Measuring the Edge: A Performance Evaluation of Edge Offloading

Heiko Bornholdt*, Kevin Röbert*, Martin Breitbach†, Mathias Fischer*, and Janick Edinger*
*Universität Hamburg, Department of Informatics, Germany
Email: {heiko.bornholdt, kevin.roebert, mathias.fischer, janick.edinger}@uni-hamburg.de
†University of Mannheim, Business School, Germany
Email: martin.breitbach@uni-mannheim.de

***Abstract*—When the demand for computationally-intensive applications exceeds the capabilities of a single device, computation offloading can help to leverage remote resources. Currently, most systems only use cloud and grid resources as offloading targets, ignoring the vast amount of end-user devices that might contribute with their idle CPU cycles. Even though these edge devices have great potential, two main reasons make their usage more complicated: First, the devices are highly heterogeneous and unreliable. Second, numerous middleboxes on the Internet impose barriers, making communication between end-user devices difficult. Thus, in this paper, we overcome these two obstacles and propose an offloading system composed not only of cloud and grid resources but also of geographically distributed and heterogeneous end-user devices. We deploy this system in realistic environments and evaluate its performance under real-world conditions. Our results indicate that offloading to edge devices owned by end users can compete with cloud and grid offloading while adding minimal communication overhead. Our empirical findings support the hypothesis that edge computing can be a cost-efficient alternative to traditional offloading systems.**

***Index Terms*—distributed computing, edge computing, peer-to-peer computing, heterogeneous networks**

I. Introduction

There is an increasing amount of computationally-intensive applications, e.g., image processing, simulation, machine learning, optimization, or virtual and augmented reality. Their resource demands often exceed the capabilities of a single device and can significantly benefit from distributed processing on multiple machines. For this purpose, developers and users typically rent cloud and grid resources, which are costly and need to be set up in advance. As an alternative, P2P offloading can be applied in which workload is transferred to end-user devices to use their idle CPU cycles. These devices may reside within companies, public institutions, or private households. In their entirety, they can be considered as a large, distributed computing cluster. However, using these distributed resources is complex for two main reasons: First, these devices are heterogeneous regarding their hardware, software, connectivity, availability, and reliability. Where cloud and grid resources are standardized, dedicated, and reliable, end-user devices per se do not provide a uniform computing platform. End-user devices are less predictable and might leave the system at any time. Sophisticated, context-aware scheduling mechanisms are required to ensure a quality of service similar to that of cloud and grid computing [1]. Providing this awareness requires a comprehensive understanding of the behavior of devices and their respective environments (edge, grid, and cloud). Second, despite the Internet being envisioned as a network that allows any-to-any communication between each set of peers, the reality could hardly be more different [2]. The presence of middleboxes, e.g., firewalls, network address translators (NATs), and internet gateway devices (IGDs) makes edge devices hard or impossible to reach. As a result, up to 87% of peers in today's popular P2P networks remain unreachable from the Internet without any further countermeasures [3]. Applied to P2P computation offloading systems, these unreachable peers would not be available as offloading targets.

Using end-user devices in a P2P offloading system and evaluating their performance compared to traditional offloading scenarios (cloud & grid) is the main contribution of this paper. To accomplish this goal, we provide solutions for the immense heterogeneity of end-user devices and the obstacles mentioned above to communication between devices. Regarding the heterogeneity, we extended the *Tasklet system* [4] to our use case. This distributed computing system provides an abstraction for computationally-intensive workloads and allows seamless migration of these workloads to process-level virtual machines running on arbitrary devices. We added an overlay network constructed by multiple middlebox traversal mechanisms to reach as many edge devices as possible. As a result, we could evaluate the distributed processing of multiple applications in a real-world P2P computing environment and compare its performance to cloud and grid offloading scenarios.

The remainder of this paper is structured as follows. We present our approach to include heterogeneous end-user devices in computation offloading architectures in Sec. II. We present a real-world evaluation in Sec. III and discuss our results in Sec. IV. We assess related work in Sec. V before we conclude the paper in Sec. VI and provide an outlook on future work.

II. A P2P Computation Offloading System

A. Computation Offloading with Tasklets

Our architecture is based on the *Tasklet system* [5] – a distributed computation middleware. The system architecture is structured as a hybrid P2P system (see Fig. 1) and consists

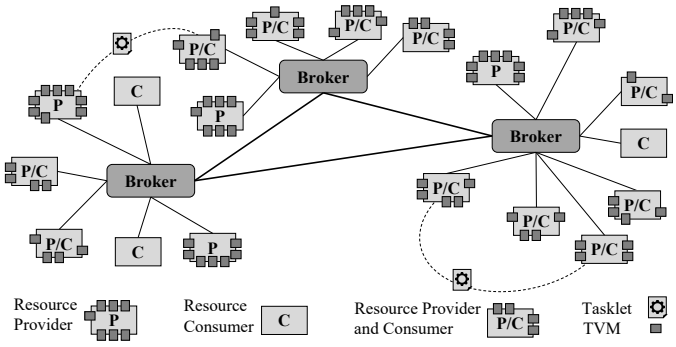


Fig. 1. Tasklet system architecture. Providers (P) share their idle resource capacities with consumers (C). The brokers perform resource management and matchmaking. The computational workload is distributed as Tasklets and executed on provided TVMs.

of three core components: *providers*, which allow other devices to use their unused resources; *consumers*, which make use of these resources; and *brokers*, which act as resource managers and perform the matchmaking between providers and consumers. Depending on their workload, devices may be consumers at one time and providers at another. Consumers offload their workload in the form of Tasklets, i.e., self-contained units of computation at the granularity of function calls, which have a typical execution time between hundreds of milliseconds up to a few minutes. Consumers and providers run the Tasklet middleware, making communication transparent to application developers. Tasklets are executed on process-level virtual machines. These *Tasklet Virtual Machines* (TVMs) provide a secure and isolated runtime environment that also abstracts from a device's underlying heterogeneous hardware-software configuration. The Tasklet system recognizes that resource providers can vary in reliability, availability, and performance and introduces the concept of quality of computation (QoC) to ensure execution guarantees [4]. This QoC concept applies context-aware scheduling and is necessary for applications that require a reliable or timely execution, as opposed to best-effort computing. The Tasklet system's mechanisms help extend serverless computing, such as the FaaS programming model, to edge systems.

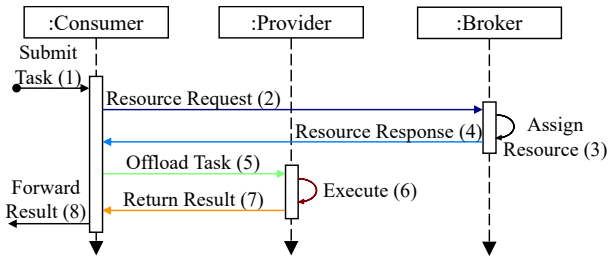


Fig. 2. Tasklet life cycle.

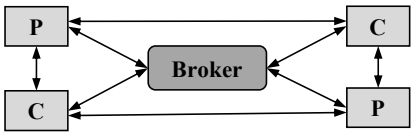
Fig. 2 shows the life cycle of a Tasklet: (1) An application requires additional computing power and passes a Tasklet to the local running Tasklet middleware, which (2) sends a resource request to the broker. (3) The broker selects the most suitable provider for this request, and (4) returns this information to the consumer. (5) The consumer forwards the Tasklet directly to the provider, which (6) executes the Tasklet on one of its TVMs. Similarly, the provider (7) returns the execution result to the consumer, whereas (8) the local middleware passes it to the application.

B. Overcome Network Barriers

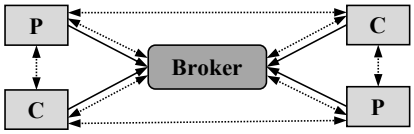
Providers, consumers, and brokers need to communicate with each other to perform computation offloading. Ideally, this would mean that the network structure is the same as the Tasklet system architecture shown in Fig. 1. This structure would allow providers and consumers to register with a broker of their choice and exchange Tasklets and results. However, this ideal network structure cannot always be applied to the Internet. It would be necessary for each host to be equipped with a public routable IP address and to communicate bidirectionally with others. However, as a collection of independent subnetworks, the Internet is covered with communication barriers between subnetworks. Middleboxes such as firewalls, NATs, and IGDs make it difficult or impossible to establish connections to some hosts. These middleboxes result in some hosts being able to connect to others but not being able to be reached via connections initiated by others. The mentioned unilateral restriction affects especially mobile, residential, or corporate networks, rendering the devices within these networks unreachable. Applying these barriers to the Tasklet system architecture would mean that brokers must be operated on reachable public hosts. In addition, providers and consumers in edge environments may be able to register with this broker but cannot exchange Tasklets or results.

Our solution to this problem for the Tasklet system is implementing the overlay network concept. On top of the Internet's physical network structure that does not best fit our requirements, we constructed an overlay that matches the ideal network structure (see Fig. 3). Within this overlay, providers, consumers, and brokers are equipped with routable IP addresses and can establish connections bilaterally. A naive approach to creating such an overlay would be introducing one always-present public reachable relay server. With this approach, all providers, consumers, and brokers would register at this relay server through which all communication would be routed. Besides the availability and scalability concerns this relay server would introduce, it also negatively affects computation offloading performance. Relaying would increase latency which can render Tasklet offloading unviable. Therefore, we created an overlay using Interactive Connectivity Establishment (ICE) [6] to render previously unreachable hosts reachable. To achieve this, ICE combines several well-known middlebox traversal techniques. ICE works by having two devices initiate a connection to a known third device (server), which shares each device's network endpoint with the other. The devices then use this information to communicate directly with each other by creating a temporary hole in their middlebox firewall. The hole allows incoming traffic from the other device, allowing direct communication between the two

devices without requiring a third device to be involved for relaying. The ICE technique is performed before computation offloading starts, creating an overlay network structure allowing all providers, consumers, and brokers to reach each other. Once the overlay is established, the Tasklet system can offload as usual without additional communication overhead.



(a) Consumers (C) and providers (P) can contact the broker and vice versa. Consumers can contact providers and vice versa.



(b) Without overlay, only the broker can be contacted (solid arrows). Dashed arrows indicate overlay-enabled communication matching the ideal network shown in (a).

Fig. 3. Ideal communication network (a) and real communication network enhanced with overlay (b). Arrows indicate who can initiate communication.

III. Evaluation Methodology

We evaluate the performance of our extended offloading architecture, i.e., the Tasklet system, under real-world conditions in edge, grid, and cloud environments to answer the following questions:

- $Q1$ To what extent can P2P communication be efficiently used for offloading in edge computing?
- $Q2$ What is the variance of the edge communication latency in our real-world setting?
- $Q3$ How do the answers to the previous questions depend on the application type (CPU- vs. network-intensive)?

A. Devices

We used 16 heterogeneous devices at 7 locations in Germany, around Hamburg, Frankfurt, and Mannheim (see Fig. 4). One device acted as a consumer that offloaded Tasklets to 15 providers. In addition, two other hosts served as broker and relay server respectively. The devices differed in type (desktop or mobile system), internet connection (cellular or wired), and communication restrictions imposed by the network the device belongs to (e.g., firewalls or IGDs blocking inbound connections, NATs render hosts unroutable, ISP policies blocking P2P connections at all). While devices without middlebox-imposed restrictions can be reached directly, devices with restrictions must be made reachable with our overlay first. Some middleboxes restrict any P2P connections, which forces communication to be made via a public relay server as a last resort. The devices were placed in different environments, which allowed us to evaluate the performance of the Tasklet system in different scenarios. Providers were either placed in the cloud, a local or remote grid, in private households (residential), or in cellular networks as mobile devices.

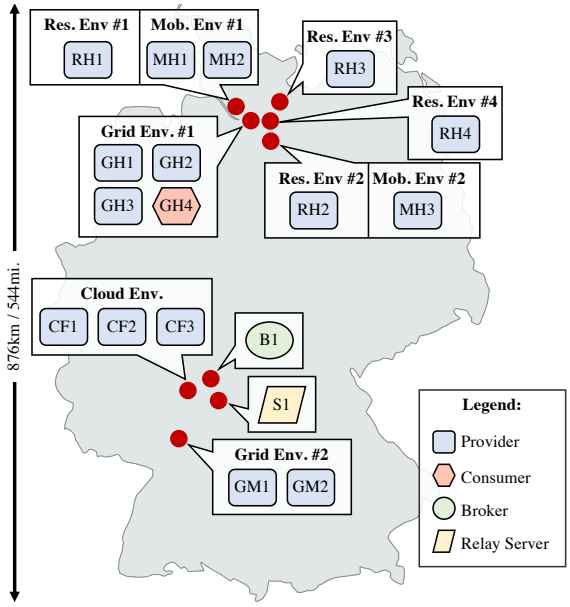


Fig. 4. Location of all environments and nodes with their respective roles.

In the following, we refer to the devices by systematic identifiers. For consumers and providers, the first letter represents the environment (`C` for Cloud, `R` for Residential, `G` for Grid, `M` for Mobile). The second letter indicates the location (`H` for Hamburg, `F` for Frankfurt, `M` for Mannheim). `B` stands for broker and `S` for the relay server (see Fig. 4).

The cloud environment (`CF1`–`CF3`) was located in the AWS data center in Frankfurt. All devices in this cloud environment have been provided with a public IP address and configured with no firewall, allowing these devices to be directly routable from any other device. The residential environment devices (`RH1`–`RH4`) were distributed in and around Hamburg and equipped with a broadband internet connection. Each residential environment network is isolated from the Internet through an IGD applying network address translation and blocking all inbound connection attempts. Therefore, to allow bilateral connection establishment with devices in these residential environments, our overlay must be used to overcome the barrier imposed by the IGD. The filtering policies applied by the IGD allow P2P connection establishment. The first grid environment was located in a university network in Hamburg (3 providers, `GH1`–`GH3`; 1 consumer, `GH4`). A second grid was placed in a university network in Mannheim (2 providers, `GM1`–`GM2`). Each university network is isolated from the Internet through a network-wide firewall blocking all inbound connections. Within a grid, devices can reach each other. Bilateral connection establishment with devices from the Internet is only possible using our overlay. There were also two mobile environments with three providers (`MH1`–`MH3`). Both mobile environments are equipped with a carrier-grade NAT, giving each mobile device a private unroutable IP address and preventing any inbound connection attempts. In addition, all mobile devices are isolated from each other, making P2P

communication between these devices impossible. Our overlay can make these devices reachable from the Internet, but P2P connections between two mobile devices are prevented by the strict filtering policies of the carrier-grade NAT. For mobile-to-mobile device communication, relaying through B1 must be used. The broker B1 and relay server were located near Frankfurt, Germany, in data centers operated by DigitalOcean and Vultr. Both are configured with a public routable IP address and no firewall.

B. Scenarios

We evaluated seven scenarios, each representing an individual experiment. In each scenario, we used a subset of providers for execution, except for the *Combined* scenario, where all available resources were used. In the following, we briefly present each scenario.

Cloud: This scenario resembles traditional cloud offloading. We rented three Amazon EC2 c5.xlarge instances in Frankfurt's same data center/availability zone (CF1–CF3). Each host runs four TVMs and thus can run at most four tasks simultaneously. As there are no network barriers, e.g., firewalls, in this scenario, our consumer GH1 can directly communicate with the providers. In this scenario, our overlay did not need to make any hosts reachable.

Residential: This scenario models typical residential setups (RH1–RH4). Each environment is located near Hamburg, Germany, and within different IGDs. This means that our overlay has to make all providers reachable first. Otherwise, the providers would not have been available as offloading targets. The relaxed filtering policies applied by the residential gateways allow P2P communication.

Grid-Local: In this scenario, a grid environment in an institute or company network is assumed (GH1–GH4). All devices – including the resource consumer – are located within the same LAN. Therefore this environment represents the ideal environment for P2P communication as there are no network barriers on any side.

Grid-Relayed: The Grid-Relay scenario is a variation of the Grid-Local environment. Here, all LAN-based communication between peers is intentionally disabled. This will result in all communication being relayed through our public server S1. Thus, this environment allows us to compare the impact of the absence of our overlay, as consumers and providers can not communicate directly.

Grid-Remote: Here, the consumer (GH4) is located in another location than the grids' providers (GM1 and GM2). These locations provide a high-quality Internet connection, but due to institute/company-wide IGDs, the providers must first be reachable via our overlay.

Mobile: This scenario will evaluate offloading in mobile environments (MH1–MH3). With Deutsche Telekom and O2, we cover two of three of Germany's available three Tier 1 mobile carriers. Typically, mobile environments are very P2P-hostile, as mobile operators deploy symmetric NATs that prevent ICE from creating any direct link between two mobile devices. Therefore, communication between mobile devices must be relayed. In addition, due to the cellular connection, this environment can suffer from significant disruptions and jitter. Hence, this environment represents the worst-case scenario for computation offloading.

Combined: This scenario combines the Cloud, Residential, Grid-Remote, Residential, and Mobile environments. With this scenario, we want to evaluate how highly heterogeneous environments impact our results. This scenario introduces the risk that we can observe negative performance on overall completion time by stragglers.

C. Applications

During the experiment, 16,800 tasks were executed across seven scenarios. Each task was split into 12 individual Tasklets and was repeated 100 times. The result of a task is returned to the application once all 12 Tasklet results have been obtained.

The offloading experiments have been conducted with four different types of applications. They are divided into two CPU-intensive (option pricing & a color key filter) and two bandwidth-intensive applications (image convolution & ray tracing). This spectrum allows us to determine the influence of network connectivity or computational power on offloading performance. For space reasons and because all four application types show the same trend, we present only two applications in this paper: *Option Pricing* and *Image Convolution*.

Option Pricing: This task depicts a CPU-intensive computation with a low network load using a Monte Carlo method to price a European call option. The compiled code of this task is around 5kB in size, takes two integers as input parameters, and outputs a single floating-point number. Therefore, the load on the network can be neglected for this task type.

Image Convolution: We implemented an image convolution filter to exemplify tasks with high network load at moderate CPU usage. This filter constructs a 9x9 matrix for each pixel to detect images' edges. Since this task uses images with equal resolution as both input and output parameters, the load on the network is higher and strongly dependent on the size of the image. In our experiment, the image was 1867 pixels wide and 1050 pixels high, resulting in 2 million pixels and 6 million RGB values that needed to be transferred twice (source input image and filtered output image) per computation.

IV. Evaluation Results

Fig. 5 and Fig. 6 summarize the results of our experiments for the CPU-intensive (left) and bandwidth-intensive (right) tasks. The figure shows the average transmission times observed for offloading the task and returning the result. Transmissions were routed directly through P2P connections or via the relay server S1, depending on the actual environment. Since computation times strongly depend on the CPU performance of the respective device, we only focus on transmission-related operations (offloading a task from a resource consumer to a provider and returning the result) that give us insights into environment-related performance. We will now present and discuss the outcomes of our results:

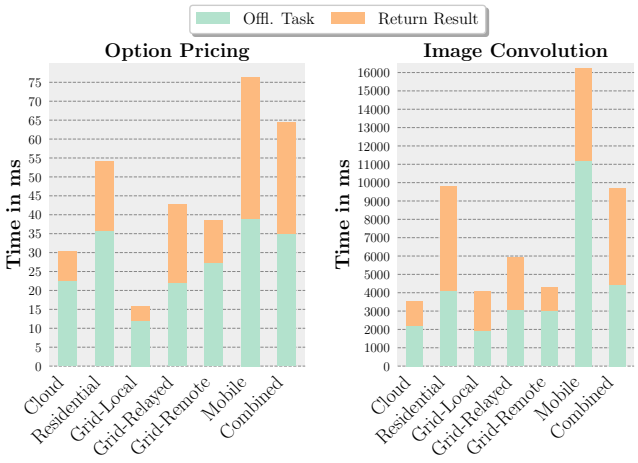


Fig. 5. Average transmission times in milliseconds for a Tasklet of a CPU-intensive (left) and bandwidth-intensive (right) application.

Option Pricing (CPU-intensive): The Cloud environment performs very well, even though the providers are hundreds of miles away from the consumer. This performance can be attributed to the cloud servers' large and reliable symmetrical network connection. However, there is a lower performance bound due to the network latency caused by long distances.

The Residential environment's median is slightly, and the mean is 50% higher than the cloud scenario. Here – regarding $Q2$ – we can see a more significant variance in performance, which is attributed to the heterogeneity of the residential devices. While some stages were completed within 5 ms, others took over a minute due to stragglers. This problem with stragglers can be circumvented by offloading the same task to several consumers in parallel and returning the first available result. Therefore, applying this scheduling strategy should increase performance, showing us that P2P-connected edge resources can achieve competitive performance ($Q1$).

It can be observed that the Grid-Local environment achieves the best overall performance. This result aligns with expectations, as this environment is optimal for offloading. Grid-Local is twice as fast as the Cloud and shows that (with comparable CPU performance) an increase in performance is possible on locality-aware computation offloading systems.

The results of the Grid-Relayed scenario show the influence of missing P2P connections compared to Grid-Local and therefore give more insights to answer $Q1$: Since all communication must be detoured via a public relay server, performance is generally lower here (about 22ms instead of 11ms).

The Grid-Remote scenario performs worse than the Cloud and better than the Residential environment. Apart from a few stragglers, the execution time has a low variance due to the heterogeneity of the devices present in the grid. Furthermore, this scenario demonstrates that edge computing can be applied to institutional resources, providing competitive advantages over traditional cloud resources.

The Mobile scenario has the worst mean/median performance of all scenarios. This is also in line with expectations regarding cellular communication, the resulting network latency, and the sharing of network capacities with all carrier customers of a radio cell. It is noticeable here that the proximity of consumers and providers does not improve performance.

In the Combined scenario, all providers of the Cloud, Residential, Grid-Remote, and Mobile environments were combined. It can be seen that the overall performance is comparatively poor, even though this scenario provides the most resources. This is due to the previously mentioned splitting of the tasks into 12 Tasklet and the fact that the stragglers here have a major impact on the performance. This problem can be avoided or reduced using a different scheduler that favors faster devices or parallel offloading.

Image Convolution (Bandwidth-intensive): Due to the fast connection between the consumer and the providers, the cloud environment is again a well-performing environment for data-insensitive tasks. Therefore, to refer to $Q3$, this environment is not strongly affected by the type of application.

The Residential scenario is slower than the Cloud, and the task offloading stage shows a higher variance in the completion times, giving us insights for $Q2$. Four clusters can be spotted on the histogram for the return result stage. Each cluster represents one of our four residential providers in this scenario. Hence, the asymmetrical providers' downlink and uplink significantly affect these environments, especially when the result is returned. The uplink will strongly influence the offloading task stage if the consumer is also placed in a residential environment.

Grid-Local has the best performance for the offloading task stage and the second-best total performance. While times are comparable to the Cloud environment, it can be seen that stragglers negatively influence the performance of the result return.

Particularly in this Grid-Relayed environment, it can be seen that relaying traffic impacts network times significantly. For example, the time needed to offload a task is increased by 33%, returning the result by 46% compared to the direct traffic observed in the Grid-local scenario.

In the offload task stage of the Grid-Remote environment, we are about 1000 ms slower in mean and almost 900 ms slower in median compared to the Grid-local environment. This decrease in performance is because the providers are now several hundred kilometers/miles away from the consumer.

The mobile scenario also represents the weakest-performing environment in these task types. Compared to the CPU-intensive Tasklet, we can see a considerable variation in the offloading task stage times ($Q2$). This result is consistent with our expectation that the varying quality of radio communications significantly impacts bandwidth-intensive tasks. In contrast, the response result does not show such a significant deviation. This observation can be explained by the more restricted uplink of the asymmetric cellular connection.

The combination of the different environments can also be seen in this scenario. It should be noted that – compared to returning the result – the offloading stage is slightly faster due to some devices' asymmetrical uplink.

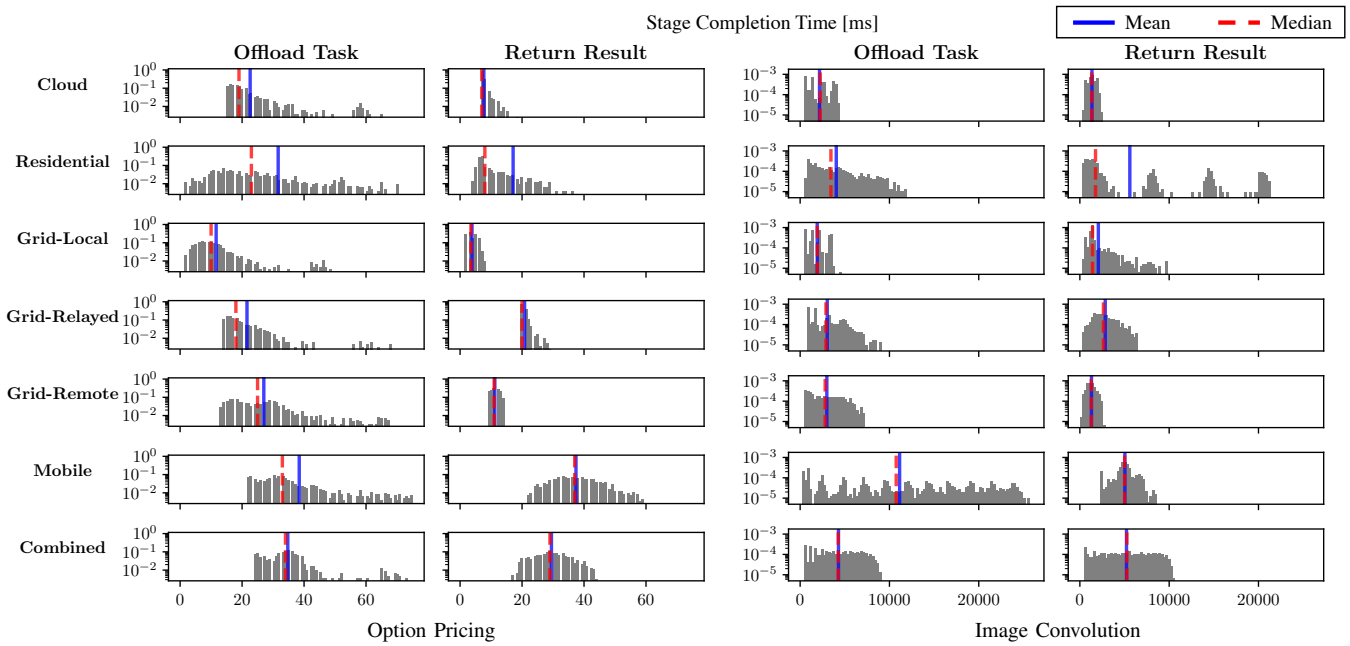


Fig. 6. Histograms showing the density of the completion times in milliseconds of the individual life cycle stage in each scenario.

V. Related Work

Computation Offloading systems that can not only use cloud and grid resources but also (unused) edge devices have various requirements. We will discuss these requirements, divided into computation offloading, communication, and various requirements, as shown in Table I.

Edge-enabled computation offloading systems require accountability to provide incentives [7], payment models [8], and to identify malicious entities. Therefore every activity must be indisputably traceable to an entity such as in [7]–[10].

While accounting is important, the core of any computation offloading framework is the scheduler – it decides whether, when, how, and where to offload. Offloading decisions can be static [4], [13], dynamic [11], [12], [14], [16], or trained [9]. Contextual information is needed to make these decisions, which can vary widely from, e.g., sensor data [18], CPU utilization, energy consumption, or network connectivity [17].

Whereby these approaches are often tailored to specific use cases and environments such as mobile cloud computing [9], [12]–[14], [18], [11], [19], web browsers [16], or limited to the immediate vicinity [15]. These can be generalized by using QoS parameters as in [4] and [17].

Openness and heterogeneity in our context should not be limited to how and by whom a system can be used and accessed. A heterogeneous spectrum of devices, applications, and networks should be supported. Systems such as [4] and [17] allow resources from any network area, like the edge, cloud, or cloudlets, and fulfill the requirements for open and heterogeneous systems.

Also, latency is crucial [23] to determine if offloading is beneficial. Thus efficient routing must take place, which can be supported by additional context information, i.e., bandwidth, latency, jitter, packet loss rate, or connection type (e.g., cellular, Wi-Fi, stationary). Approaches like [9], and [17] use

TABLE I
Overview of related computation offloading systems.
(Comm. = Communication, Misc. = Miscellaneous)
(no circle) not fulfilled, ◐ partially fulfilled, or ● fulfilled

		Comp. Offl.					Comm.			Misc.	
System / *Author*	**Year**	**Accounting**	**Context-Aware Scheduling**	**Quality of Service Support**	**Openness**	**Heterogeneity**	**Context-Aware Routing**	**Decentralized Identity**	**Resilience and Robustness**	**Real-World Testbed**	**P2P-enabled Applications**
BOINC [7]	2004	●				●			●	●	
Cloudlet [11]	2009		◐			◐	◐		◐	●	
MAUI [12]	2010		◐			◐			◐	●	
CloneCloud [13]	2011		◐			◐			◐	●	
ThinkAir [14]	2012		◐			◐			◐	●	
Serendipity [15]	2012		◐		◐	◐			◐	●	
Nebula [16]	2014		◐		●	◐	◐		●	●	
CloudAware [9]	2015	◐	◐			◐	●	◐	◐	●	◐
Tasklet [4]	2016	◐	◐	●	●	●			●	●	◐
SDFog [17]	2016		●	●	●	●	●				◐
EMCO [18]	2018		◐			◐			◐	●	
ULOOF [19]	2018		●		◐	◐				◐	
Tang et al. [10]	2020	◐	◐	◐		●			◐		
MRLCO [20]	2020		◐	●		◐					
DMRO [21]	2021		◐			◐					
MLOOF [22]	2021		◐			◐				◐	
Our System		◐	◐	●	●	●	●	●	●	●	◐

context-aware routing to find the optimal path to a device and reduce overhead. The ICE protocol is almost transparent for applications resulting in minimal required changes that need to be applied to the Tasklet system. We chose ICE because it works independently of the middleboxes, combines many traversal techniques and can establish direct connections. This is important for us because a terminal itself cannot always control how the current network is configured and direct connections have low latencies. Other approach would be to relay communication through third parties which increases latency, decreases availavbility and security and imposes the createion of single point of failures.

Offloading systems often abstract from real-world conditions, such as unreliable nodes or links, changing network conditions, non-public nodes, and other network barriers. These conditions can significantly complicate P2P computation offloading and make existing approaches unsuitable for real-world situations. A computation offloading system is P2P-capable whenever it allows any-to-any communication between devices. Thus, it must be open like [16], [4], and [17] and provide techniques (e.g., hole punching, rendezvous search, relayed communication) to overcome network barriers like [9]. Mostly, only one or none of these requirements is fulfilled, which unnecessarily narrows the application context. Moreover, these approaches commonly form closed-membership networks that enforce certain role models, where resources are restricted to cloud [7], [11]–[14]. Edge devices are often only consumers and cannot be resource providers.

VI. Conclusion

This paper proposed a computation offloading system that uses cloud, grid, and (idle) edge computation resources. For this, our system had to address two main challenges: First, the heterogeneity of hardware, software, connectivity, and reliability of the devices involved had to be overcome to create a unified computing platform. Second, the Internet is covered with P2P communications barriers that must be traversed, especially in edge environments. This system was then deployed in a real-world setup consisting of peers placed in multiple cloud, residential, grid, and mobile environments. We could show that residential and grid resources are marginally slower than the cloud regarding network times. These performance results make them cost- and energy-efficient alternatives. In particular, we demonstrated that the additional resources from the edge positively impact the performance of applications. Hence, these edge resources can be a cost-effective alternative to cloud resources. In addition, our experiments have shown that the absence of P2P communications can significantly degrade the overall application performance by up to a factor of 5. Our evaluation results help better to understand different edge environments in terms of offload performance. In particular, our data can be used for future scheduling algorithms to make better assumptions about expected offloading runtimes and thus improve overall performance.

References

[1] B. Sotomayor, K. Keahey, and I. Foster, "Combining batch execution and leasing using virtual machines," in *ACM International Symposium on High Performance Distributed Computing*, 2008.

[2] B. Ford, "Unmanaged internet protocol: Taming the edge network management crisis," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 34, 2004.

[3] S. Henningsen, M. Florian, S. Rust, and B. Scheuermann, "Mapping the interplanetary filesystem," in *IEEE IFIP Networking Conference*, 2020.

[4] D. Schäfer, J. Edinger, S. VanSyckel, J. M. Paluska, and C. Becker, "Tasklets: Overcoming heterogeneity in distributed computing systems," in *IEEE International Conference on Distributed Computing Systems Workshops*, 2016.

[5] D. Schäfer, J. Edinger, J. M. Paluska, S. VanSyckel, and C. Becker, "Tasklets: "better than best-effort" computing," in *IEEE International Conference on Computer Communication and Networks*, 2016.

[6] A. Keranen, C. Holmberg, and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal," RFC Editor, RFC 8445, 2018.

[7] D. Anderson, "Boinc: a system for public-resource computing and storage," in *IEEE/ACM International Workshop on Grid Computing*, 2004.

[8] J. Edinger, S. VanSyckel, C. Krupitzer, J. M. Paluska, and C. Becker, "Developing a qos-based tasklet trading system," in *IEEE International Conference on Pervasive Computing and Communication Workshops*, 2014.

[9] G. Orsini, D. Bade, and W. Lamersdorf, "Cloudaware: Towards context-adaptive mobile cloud computing," in *IFIP/IEEE International Symposium on Integrated Network Management*, 2015.

[10] J. Tang, R. Yu, S. Liu, and J.-L. Gaudiot, "A container based edge offloading framework for autonomous driving," *IEEE Access*, vol. 8, 2020.

[11] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE perv. Computing*, vol. 8, 2009.

[12] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *ACM International Conference on Mobile Systems, Applications, and Services*, 2010.

[13] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *ACM Conference on Computer Systems*, 2011.

[14] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *IEEE International Conference on Computer Communications*, 2012.

[15] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling remote computing among intermittently connected mobile devices," in *ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2012.

[16] M. Ryden, K. Oh, A. Chandra, and J. Weissman, "Nebula: Distributed edge cloud for data intensive computing," in *IEEE International Conference on Cloud Engineering*, 2014.

[17] H. Gupta, S. B. Nath, S. Chakraborty, and S. K. Ghosh, "Sdfog: A software defined computing architecture for qos aware service orchestration over edge devices," 2016.

[18] H. Flores, P. Hui, P. Nurmi, E. Lagerspetz, S. Tarkoma, J. Manner, V. Kostakos, Y. Li, and X. Su, "Evidence-aware mobile computational offloading," *IEEE Transactions on Mobile Computing*, 2018.

[19] J. L. D. Neto, S.-Y. Yu, D. F. Macedo, J. M. S. Nogueira, R. Langar, and S. Secci, "Uloof: A user level online offloading framework for mobile edge computing," *IEEE Transactions on Mobile Computing*, 2018.

[20] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[21] G. Qu, H. Wu, R. Li, and P. Jiao, "Dmro: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing," *IEEE Transactions on Network and Service Management*, 2021.

[22] L. N. Ferreira, J. M. Nogueira, and D. F. Macedo, "Mloof: Multi-level online offload framework for iot devices and smartphones," in *IEEE Latin-American Conference on Communications*, 2021.

[23] V. Bahl, "Emergence of micro datacenter (cloudlets/edges) for mobile computing," in *Microsoft Devices & Networking Summit*, 2015.