the given EA, then its performance, namely the end-to-end latency, is expressed as the sum of $D\_prop_i^j$, $D\_trans_i^j$, and $D\_proc(e_j, S_j)$. D_prop is the RTT propogation delay,

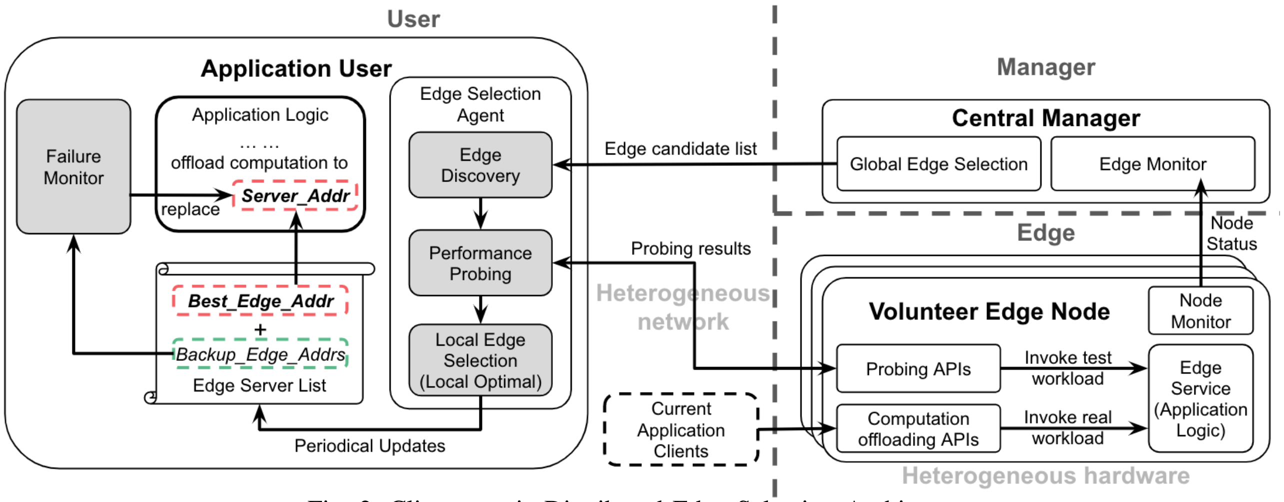

Fig. 2: Client-centric Distributed Edge Selection Architecture

D_trans stands for the data transfer delay of each request, and D_proc stands for the queuing delay and processing time within the edge node. Specifically, D_proc is determined by the heterogeneous hardware/capacity of node e_j and its current workload denoted by S_j. The resource contention caused by overloading and capacity limitation can easily become bottleneck and dominate the latency performance, leaving edge proximity not effective. We use the following expression to describe the average end-to-end latency under the given EA:

$$P(EA) = \frac{1}{n} \cdot \sum_{i=1}^{n} (D\_prop_i^{j_i} + D\_trans_i^{j_i} + D\_proc(e_{j_i}, S_{j_i}))$$

In a system with n users and m edge nodes, there are totally $\Phi = m^n$ possible EAs since each user has potentially m options towards the edge selection in an edge-dense environment. The objective is to find such an EA that minimizes the average end-to-end latency of n users:

$$\min_{EA \in \Phi} \; P(EA)$$

This is a NP-hard problem without a simple solution. Furthermore, knowledge of D_prop, D_trans and D_proc are unknown information from a central view to make global decisions. D_prop and D_trans are only subject to client-centric views. And D_proc is determined by current workload and server hardware/platform. It requires a costly profiling process to obtain the processing time under specific configurations, which is impractical to execute on every single edge node in volunteer environments.

The problem becomes more complicated when both application users and volunteer edge nodes are highly dynamic with unpredictable node churn. How to construct an efficient local search solution with effective heuristics becomes critical to reach the new optimal state faster from the current state in the presence of system dynamics.

IV. Proposed Approach

A. Architecture Overview

Given an edge-dense environment, each user may have a choice of multiple edge nodes. A key question is how to select resources for individual clients so as to meet desired performance goals such as minimizing latency. We present a distributed edge selection approach applied in a geo-distributed volunteer-based edge cloud system [31]. Figure 2 shows a simplified version of this edge cloud architecture to emphasize the edge selection process. Central Manager collects real-time node status/resource utilization information from edge nodes to serve edge discovery queries discussed below.

We use a 2-step approach: global edge selection on the manager or server side followed by local edge selection on the user side, to accurately locate the best edge candidate for each user and at the same time reduce profiling overhead. Our approach leverages client-centric views to closely monitor edge environments, predict heterogeneous edge performance, and make local edge selection decisions by using a heuristic that is geared toward optimizing average end-to-end latency. Our approach contains four critical client-centric modules (in grey boxes) as shown on the left side of Figure 2:

- **Edge discovery:** Application users send initial edge discovery queries to the Central Manager where the first step global edge selection is triggered. By examining present edge nodes on geo-proximity, resource utilization, and network affiliation, Central Manager generates a coarse-grained but narrow-ranged *candidate edge list* for the user.
- **Performance probing:** Application user probes each edge candidate on the list to address the environment heterogeneity. It collects networking metrics, information related to node processing capacity/speed and existing workload to identify overall latency performance in real time.
- **Local edge selection:** Based on accurate evaluations from the probing step, users make local decisions selecting the

best-performing candidate for computation offloading. To reduce the global average latency and preserve fairness, we use a simple heuristic to guarantee that newly accepted users should not dramatically diminish the QoS of existing users.

- **Failure monitor:** Upon node failure/leaving, failure monitor can immediately detect the connection interruption and switch upcoming compute requests to alternate edge nodes whose connections are proactively established. Failure monitor provides the robustness for unreliable volunteer resources.

B. Edge Discovery and Global Edge Selection

Edge discovery happens when new users join the system or existing users periodically query environment updates. A manager-side global edge selection process is then triggered to locate nearby edge access points for requesting users. However, manager views alone cannot entirely identify the environment heterogeneity and edge status without end-to-end network probing and real-time performance prediction. We therefore employ a 2-step approach that first generates an approximate *candidate edge list* on the manager side, followed by a local selection process in client-centric views.

The *candidate edge list* is a small subset of edge nodes that are expected to provide low latency responses for specific users. Global edge selection policies incorporate multiple factors that affect user end-to-end latency, including geo-proximity, node capacity/load, resource utilization, and optionally-provided network affiliation information between users and edge (indicating existing LAN or preferred networking channels).

We first apply a geo-proximity filter to rule out unqualified nodes, and then prioritize the local candidates based on resource availability, network affiliation and user preferences. Specifically in geo-proximity search, we use GeoHash [32] to identify a wider-range geographical area to include remote nodes which may be useful as a last resort (if all local nodes are performing poorly). Selection filters and sorting policies can be flexibly combined/modified to prioritize available edge nodes towards different application requirements. Since final decisions are eventually made on the user side, the global edge selection of our 2-step approach is coarse-grained with high tolerance to edge selection inaccuracy and mismatch.

We use the concept *TopN* to indicate the size of *candidate edge list*, referring to the top N edge nodes in the sorted list of the global edge selection results. Multiple choices can deal with both node churn and other system dynamics. This is an important configuration parameter in our approach. Larger *TopN* value brings higher accuracy and flexibility to the edge selection process by providing more edge options to end users at the local selection step. It also improves the fault tolerance by enabling users to proactively establish redundant connections to multiple edge nodes in preparation for potential node failures. However, larger *TopN* value can also bring higher probing and synchronization overhead as discussed in the following sections. In our evaluation, we examine the influences of different *TopN* values on fault tolerance, latency performance, system overhead and fairness across users.

Probing API	Description
RTT_probe()	User probes RTT propagation delay to edge nodes.
Process_probe()	User probes "what-if" processing delay in edge nodes.
Join()	User requests to attach an edge node for computation. This request can be rejected if the node state has changed.
Unexpected_join()	Upon failures, user notifies a backup edge node for joining. This request cannot be rejected.
Leave()	User notifies the current attached node of leaving. This can be caused by node switch or task finish.

TABLE I: Probing APIs exposed by edge nodes

C. Performance Probing

To accurately address the heterogeneity of real-world edge environments and avoid costly profiling overhead on various edge devices, we use a real-time probing approach to predict edge performance during runtime. Performance probing requests are initiated by application users to collect (1) end-to-end networking metrics to each edge candidate, and (2) "what-if" processing performance on these candidates if they are selected for computation offloading. Furthermore, information related to existing workloads on each edge node is also reported in probing results to optimize global average latency (discussed in section IV-D). Table I describes APIs exposed by edge nodes to interact with probing users.

1) Probe networking delay: Networking delay for an offloading request comprises RTT propagation delay (D_prop) and data transfer overhead (D_trans) introduced by constrained bandwidth. While probing D_prop is fairly easy, probing D_trans consumes available bandwidth and competes with existing networking traffic. Therefore, we only conduct D_prop probing to avoid interfering with concurrent workload. Our observations show that D_prop alone is sufficient to evaluate end-to-end connectivity for typical latency-sensitive offloading workloads such as AR-based wearable assistance. The reasons are twofold as follows.

First, edge resources can only provide low/modest improvements to bandwidth-hungry applications like video streaming [12], since edge proximity improves propagation but not necessarily transmission unless multiple single hop network options are available. We target interactive and compute-intensive workload where bandwidth is not the bottleneck compared to processing time and propagation delay. Second, for AR-based wearable assistance applications, requests carry video frames for image processing and responses are lightweight instructions related to detected objects. User outbound bandwidth usually becomes the data transfer bottleneck, which is determined by network access method/ISP configurations/traffic plans. Edge selection has limited effect on first-hop data transfer performance.

2) Probe processing delay: A user offloads requests to the selected edge node for real-time processing, and we take the compute time of single requests as the value of D_proc. To avoid time-consuming profiling and to improve the accuracy of performance prediction, we invoke a *test synthetic workload* to simulate “new-user-join” scenarios. The test workload is based on the same application logic and compute requirements as the real offloading task. Its invocation generates the “what-if” performance upon new user/request arrivals, which reflects specific hardware/platform configurations and resource contention due to existing workload. Taking the AR application as an example, the test workload is image processing for a single synthetic video frame with standard image size.

To minimize the overhead, "what-if" performance is cached in the node memory responding to probing queries, and we only invoke the test workload to update the “what-if” performance upon node state changes. There are three scenarios for which a test workload is invoked: (1) A new user is attached to an edge node - workload increase. (2) An existing user stops offloading requests or switches to another edge node - workload decrease. (3) Performance monitor in edge nodes reports noticeable change of processing time under the same number of attached users. This can be caused by adaptive request rate (e.g. varying FPS for AR applications) or higher priority host workloads running on volunteer resources that are out of our control.

Offloading request rates for different users can dynamically change according to networking conditions, processing speed and user preferences, which leads to varying amount of workload under the same number of attached users. The performance monitor trigger enables the system to deal with workload dynamics in a fine-grained manner. The more aggressively test workload is triggered, the more accurately the performance is predicted along with higher overhead.

When one of the above scenarios occurs, new “what-if” performance is measured and updated in the cache. Upon receiving $Process\_probe()$ requests, edge node only reads the cache to fetch the “what-if” performance instead of invoking a new test workload. Therefore, a large number of probing requests do not necessarily lead to more test workload invocations and higher overhead.

3) Join selected edge node: We add up RTT propagation delay from $RTT\_probe()$ and “what-if” processing time from $Process\_probe()$ as the predicted performance of edge candidates. Based on the performance prediction and selection policies, each user selects a candidate to start offloading workload. Since probing requests are conducted asynchronously, a selection conflict happens when multiple users simultaneously select the same edge node. Probing predictions are inaccurate in this case since the "what-if" performance only reflects the one-additional-user scenario. Therefore, we use the $Join()$ interface to synchronize edge selection procedures, and users only start offloading workloads after their $Join()$ requests are accepted by selected edge nodes.

Join synchronization is implemented using $seqNum$ which incrementally identifies the node state change. $seqNum$ is updated along with test workload invocation in critical sections. They share the same three triggers discussed in IV-C2. Algorithm 1 shows the $Join()$ interface which corresponds to the first trigger type - user join. Note that the test workload is invoked asynchronously in a separate thread with some delay (line 5), since it should reflect the “what-if” performance after this new accepted user is officially attached. This delay is set to be two times the common user RTT propagation to make sure the test workload is invoked after the arrival of the new user offloading requests.

Algorithm 1 Join() API exposed by edge nodes (Critical section locks are hidden for simplicity)

```
1: function JOIN($User\_seqNum$)
2:     $seqNum$ ← Current sequence number maintained by this edge node
3:     if $seqNum$ = $User\_seqNum$ then
4:         $seqNum$ += 1
5:         New_Thread(InvoketestWorkload())
6:         return True
7:     else    ▷ State has changed since last probing request
8:         return False
9:     end if
10: end function
```

Algorithm 2 Client-centric performance probing procedure

```
1: $C$ ← Edge candidate list (size > 1)
2: $Current$ ← Currently attached edge node for processing
3: $Backups$ ← Current backup edge node list
4: for $c \in C$ do
5:     $start$ ← time.Now()
6:     $c$.RTT_probe()
7:     $D\_prop$ ← time.Now() - start
8:     $(D\_proc, c.seqNum, c.state)$ ← $c$.Process_probe()
9:     $c$.probing_result ← $D\_prop$ + $D\_proc$
10: end for
11: SortLocalSelectionPolicy(C)
12: if $Current \neq C[0]$ then    ▷ A better performing edge is found
13:     $success$ ← $C[0]$.Join($C[0].seqNum$)
14:     if not $success$ then    ▷ This join request is rejected
15:         Repeat probing process from edge discovery step
16:     end if
17:     $Current$.Leave()
18:     $Current$ ← $C[0]$.address
19: end if
20: $Backups$ ← C[1:].addresses
```

Algorithm 2 shows the user-side edge selection procedure with join synchronization. After collecting the probing results (line 4 - 10), local edge selection policies are applied to sort edge candidates. We discuss local selection policies in Section IV-D. The first edge node in the sorted list is the best-performing candidate. If it is the same node as the currently-used node for offloading, no action is required besides updating the backup edge list (used by failure monitor in section IV-E). If a better candidate is found, then $Join()$ request is sent to this selected node. Upon join acceptance, users call

$Leave()$ to notify the previous edge node of leaving, which triggers the test workload invocation and $seqNum$ update.

D. Local Edge Selection

In this section, we discuss local edge selection policies applied in $SortLocalSelectionPolicy()$ (Algorithm 2 line 11). We use LO_j to represent the probing result or the Local-view Overhead of edge candidate j. The edge node with smallest LO is the Best Local Candidate (BLC) which can deliver the lowest latency performance:

$$LO_j = D\_prop_{probing} + D\_proc_{probing}$$

$$BLC = \min_{j=1}^{TopN} LO_j$$

However, this selection scheme neglects the interference to existing workloads on each candidate. Assume $D\_proc_{current}$ is the processing delay for existing attached users on an edge node, and $D\_proc_{probing}$ is the "what-if" processing delay on this edge node, then $D\_proc_{probing} - D\_proc_{current}$ is the performance degradation that can be experienced by all existing users if one additional user is accepted to join. To optimize the average performance, we should consider both the local and global views to minimize the end-to-end latency for all users. We use GO_j to denote the Global Overhead of edge candidate j (n is the number of existing users on candidate j):

$$GO_j = n \times (D\_proc_{probing} - D\_proc_{current}) + LO_j$$

Note that the information of existing workloads on each candidate is shared with the requesting user at line 8 in Algorithm 2. Therefore, we have the following expression to optimize the global average performance:

$$BLC = \min_{j=1}^{TopN} GO_j$$

In common scenarios, LO and GO are positively correlated towards affecting both individual performance and global average performance, since an edge candidate with multiple existing users tends to have high LO value (performance degradation caused by overloading) and high GO value (larger n in GO equation) at the same time.

Our framework based on LO and GO can be easily modified to optimize edge applications with specific QoS requirements. Users can first filter out edge candidates whose LO violates QoS requirements and then select the node with lowest GO to optimize global performance. In this case, new users can be rejected to join the system if (1) no available edge nodes can satisfy the QoS requirements, or (2) new joins lead to QoS violations of existing users.

E. Failure Monitor

Static user-to-edge assignment is susceptible to failures, since volunteer edge resources are unreliable with high node churn. Failure monitor detects the connection failures and immediately replaces the edge node by looking up the backup edge list to guarantee continuous service. Backup edge list is comprised of unselected edge nodes from the *candidate edge list*. It is pre-sorted based on the local selection policies discussed in section IV-D, therefore the first backup node used is always the second best option. The backup edge list is periodically updated by the performance probing step as shown in Algorithm 2, and connections to backup edge nodes are proactively established to make the service downtime during connection switch negligible.

Specifically, the probability of users still experiencing node failures in our approach is determined by the following two parameters:

- **TopN value:** $TopN$ is the size of *candidate edge list* generated by the global edge selection process. $TopN - 1$ is the size of backup edge list after each update of probing requests. Larger $TopN$ value adds more backup edge nodes maintained on the user side, and therefore enables users to be more fault tolerant to simultaneous edge failures.
- **Probing period $T_{probing}$:** $T_{probing}$ is the time period between two consecutive edge discovery/performance probing requests. It indicates the frequency with which users query the Application Manager for environment changes. The smaller $T_{probing}$, the more frequent the backup edge list gets updated, during which failed edge nodes get replaced with alive ones. Therefore, smaller $T_{probing}$ brings higher robustness.

As a tradeoff, higher $TopN$ and smaller $T_{probing}$ also bring higher overhead to the system by increasing the number of asynchronous operations (detecting, probing and updating). Based on the level of node churn and reliability of volunteer resources [33], $TopN$ and $T_{probing}$ can be modified accordingly.

V. Evaluation

We evaluate our approach using an AR-based cognitive assistance application, in both real-world and emulated environments, to show its validity in achieving elastic edge computing with volunteer resources.

A. Application: AR-based Cognitive Assistance

AR-based cognitive assistance is one promising category of edge-native applications [26] that can fully exploit the potential of edge resources. Its core building block, real-time object detection, is highly compute-intensive with strict latency requirements to deliver a satisfying user experience.

We integrate the proposed distributed edge selection approach into a cognitive assistance application that helps visually impaired people to identify objects. Users constantly send video frames to edge servers at a max rate of 20 FPS (which can adaptively decrease based on the network and processing performance). All video frames have the standard size of 0.02 MB after encoding, with the same compute requirements. After queuing, decoding and processing of the frame, lightweight cognitive assistance instructions (negligible size) are returned back to users based on the detected objects and application-specific assistance logic. The test workload in our approach is configured to process one pre-loaded synthetic video frame on edge nodes.

B. Evaluation Baselines

We use the global average end-to-end latency collected on the user side as the evaluation metric to determine the overall elasticity performance in geo-distributed volunteer environments.

We contrast the proposed distributed edge selection approach with the following performance baselines:

- **Geo-proximity**: Locality-based edge selection policy is widely applied in coarse-grained edge data center environments. Users are assigned to their closest edge nodes geographically to offload the computation. The latency between users and edge nodes is assumed to be proportional to the distance, and resource capacity is not considered to be the bottleneck.
- **Resource-aware Weighted Round Robin**: This is a common edge selection and load balancing policy used in fine-grained multi-edge environments. Multiple edge servers are present in the local area, and incoming user requests are forwarded to the most available edge nodes in a weighted round robin fashion. The weight applied for each edge node is determined by the resource availability and utilization.
- **Dedicated-only Edge Infrastructure**: Dedicated-only edge refers to the existing edge infrastructure with limited PoP and resource capacity. In our experiments, we use AWS Local Zone with a static number of EC2 instances to emulate this category of resources. We add this performance baseline in our real-world experiments to illustrate the utility of leveraging volunteer resources as a complement to existing dedicated edge infrastructure.
- **Closest Cloud**: The offloading performance on the closest AWS cloud is used as a baseline reference in our real-world experiments to show the latency reduction achieved by various edge policies/infrastructures.

C. Real-world Experiments

We first show the adaptability and overall effectiveness of our approach in real-world volunteer environments.

1) Experimental Setup: We recruit 20 participants with heterogeneous access network. They are all within 10 miles away from each other in Minneapolis-Saint Paul metropolitan area, with 15 participants being users and 5 participants using their laptops registered as volunteer edge nodes. We also register 4 additional AWS Local Zone instances into our volunteer platform to illustrate that our approach can also be applied to form a hybrid edge cloud consisting of both volunteer and dedicated resources. We use these 4 instances alone to demonstrate the dedicated-only scenario for comparisons.

Node	Processor	Processing
V1	Intel® Core™ i7-9700, 8 cores	24ms
V2	Intel® Core™ i7-2720, 6 cores	32ms
V3	Intel® Core™ i9-8950HK, 6 cores	31ms
V4	Intel® Core™ i5-8250U, 4 cores	45ms
V5	Intel® Core™ i5-5250U, 2 cores	49ms
D6 - D9	AWS Local Zone t3.xlarge	30ms
Cloud	AWS ec2 t3.xlarge	30ms

TABLE II: Real-world experiments setup

Table II shows the hardware details of all participants in the system, along with their heterogeneous processing performance for a single application video frame.

2) Single-user View: Figure 3 shows the end-to-end latency perceived by a random user to 3 volunteer nodes and 1 AWS Local Zone instance. We can see that volunteer nodes (V1 and V2) can deliver better performance compared to dedicated nodes (D6) in real-world environments. This is because V1 and V2 have lower network latency to the user as compared to D6, thus reducing the overall latency.

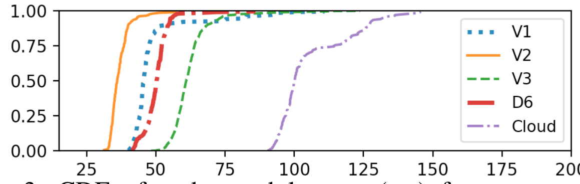

Fig. 3: CDF of end-to-end latency (ms) from a user to 4 different edge servers in real-world environments.

Client	V1	V2	V3	V4	V5	D6	Cloud
U1	<u>**38**</u>	47	49	65	72	42	107
U2	43	<u>**35**</u>	56	58	61	45	102
U3	49	50	45	59	71	<u>**42**</u>	112

TABLE III: Pairwise end-to-end latency (ms) between 3 random users and edge nodes in real-world environments ($TopN$ = 6). The underlined numbers indicate the selected nodes

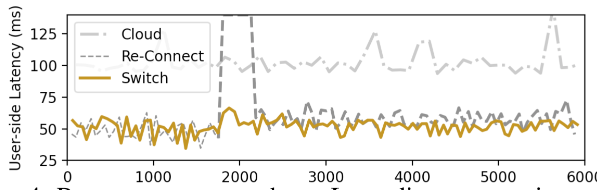

Fig. 4: Re-connect approach vs. Immediate connection switch performance trace over timeline (ms).

Table III shows the pairwise latency performance between 3 random users and edge nodes, along with selection results using the proposed distributed edge selection approach. The experiment is conducted separately for 3 users to avoid interference. We configure $TopN = 6$ on each user to guarantee that all volunteer edge nodes are selected in the *candidate edge list*. Therefore, the performance probing approach probes all edge nodes in the system to collect probing results. Best-performing nodes are accurately selected for 3 users, addressing the networking and processing heterogeneity.

Figure 4 shows the end-to-end latency performance trace for consecutive video frames upon node failure. While a re-connection approach experiences a large service downtime to re-discover an alternative edge node upon failure, our approach can immediately switch to a backup edge node maintaining the continuous service.

3) Edge Elasticity: To explore the elasticity performance of the proposed approach, we incrementally let all 15 application users join the system to evaluate the global average end-to-end latency. Figure 5 shows the average performance for our approach (Client-centric) vs. 4 baselines in V-B.

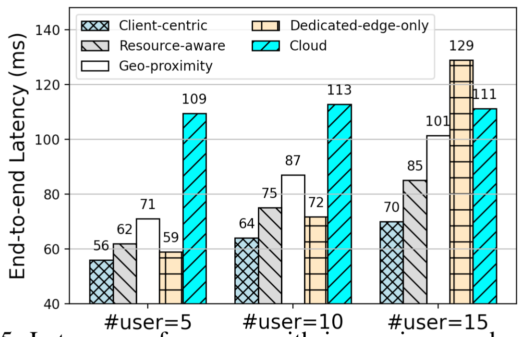

Fig. 5: Latency performance with increasing number of users in real-world environments ($TopN = 3$)

As we can see in the figure, our approach can better balance the load with increasing number of attached users. Locality-based approach selects the closest nodes without examining the actual networking connectivity and processing capacity. Resource-aware approach checks the resource utilization, however, the heterogeneous hardware/networking environments lead to unexpected performance. Dedicated-only scenario lacks hardware scaling flexibility upon increasing workload, which results in a worse-than-cloud performance at #user = 15 due to significant performance degradation caused by overloading.

D. Emulation Experiments

We conduct emulation experiments in AWS cloud to simulate a wider geographical distribution of users/nodes, with flexibly controlled users/nodes numbers, networking performance and more importantly node churn in a dynamic environments. We discuss two experiment configurations in the following sections.

1) Static edge nodes with increasing number of users: In the first emulation experiment, we incrementally add randomly-distributed users into the system one by one, as a way to examine each user's behavior towards the edge selection and performance degradation due to resource contention. We keep the number of edge nodes and distribution static for simplicity.

Experimental setup: We set up 9 volunteer edge nodes (4 $\times$ t2.medium, 4 $\times$ t2.xlarge, 1 $\times$ t2.2xlarge) and 15 application users (15 $\times$ t2.micro), such that they are simulated to be within 50 miles away from each other. We configure the pairwise networking performance (latency/bandwidth) using tc with real-world measurement data. Specifically, the RTT propagation delay is between the range of 8 to 55ms in the corresponding geo-distribution.

Performance trace: Figure 6 shows the latency performance trace of each individual user using three edge selection methods. 15 users join the system one after another every 10 seconds, indicated by vertical lines in the figure.

While locality-based method (a) should have performed better in a wider range geo-distribution, it lacks the flexibility upon resource contention. A few users in (a) exceed 150ms latency due to the overload of local nodes, when an idle and relatively far-away node can deliver better end-to-end latency.

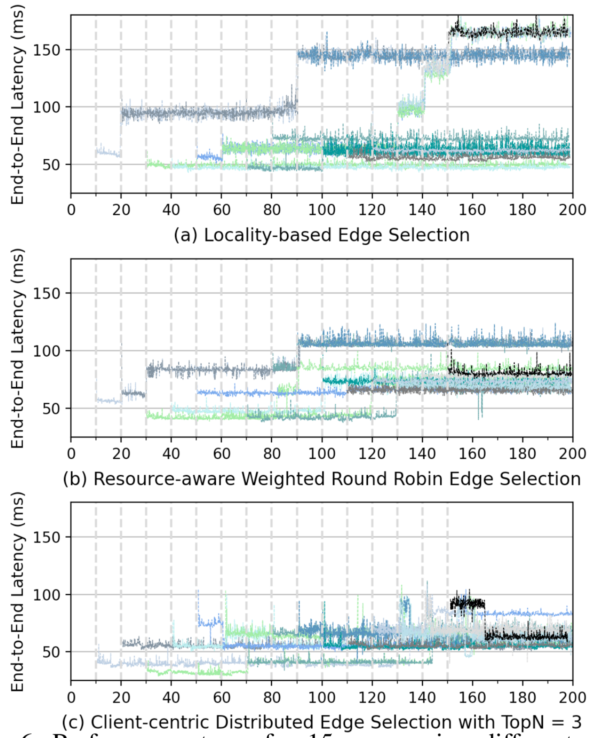

Fig. 6: Performance trace for 15 users using different edge selection methods. 10 edge nodes are static with users incrementally joining the system (every 10 seconds)

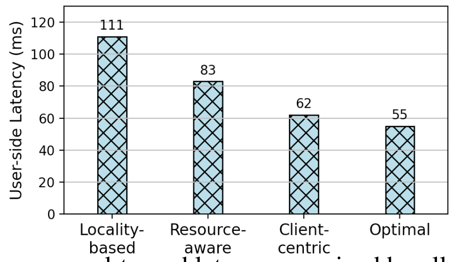

Fig. 7: Average end-to-end latency perceived by all users using different edge selection approaches

(b) performs better than (a) since resource contention is fairly balanced to all edge nodes. However, resource-aware selection cannot identify the network heterogeneity between users and nodes to tradeoff resource availability and faster networking channel. Our approach (c) can assign all users a low-latency edge node by combining networking and processing performance probing. And we can see that dynamic load balancing happens due to the proactive multi-node (TopN) connections, where some of the users switch their selected edge nodes during the processing when a better option is found upon varying workloads.

Comparison to optimal selection: We next compare the different edge selection approaches to optimal selection. Figure 7 shows the average performance after all users join the system (after 150 seconds in Figure 6). We calculate the optimal edge assignment for this specific configuration based on the application profile on three types of EC2 instance we use and the emulated network setup. The figure shows that our approach has about 12% higher latency than the optimal,

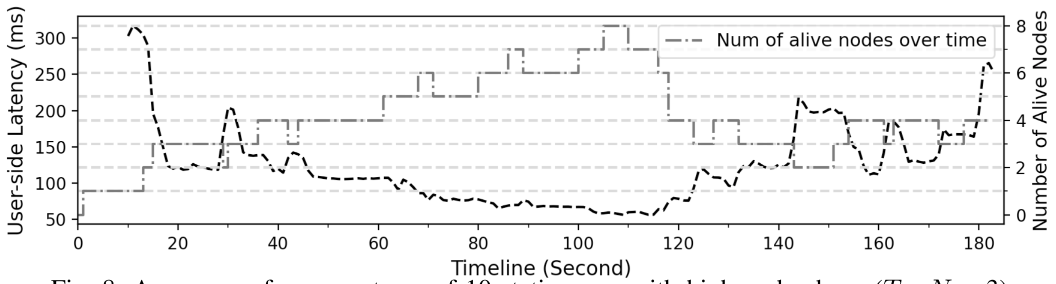

Fig. 8: Average performance trace of 10 static users with high node churn ($TopN$ = 3)

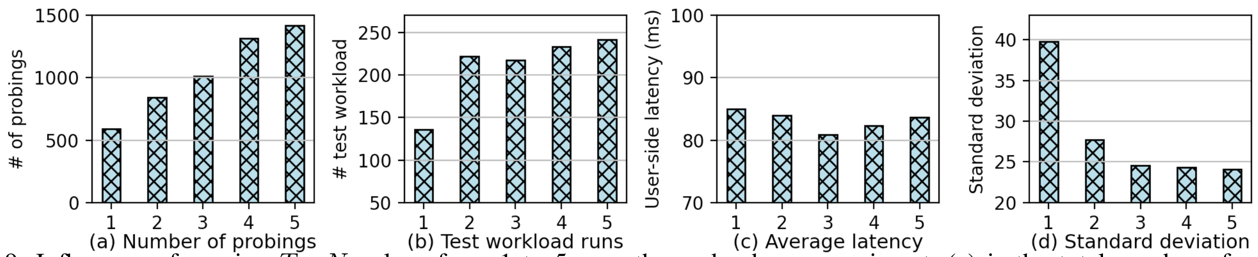

Fig. 9: Influences of varying $TopN$ values from 1 to 5 over the node churn experiment. (a) is the total number of probing requests sent by all users. (b) is the number of test workload invocations by all users. (c) is the average latency calculated between 60 -120 seconds. (d) is the latency standard deviation across all users indicating system fairness

as compared to 102% and 51% higher respectively for the locality-based and resource-aware selection approaches.

2) Static users with high edge node churn: In the second emulation experiment, we apply a high node churn to edge nodes (along with distribution changes) to examine dynamic load balancing behavior of our approach. We configure different *TopN* values to explore its influence on the fault tolerance, system overhead, user-side latency and fairness performance. The number of users is static for simplicity.

Experimental setup: To model the node churn of volunteer edge nodes, we assume that the probability of nodes joining the system every 30 seconds follows the Poisson distribution (k = 4 edge nodes). Arriving nodes are randomly assigned a timestamp (second) in each 30 seconds period. And the lifetime of edge nodes is modeled using Weibull distribution (average lifetime = 50 seconds). We randomly select a configuration from multiple runs of this process, which results in a total of 18 edge nodes over a 3-minute timeline as shown in Figure 8 (grey stair line). Then we randomly match 18 simulated edge nodes with 18 AWS ec2 instances (8 × t2.medium, 8 × t2.xlarge, 2 × t2.2xlarge) which are configured in the same way as section V-D1 with 10 static users (10 × t2.micro).

Performance trace: Figure 8 shows the average performance of 10 static users ($TopN$ = 3) at each timestamp, along with the number of alive nodes over time based on our node churn model. It gives us the correlation between average performance and edge resource availability. Whenever new edge nodes join the system (upward steps), the average latency correspondingly decreases within seconds. This implies effective dynamic load balancing since users can immediately discover and switch to newly joined edge nodes, benefiting from the periodic performance probing approach. When edge

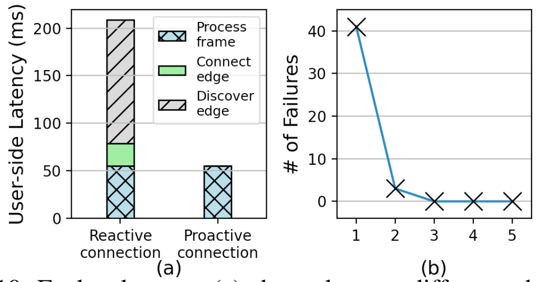

Fig. 10: Fault tolerance. (a) shows latency difference between proactive and reactive connections upon failure. (b) shows the number of edge failures with varying $TopN$ value.

nodes leave the system (downward steps), the average latency does increase but there is no service disruption, since a second best backup edge node can immediately take over the workload without any service downtime to users.

Effect of TopN: Based on the same node churn model, we run the experiment multiple times using different $TopN$ values to explore its influences on the following factors:

System overhead: The system overhead of probing is primarily due to the number of test workload runs on the edge nodes. Figure 9 (a) shows that the number of probing requests increases linearly with $TopN$ value, however, the increase in number of test workload invocation is much smaller (Figure 9 (b)). This is because test workload is invoked upon 3 scenarios referring to node state changes, while probing requests only access the cache to fetch the historical data.

Overall performance: Figure 9 (c) shows that the overall end-to-end latency for different $TopN$ values are fairly close with $TopN$ = 3 slightly better than the others. They have similar performance since they can all benefit from the periodic performance probing mechanism. Even $TopN$ = 1 can obtain a fresh edge node every period of time and compare its performance with the current node to decide if a better-performing option is available. Higher $TopN$ can enhance this benefit by probing more options to locate the best choice. However, there is a point of diminishing returns as overhead begins to outpace any benefit beyond $TopN$ = 3. In our experiments, $TopN$ = 3 has worked well.

Fairness: Figure 9 (d) shows the standard derivation of average performance. Higher values refers to higher performance variation and lower fairness across all the users. While all 5 $TopN$ values can deliver similar latency performance, lower $TopN$ values lead to weak system fairness due to less probing options during the selection process.

Fault tolerance: Figure 10 (a) shows the latency difference between reactive connection (re-connect) and proactive connection (our approach) upon node failure. We regard the re-connect situation as a failure since it leads to an unacceptable delay gap for latency-critical applications. It only happens when all backup edge nodes fail simultaneously in our approach.

Figure 10 (b) shows the number of failures experienced by all users under different $TopN$ values. $TopN$ = 1 stands for 0 backup edge nodes on the user side, which corresponds to common user-to-edge policies. We can see that $TopN = 2$ can dramatically reduce the number of failures with 1 backup edge node in our node churn model. Starting at $TopN = 3$, the number of failures can be reduced to 0.

VI. Conclusion

In this paper, we gave a detailed discussion of volunteer resource characteristics in the edge computing landscape, and proposed the use of *geo-distributed heterogeneous edge-dense environments*. We used this new edge resource model to formulate a latency optimization problem. We then presented a distributed edge selection solution that leverages client-centric views to identify the environment heterogeneity and optimize global average end-to-end latency. We validated the effectiveness of the proposed approach in both real-world and emulation environments, and showed that our approach can deliver near-optimal performance in a dynamic environment.

References

[1] C. Nguyen, C. Klein, and E. Elmroth, "Multivariate lstm-based location-aware workload prediction for edge data centers," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2019.

[2] ——, "Elasticity control for latency-intolerant mobile edge applications," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*, 2020.

[3] J. Wang, Z. Feng, S. George, R. Iyengar, P. Pillai, and M. Satyanarayanan, "Towards scalable edge-native applications," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, 2019.

[4] J. Gedeon, M. Wagner, K. Skaisgiris, F. Brandherm, and M. Mühlhäuser, "Chameleons on cloudlets: Elastic edge computing through microservice variants," *CoRR*, vol. abs/2105.10355, 2021.

[5] Z. Feng, S. George, H. Turki, R. Iyengar, P. Pillai, J. Harkes, and M. Satyanarayanan, "Improving edge elasticity via decode offload," 2021.

[6] Amazon Web Service. (2022) Aws local zone. [Online]. Available: https://aws.amazon.com/about-aws/global-infrastructure/localzones/

[7] Microsoft Azure. (2022) Azure edge zone. [Online]. Available: https://azure.microsoft.com/en-us/solutions/private-multi-access-edge-compute-mec/

[8] Amazon Web Service, "Aws wavelength," 2022. [Online]. Available: https://aws.amazon.com/wavelength/

[9] ——. (2022) Aws outposts. [Online]. Available: https://aws.amazon.com/outposts/

[10] Google Cloud. (2022) Google anthos. [Online]. Available: https://cloud.google.com/anthos

[11] Microsoft Azure. (2022) Azure stack edge. [Online]. Available: https://azure.microsoft.com/en-us/products/azure-stack/edge/

[12] M. Xu, Z. Fu, X. Ma, L. Zhang, Y. Li, F. Qian, S. Wang, K. Li, J. Yang, and X. Liu, *From Cloud to Edge: A First Look at Public Edge Platforms*. New York, NY, USA: Association for Computing Machinery, 2021, p. 37–53.

[13] P. Lai, Q. He, M. Abdelrazek, F. Chen, J. G. Hosking, J. C. Grundy, and Y. Yang, "Optimal edge user allocation in edge computing with variable sized vector bin packing," *CoRR*, vol. abs/1904.05553, 2019.

[14] P. Lai, Q. He, G. Cui, X. Xia, M. Abdelrazek, F. Chen, J. G. Hosking, J. C. Grundy, and Y. Yang, "Edge user allocation with dynamic quality of service," *CoRR*, vol. abs/1907.11580, 2019.

[15] P. Lai, Q. He, G. Cui, F. Chen, M. Abdelrazek, J. Grundy, J. Hosking, and Y. Yang, "Quality of experience-aware user allocation in edge computing systems: A potential game," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*.

[16] The Folding@home Consortium. (2022) Folding@home.

[17] D. P. Anderson, "Boinc: a platform for volunteer computing," *Journal of Grid Computing*, 2020.

[18] T. M. Mengistu and D. Che, "Survey and taxonomy of volunteer computing," *ACM Comput. Surv.*, 2019.

[19] T. M. Mengistu, A. Albuali, A. Alahmadi, and D. Che, "Volunteer cloud as an edge computing enabler," in *International Conference on Edge Computing*, 2019.

[20] B. Ali, M. Adeel Pasha, S. u. Islam, H. Song, and R. Buyya, "A volunteer-supported fog computing environment for delay-sensitive iot applications," *IEEE Internet of Things Journal*, 2021.

[21] S. Alonso-Monsalve, F. García-Carballeira, and A. Calderón, "A heterogeneous mobile cloud computing model for hybrid clouds," *Future Generation Computer Systems*, 2018.

[22] B. C. Şenel, M. Mouchet, J. Cappos, O. Fourmaux, T. Friedman, and R. McGeer, "Edgenet: A multi-tenant and multi-provider edge cloud," in *Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking*, ser. EdgeSys '21.

[23] EDJX. (2021) Edjx. [Online]. Available: https://edjx.io/

[24] Mutable. (2021) Mutable. [Online]. Available: https://mutable.io/

[25] A. Jonathan, M. Ryden, K. Oh, A. Chandra, and J. Weissman, "Nebula: Distributed edge cloud for data intensive computing," *IEEE Transactions on Parallel and Distributed Systems*, 2017.

[26] M. Satyanarayanan, G. Klas, M. Silva, and S. Mangiante, "The seminal role of edge-native applications," in *2019 IEEE International Conference on Edge Computing (EDGE)*.

[27] M. Wagner, J. Gedeon, K. Skaisgiris, F. Brandherm, and M. Mühlhäuser, "Poster: An assessment framework for edge applications," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*.

[28] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," *ACM SIGPLAN Notices*, 2013.

[29] H. Ma, Z. Zhou, and X. Chen, "Leveraging the power of prediction: Predictive service placement for latency-sensitive mobile edge computing," *IEEE Transactions on Wireless Communications*, 2020.

[30] A. Narayanan, X. Zhang, R. Zhu, A. Hassan, S. Jin, X. Zhu, X. Zhang, D. Rybkin, Z. Yang, Z. M. Mao, F. Qian, and Z.-L. Zhang, "A variegated look at 5g in the wild: Performance, power, and qoe implications," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021.

[31] L. Huang, Z. Liang, N. Sreekumar, S. K. Vishwanath, C. Perakslis, A. Chandra, and J. B. Weissman, "Armada: A robust latency-sensitive edge cloud in heterogeneous edge-dense environments," *CoRR*, vol. abs/2111.12002, 2021.

[32] Z. Balkić, D. Šoštarić, and G. Horvat, "Geohash and uuid identifier for multi-agent systems," in *KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*.

[33] J. Sonnek, M. Nathan, A. Chandra, and J. Weissman, "Reputation-based scheduling on unreliable distributed infrastructures," in *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*.