Towards Scalable Edge-Native Applications

Junjue Wang
Carnegie Mellon University
junjuew@cs.cmu.edu

Ziqiang Feng
Carnegie Mellon University
zf@cs.cmu.edu

Shilpa George
Carnegie Mellon University
shilpag@cs.cmu.edu

Roger Iyengar
Carnegie Mellon University
raiyenga@cs.cmu.edu

Padmanabhan Pillai
Intel Labs
padmanabhan.s.pillai@intel.com

Mahadev Satyanarayanan
Carnegie Mellon University
satya@cs.cmu.edu

ABSTRACT

Latency-sensitive *edge-native* applications may be the key to commercial success of edge infrastructure. However, success in the form of widespread deployment of such applications poses its own challenges. These applications are edge-dependent by definition, and therefore cannot simply fail over to the cloud if the edge is overloaded. In this paper, we propose an adaptation-based strategy to allow scaling up the number of concurrent edge-native applications on a resource-limited cloudlet and wireless network. We demonstrate up to 40% reduction in offered load with minimal impact on latency on a variety of cognitive assistance tasks over non-adaptive approaches. Our approach is able to gracefully degrade and maintain quality of service for a subset of applications in the face of severely loaded conditions.

CCS CONCEPTS

• **Computer systems organization → n-tier architectures**; *Real-time system architecture*; • **Human-centered computing → Ubiquitous and mobile computing systems and tools**.

KEYWORDS

Wearable Cognitive Assistance, Cloudlet, Edge Computing, Resource Management, Mobile Computing, Gabriel

ACM Reference Format:
Junjue Wang, Ziqiang Feng, Shilpa George, Roger Iyengar, Padmanabhan Pillai, and Mahadev Satyanarayanan. 2019. Towards Scalable Edge-Native Applications. In *SEC '19: ACM/IEEE Symposium on Edge Computing, November 7–9, 2019, Arlington, VA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3318216.3363308

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
SEC '19, November 7–9, 2019, Arlington, VA, USA
© 2019 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-6733-2/19/11.
https://doi.org/10.1145/3318216.3363308

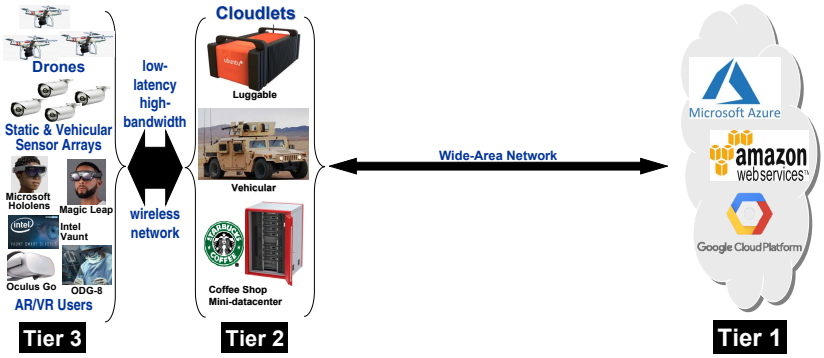
1 Introduction

Elasticity is a key attribute of cloud computing. When load rises, new servers can be rapidly spun up. When load subsides, idle servers can be quiesced to save energy. Elasticity is vital to scalability, because it ensures acceptable response times under a wide range of operating conditions. To benefit, cloud services need to be architected to easily scale out to more servers. Such a design is said to be "cloud-native."

In contrast, edge computing has limited elasticity. As its name implies, a cloudlet is designed for much smaller physical space and electrical power than a cloud data center. Hence, the sudden arrival of an unexpected flash crowd can overwhelm a cloudlet and its wireless network. Since low end-to-end latency is a prime reason for edge computing, shifting load elsewhere (e.g., the cloud) is not an attractive solution. *How do we build multi-user edge computing systems that preserve low latency even as load increases?* That is our focus.

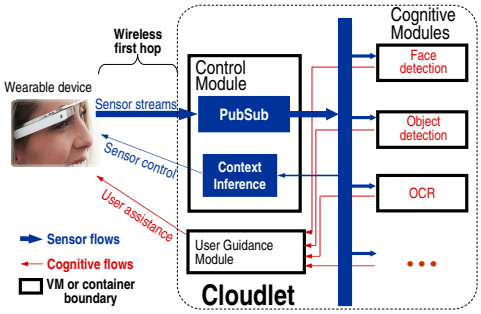
Our approach to scalability is driven by the following observation. Since compute resources and wireless capacity at the edge cannot be increased on demand, the only paths to scalability are (a) to reduce offered load, or (b) to reduce queueing delays through improved end-to-end scheduling. Otherwise, the mismatch between resource availability and offered load will lead to increased queueing delays and hence increased end-to-end latency. Both paths require the average burden placed by each user on the cloudlet and the wireless channel to fall as the number of users increases. This, in turn, implies *adaptive application behavior* based on guidance received from the cloudlet or inferred by the user's mobile device. In the context of Figure 1, scalability at the left is achieved very differently from scalability at the right. The relationship between Tier-3 and Tier-2 is *non-workload-conserving*, while that between Tier-1 and other tiers is workload-conserving.

With rare exceptions, reducing offered load is only possible with application assistance. Scalability at the edge is thus only achievable for applications that have been designed with this goal in mind. We refer to applications that are specifically written for edge computing as *edge-native applications.* These applications are deeply dependent on services that are only



From left to right, this tiered model represents a hierarchy of increasing physical size, compute power, energy usage, and elasticity. Tier-3 represents IoT and mobile devices; Tier-2 represents cloudlets; and Tier-1 represents the cloud. We use "Tier-2" and "cloudlet" interchangeably in the paper. We also use "Tier-3" to mean "mobile or IoT device."

Figure 1: Tiered Model of Computing



(Source: Chen et al [4])

Figure 2: Gabriel Platform

available at the edge (such as low-latency offloading of compute, or real-time access to video streams from edge-located cameras), and are written to adapt to scalability-relevant guidance. For example, an application at Tier-3 may be written to offload object recognition in a video frame to Tier-2, but it may also be prepared for a return code indicating that a less accurate (and hence less compute-intensive) algorithm than normal was used because Tier-2 is heavily loaded. Alternatively, Tier-2 or Tier-3 may determine that the wireless channel is congested; based on this guidance, Tier-3 may reduce offered load by preprocessing a video frame and using the result to decide whether it is worthwhile to offload further processing of that frame to the cloudlet. In earlier work [13], we have shown that even modest computation at Tier-3 can make surprisingly good predictions about whether a specific use of Tier-2 is likely to be worthwhile.

Edge-native applications may also use *cross-layer adaptation strategies,* by which knowledge from Tier-3 or Tier-2 is used in the management of the wireless channel between them. For example, an assistive augmented reality (AR) application that verbally guides a visually-impaired person may be competing for the wireless channel and cloudlet resources with a group of AR gamers. In an overload situation, one may wish to favor the assistive application over the gamers. This knowledge can be used by the cloudlet operating system to preferentially schedule the more important workload. It can also be used for prioritizing network traffic by using *fine-grain network slicing,* as envisioned in 5G [5].

Since the techniques for reducing offered workload are application-specific, we focus on a specific class of edge-native applications to validate our ideas. Our choice is a class of applications called *Wearable Cognitive Assistance* (WCA) applications [10]. They are perceived to be "killer apps" for edge computing because (a) they transmit large volumes of video data to the cloudlet; (b) they have stringent end-to-end latency requirements; and (c) they make substantial compute demands of the cloudlet, often requiring high-end GPUs. We leverage unique characteristics of WCA applications to reduce offered load through graceful degradation and improved resource allocation.

Our contributions are as follows:

- An architectural framework for WCA that enables graceful degradation under heavy load.
- An adaptation taxonomy of WCA applications, and techniques for workload reduction.
- A cloudlet resource allocation scheme based on degradation heuristics and external policies.
- A prototype implementation of the above.
- Experimental results showing up to 40% reduction in offered load and graceful degradation in oversubscribed edge systems.

2 Background

2.1 Wearable Cognitive Assistance

Amplifying human cognition in real time through low-latency wireless access from wearable devices to infrastructure resources was first presented as science fiction in 2004 [29]. The building blocks for this vision came into place by 2014, enabling the first implementation of this concept in *Gabriel* [10]. In 2017, Chen et al [4] described a number of applications of this genre, quantified their latency requirements, and profiled the end-to-end latencies of their implementations. In late 2017, SEMATECH and DARPA jointly funded $27.5 million of research on such applications [26, 34]. At the Mobile World Congress in February 2018, wearable cognitive assistance was the focus of an entire session [28]. For AI-based military use cases, this class of applications is the centerpiece of "Battlefield 2.0" [7]. By mid-2019, WCA was being viewed as a prime source of "killer apps" for edge computing [30, 31].

2.2 Gabriel Platform

Our work is built on the Gabriel platform [4, 10], shown in Figure 2. The Gabriel front-end on a wearable device performs preprocessing of sensor data (e.g., compression and encoding), which it streams over a wireless network to a cloudlet. The Gabriel back-end on the cloudlet has a modular structure. The *control module* is the focal point for all interactions with the wearable device. A publish-subscribe (PubSub) mechanism decodes and distributes the incoming sensor streams to multiple *cognitive modules* (e.g., task-specific computer vision algorithms) for concurrent processing. Cognitive module outputs are integrated by a task-specific *user guidance module* that performs higher-level cognitive processing such as inferring task state, detecting errors, and generating guidance in one or more modalities (e.g., audio, video, text, etc.).

The original Gabriel platform was built with a single user in mind, and did not have mechanisms to share cloudlet resources in a controlled manner. It did, however, have a token-based transmission mechanism. This limited a client to only a small number of outstanding operations, thereby offering a simple form of rate adaptation to processing or network bottlenecks. We have retained this token mechanism in our system, described in the rest of this paper. In addition, we have extended Gabriel with new mechanisms to handle multitenancy, perform resource allocation, and support application-aware adaptation. We refer to the two versions of the platform as "Original Gabriel" and "Scalable Gabriel."

2.3 Example Gabriel Applications

Many applications have been built on top of the Gabriel platform. Recent papers [4] [3] describe these applications, along with detailed analysis of their end-to-end latency. For example, the LEGO application guides a user to construct a Lego model, step by step, continuously monitoring the task with computer vision, and providing instructions when it has detected that the user has completed a step. POOL assists a user in aiming a pool cue stick. PING PONG suggests hitting a ball to the left or right to win a rally in table tennis. FACE recognizes a face that has appeared in a scene, searches the user's personal database, and whispers the person's name. IKEA helps a user to assemble an IKEA lamp step by step.

These applications run on multiple wearable devices such as Google Glass, Microsoft HoloLens, Vuzix Glass, and ODG R7. At a high level, the cloudlet workflows of these applications are similar, and consist of two major phases. The first phase uses computer vision to extract a symbolic, idealized representation of the state of the task, accounting for real-world variations in lighting, viewpoint, etc. The second phase operates on the symbolic representation, implements the logic of the task at hand, and occasionally generates guidance for the user. In most WCA applications, the first phase is far more compute intensive than the second phase.

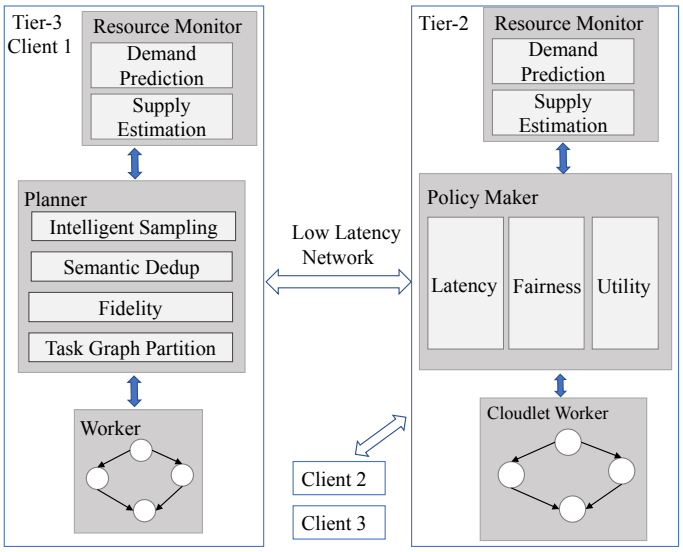


Figure 3: System Architecture

3 Architecture and Adaptation Strategy

The original Gabriel platform has been validated in meeting the latency bounds of WCA applications in single-user settings [4]. Scalable Gabriel aims to meet these latency bounds in multi-user settings, and to ensure performant multitenancy even in the face of overload. We take two complementary approaches to scalability. The first is for applications to reduce their offered load to the wireless network and the cloudlet through adaptation. The second uses end-to-end scheduling of cloudlet resources to minimize queueing and impacts of overload. We embrace both approaches, and combine them using the system architecture shown in Figure 3. We assume benevolent and collaborative clients in this paper, and leave the handling of malicious users to future work.

3.1 System Architecture

We consider scenarios in which multiple Tier-3 devices concurrently offload their vision processing to a single cloudlet over a shared wireless network. The devices and cloudlet work together to adapt workloads to ensure good performance across all of the applications vying for the limited Tier-2 resources and wireless bandwidth. This is reflected in the system architecture shown in Figure 3.

Monitoring of resources is done at both Tier-3 and Tier-2. Certain resources, such as battery level, are device-specific and can only be monitored at Tier-3. Other shared resources can only be monitored at Tier-2: these include processing cores, memory, and GPU. Wireless bandwidth and latency are measured independently at Tier-3 and Tier-2, and aggregated to achieve better estimates of network conditions.

This information is combined with additional high-level predictive knowledge and factored into scheduling and adaptation decisions. The predictive knowledge could arise at the cloudlet (e.g., arrival of a new device, or imminent change in resource allocations), or at the Tier-3 device (e.g., application-specific, short-term prediction of resource demand). All of this information is fed to a *policy module* running on the cloudlet. This module (described in detail in Section 5) is guided by an external policy specification and determines how cloudlet resources should be allocated across competing Tier-3 applications. Such policies can factor in latency needs and fairness, or simple priorities (e.g., a blind person navigation assistant may get priority over an AR game).

A *planner module* on the Tier-3 device uses current resource utilization and predicted short-term processing demand to determine which workload reduction techniques (described in Section 3.3) should be applied to achieve best performance for the particular application given the resource allocations.

3.2 Adaptation Goals

For the applications of interest in this paper, the dominant class of offloaded computations are computer vision operations, e.g., object detection with deep neural networks (DNNs), or activity recognition on video segments. The interactive nature of these applications precludes the use of deep pipelining that is commonly used to improve the efficiency of streaming analytics. Here, end-to-end latency of an individual operation is more important than throughput. Further, it is not just the mean or median of latency, but also the tail of the distribution that matters. There is also significant evidence that user experience is negatively affected by unpredictable variability in response times. Hence, a small mean with short tail is the desired ideal. Finally, different applications have varying degrees of benefit or utility at different levels of latency. Thus, our adaptation strategy incorporates application-specific utility as a function of latency, as well as policies maximizing the total utility of the system.

3.3 Leveraging Application Characteristics

WCA applications exhibit certain properties that distinguish them from other video analytics applications studied in the past. Adaptation based on these attributes provides a unique opportunity to improve scalability.

Human-Centric Timing: The frequency and speed with which guidance must be provided in a WCA application often depends on the speed at which the human performs a task step. Generally, additional guidance is not needed until the instructed action has been completed. For example, in the RibLoc assistant (a medical training application), drilling a hole in bone can take several minutes to complete. During the drilling, no further guidance is provided after the initial instruction to drill. Inherently, these applications contain *active phases,* during which an application needs to sample and process video frames as fast as possible to provide timely guidance, and *passive phases,* during which the human user is busy performing the instructed step. During a passive phase, the application can be limited to sampling video frames at a low rate to determine when the user has completed or nearly completed the step, and may need guidance soon. Although the durations of human operations cannot be predicted and must be considered random variables, many have empirical lower bounds. Adapting sampling and processing rates to match these active and passive phases can greatly reduce offered load. Further, the offered load across users is likely to be uncorrelated because they are working on different tasks or different steps of the same task. If inadvertent synchronization occurs, it can be broken by introducing small randomized delays in the task guidance to different users. These observations suggest that proper end-to-end scheduling can enable effective use of cloudlet resources even with multiple concurrent applications.

Event-Centric Redundancy: In many WCA applications, guidance is given when a user event causes visible state change. For example, placing a lamp base on a table triggers the IKEA Lamp application to deliver the next assembly instruction. Typically, the application needs to process video at a high frame rate to ensure that such state change is detected promptly, leading to further guidance. However, all subsequent frames will continue to reflect this change, and are essentially redundant, wasting wireless and computing resources. Early detection of redundant frames through careful semantic deduplication and frame selection at Tier-3 can reduce the use of wireless bandwidth and cloudlet cycles on frames that show no task-relevant change.

Inherent Multi-Fidelity: Many vision processing algorithms can tradeoff fidelity and computation. For example, frame resolution can be lowered, or a less sophisticated DNN used for inference, in order to reduce processing at the cost of lower accuracy. In many applications, a lower frame rate can be used, saving computation and bandwidth at the expense of response latency. Thus, when a cloudlet is burdened with multiple concurrent applications, there is scope to select operating parameters to keep computational load manageable. Exactly how to do so may be application-dependent. In some cases, user experience benefits from a trade-off that preserves fast response times even with occasional glitches in functionality. For others, e.g., safety-critical applications, it may not be possible to sacrifice latency or accuracy. This in turn translates to lowered scalability of the latter class of application, and hence, a need for more powerful cloudlets and possibly different wireless technology in order to service multiple users.

	Question	Example	Load-reduction Technique
1	How often are instructions given, compared to task duration?	Instructions for each step in IKEA lamp assembly are rare compared to the total task time, e.g., 6 instructions over a 10 minute task.	Enable adaptive sampling based on active and passive phases.
2	Is intermittent processing of input frames sufficient for giving instructions?	Recognizing a face in any one frame is sufficient for whispering the person's name.	Select and process key frames.
3	Will a user wait for system responses before proceeding?	A first-time user of a medical device will pause until an instruction is received.	Select and process key frames.
4	Does the user have a pre-defined workspace in the scene?	Lego pieces are assembled on the lego board. Information outside the board can be safely ignored.	Focus processing attention on the region of interest.
5	Does the vision processing involve identifying and locating objects?	Identifying a toy lettuce for a toy sandwich.	Use tracking as cheap approximation for detection.
6	Are the vision processing algorithms insensitive to image resolution?	Many image classification DNNs limit resolutions to the size of their input layers.	Downscale sampled frames on device before transmission.
7	Can the vision processing algorithm trade off accuracy and computation?	In image classification, MobileNet is computationally cheaper than ResNet, but less accurate.	Change computation fidelity based on resource utilization.
8	Can IMUs be used to identify the start and end of user activities?	User's head movements are of significantly higher magnitude when searching for a Lego block.	Enable IMU-based frame suppression.
9	Is the Tier-3 device powerful enough to run parts of the processing pipeline?	A Jetson TX2 can run MobileNet-based image recognition in real-time.	Partition the vision pipeline between Tier-3 and Tier-2.

Table 1: Application characteristics and corresponding applicable techniques to reduce load

3.4 Adaptation-Relevant Taxonomy

The characteristics described in the previous section largely hold for a broad range of WCA applications. However, the degree to which particular aspects are appropriate to use for effective adaptation is very application dependent, and requires a more detailed characterization of each application. To this end, our system requests a manifest from the developers describing an application. This manifest is a set of yes/no or short numerical responses to the questions in Table 1. Using these as a basis, we construct a taxonomy of WCA applications (shown in Figure 4), based on clusters of applications along dimensions induced from the checklist of questions. In this case, we consider two dimensions – the fraction of time spent in "active" phase, and the significance of the provided guidance (from merely advisory, to critical instructions). Our system varies the adaptation techniques employed to the different clusters of applications. We note that as more applications and more adaptation techniques are created, the list of questions can be extended, and the taxonomy can be expanded.

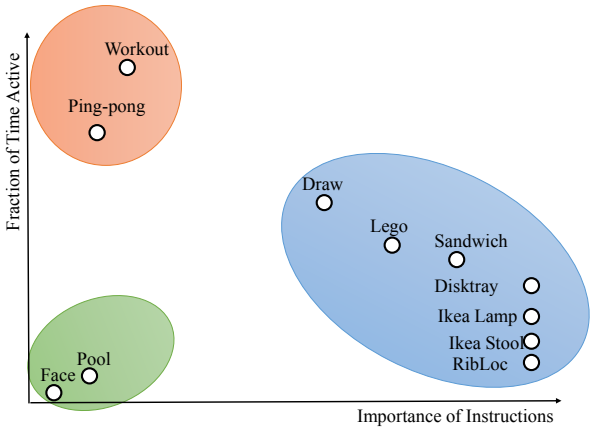


Figure 4: Design Space of WCA Applications

4 Techniques for Workload Reduction

In this section, we focus on techniques to reduce cloudlet workload. We first discuss the intuition and mechanisms of the techniques and then present micro-benchmarks applying these techniques to example applications.

4.1 Adaptive Sampling

The processing demands and latency bounds of a WCA application can vary considerably during task execution because of human speed limitations. When the user is awaiting guidance, it is desirable to sample input at the highest rate to rapidly determine task state and thus minimize guidance latency. However, while the user is performing a task step, the application can stay in a passive state and sample at a lower rate. For a short period of time immediately after guidance is

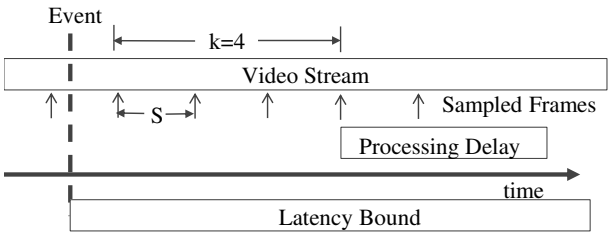


Figure 5: Dynamic Sampling Rate for LEGO

given, the sampling rate can be very low because it is not humanly possible to be done with the step. As more time elapses, the sampling rate has to increase because the user may be nearing completion of the step. Although this active-passive phase distinction is most characteristic of WCA applications that provide step-by-step task guidance (the blue cluster in the lower right of Figure 4), most WCA applications exhibit this behavior to some degree. As shown in the rest of this section, adaptive sampling rates can reduce processing load without impacting application latency or accuracy.

We use task-specific heuristics to define application active and passive phases. In an active application phase, a user is waiting for instructions or is close to needing instructions; therefore the application needs to be "active" by sampling and processing at high frequencies. On the other hand, applications can run at low frequency during passive phases when an instruction-triggering event is unlikely to occur.

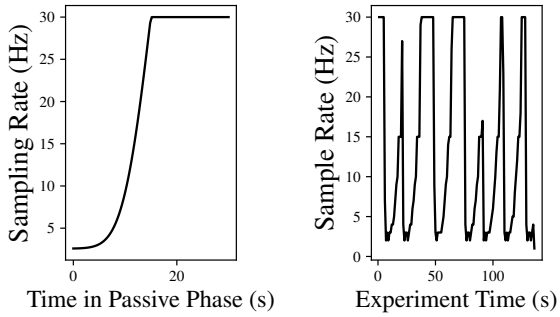
We use the LEGO application to show the effectiveness of adaptive sampling. By default, the LEGO application runs in the active phase. The application enters passive phases immediately following the delivery of an instruction, since the user is going to take a few seconds searching and assembling Lego blocks. The length and sampling rate of a passive phase is provided by the application to the framework. We provide the following system model as an example of what can be implemented. We collect five LEGO traces with 13739 frames as our evaluation dataset.

Length of a Passive Phase: We model the time it takes to finish each step as a Gaussian distribution. We use maximum likelihood estimation to calculate the parameters of the Guassian model.

Lowest Sampling Rate in Passive Phase: The lowest sampling rate in passive phase still needs to meet the application's latency requirement. Figure 5 shows the system model used to calculate the largest sampling period S that still meets the latency bound. In particular,

$$(k-1)S + processing\_delay \leq latency\_bound$$

k represents the cumulative number of frames an event needs to be detected in order to be certain an event actually occurred. The LEGO application empirically sets this value to 5.



(a) Passive Sampling Rate (b) Trace Sampling Rate

Figure 6: Adaptive Sampling Rate

Adaptation Algorithm: At the start of a passive phase, we set the sampling rate to the minimum calculated above. As time progresses, we gradually increase the sampling rate. The idea behind this is that the initial low sampling rates do not provide good latency, but this is acceptable, as the likelihood of an event is low. As the likelihood increases (based on the Gaussian distribution described earlier), we increase sampling rate to decrease latency when events are likely. Figure 6(a) shows the sampling rate adaptation our system employs during a passive phase. The sampling rate is calculated as

$$sr = min\_sr + \alpha * (max\_sr - min\_sr) * cdf\_Gaussian(t)$$

sr is the sampling rate. t is the time after an instruction has been given. α is a recovery factor which determines how quickly the sampling rate rebounds to active phase rate.

Figure 6(b) shows the sampling rate for a trace as the application runs. The video captures a user doing 7 steps of a LEGO assembly task. Each drop in sampling rate happens after an instruction has been delivered to the user. Table 2 shows the percentage of frames sampled and guidance latency comparing adaptive sampling with naïve sampling at half frequency. Our adaptive sampling scheme requires processing fewer frames while achieving a lower guidance latency.

4.2 IMU-based Passive Phase Suppression

In many applications, passive phases can often be associated with the user's head movement. We illustrate with two applications here. In LEGO, during the passive phase, which begins after the user receives the next instruction, a user typically turns away from the Lego board and starts searching for the next brick to use in a parts box. During this period, the computer vision algorithm would detect no meaningful task states if the frames are transmitted. In PING PONG, an active phase lasts throughout a rally. Passive phases are in between actual game play, when the user takes a drink, switches sides, or, most commonly, tracks down and picks up a wayward ball

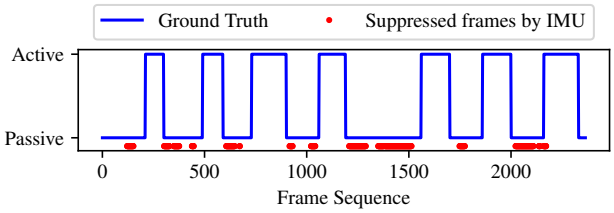
Trace	Sample Half Freq	Adaptive Sampling
1	50%	25%
2	50%	28%
3	50%	30%
4	50%	30%
5	50%	43%

(a) Percentage of Frames Sampled

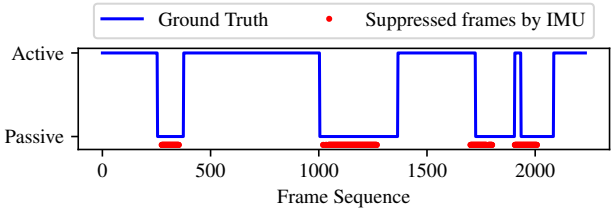
	Guidance Delay (frames±stddev)
Sample Half Freq	7.6 ± 6.9
Adaptive Sampling	5.9 ± 8.2

(b) Guidance Latency

Table 2: Frames Sampled and Guidance Latency



(a) LEGO



(b) PING PONG

Figure 7: Accuracy of IMU-based Frame Suppression

from the floor. They are associated with a much larger range of head movements than during a rally when the player generally looks toward the opposing player. Again, the frames can be suppressed on the client to reduce wireless transmission and load on the cloudlet. In both scenarios, significant body movement can be detected through Inertial Measurement Unit (IMU) readings on the wearable device, and used to predict such passive phases.

For each frame, we get a six-dimensional reading from the IMU: rotation in three axes, and acceleration in three axes. We train an application-specific SVM to predict active/passive phases based on IMU readings, and suppress predicted passive frames on the client. Figure 7(a) and (b) show an example trace from LEGO and PING PONG, respectively. Human-labeled ground truth indicating passive and active phases is

	Suppressed Passive Frames (%)	Max Delay of State Change Detection
Trace 1	17.9%	0
Trace 2	49.9%	0
Trace 3	27.1%	0
Trace 4	37.0%	0
Trace 5	34.1%	0

(a) LEGO

	Suppressed Passive Frames (%)	Loss of Active Frames (%)
Trace 1	21.5%	0.8%
Trace 2	30.0%	1.5%
Trace 3	26.2%	1.9%
Trace 4	29.8%	1.0%
Trace 5	38.4%	0.2%

(b) PING PONG

Table 3: Effectiveness of IMU-based Frame Suppression

shown in blue. The red dots indicate predictions of passive phase frames based on the IMU readings; these frames are suppressed at the client and not transmitted. Note that in both traces, the suppressed frames also form streaks. In other words, a number of frames in a row can be suppressed. As a result, the savings we gain from IMU is orthogonal to that from adaptive sampling.

Although the IMU approach does not capture all of the passive frames (e.g., in LEGO, the user may hold his head steady while looking for the next part), when a passive frame is predicted, this is likely correct (i.e., high precision, moderate recall). Thus, we expect little impact on event detection accuracy or latency, as few if any active phase frames are affected. This is confirmed in Table 3, which summarizes results for five traces from each application. We are able to suppress up to 49.9% of passive frames for LEGO and up to 38.4% of passive frames in case of PING PONG on the client, while having minimal impact on application quality — incurring no delay in state change detection in LEGO, and less than 2% loss of active frames in PING PONG.

5 Cloudlet Resource Allocation

A complementary method to improve scalability is through judicious allocation of cloudlet resources among concurrent application services. Resource allocation has been well explored in many contexts of computer systems, including operating systems, networks, real-time systems, and cloud data centers. While these prior efforts can provide design blueprints for cloudlet resource allocation, the characteristics of edge-native applications emphasize unique design challenges.

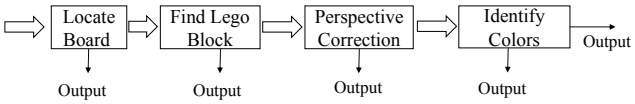


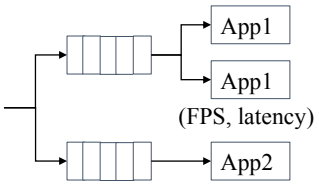
Figure 8: LEGO Processing DAG

The ultra-low application latency requirements of edge-native applications are at odds with large queues often used to maintain high resource utilization of scarce resources. Even buffering a small number of requests may result in end-to-end latencies that are several multiples of processing delays, hence exceeding acceptable latency thresholds. On the other hand, when using short queues, accurate estimations of throughput, processing, and networking delay are vital to enable efficient use of cloudlet resources. However, sophisticated computer vision processing represents a highly variable computational workload, even on a single stream. For example, as shown in Figure 8, the processing pipeline for LEGO has many exits, resulting in highly variable execution times.

To adequately provision resources for an application, one approach is to leave the burden to developers, asking them to specify and reserve a static number of cores and amount of memory needed for the service. However, this method is known to be highly inaccurate and typically leads to over-reservation in data-centers. For cloudlets, which are more resource constrained, such over-reservation will lead to even worse under-utilization or inequitable sharing of the available resources. Instead, we seek to create an automated resource allocation system that leverages knowledge of the application requirements and minimizes developer effort. To this end, we ask developers to provide target Quality of Service (QoS) metrics or a utility function that relates a single, easily-quantified metric (such as latency) to the quality of user experience. Building on this information, we construct offline application profiles that map multidimensional resource allocations to application QoS metrics. At runtime, we calculate a resource allocation plan to maximize a system-wide metric (e.g., total utility, fairness) specified by the cloudlet owner. We choose to consider the allocation problem per application rather than per client in order to leverage statistical multiplexing among clients and multi-user optimizations (e.g., cache sharing) in an application.

5.1 System Model

Figure 9 shows the system model we consider. Each application is given a separate input queue. Each queue can feed one or more application instances. Each application instance is encapsulated in a container with controlled resources. In this model, with adequate computational resources, clients of different applications have minimal sharing and mainly contend for the wireless network.



Only request flow is shown.

Figure 9: Resource Allocation System Model

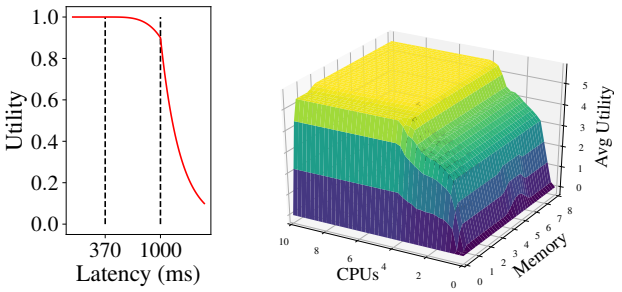
We use a utility-based approach to measure and compare system-wide performance under different allocation schemes. For WCA, the utility of a cloudlet response depends on both the quality of response and its QoS characteristics (e.g., end-to-end latency). The total utility of a system is the sum of all individual utilities. A common limitation of a utility-based approach is the difficulty of creating utility functions. One way to ease such burden is to position an application in the taxonomy described in Section 3.4 and borrow from similar applications. Another way is to calculate or measure application latency bounds, such as through literature review or physics-based calculation as done in [4].

The system-wide performance is a function of the following independent variables: (a) the number of applications and the number of clients of each application; (b) the number of instances of each application; and, (c) the resource allocation for each instance. Although (a) is not under our control, Gabriel is free to adapt (b) and (c). Furthermore, to optimize system performance, it may sacrifice the performance of certain applications in favor of others. Alternatively, it may choose not to run certain applications at all.

5.2 Application Utility and Profiles

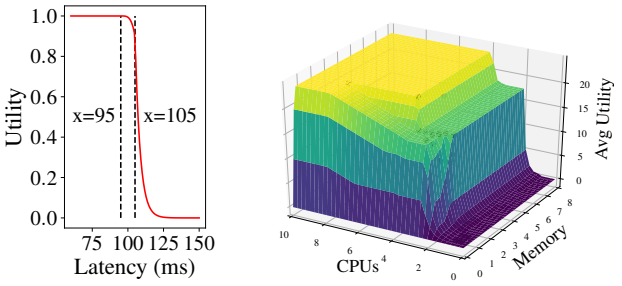
We build application profiles offline in order to estimate latency and throughput at runtime. First, we ask developers to provide a utility function that maps QoS metrics to application experience. Figure 10(a) and Figure 11(a) show utility functions for two applications based on latency bounds identified by [4] for each request. Next, we profile an application instance by running it under a discrete set of cpu and memory limitations, with a large number of input requests. We record the processing latency and throughput, and calculate the system-wide utility per unit time. We interpolate between acquired data points of <system utility, resources> to produce continuous functions. Hence, we effectively generate a multi-dimensional resource to utility profile for each application.

We make a few simplifying assumptions to ensure profile generation and allocation of resources by utility are tractable. First, we assume utility values across different applications are comparable. Furthermore, we assume utility is received on a per-frame basis, with values that are normalized between



(a) Utility For FACE (b) Profile for FACE

Figure 10: FACE Application Utility and Profile



(a) Utility For POOL (b) Profile for POOL

Figure 11: POOL Application Utility and Profile

0 and 1. Each frame that is sent, accurately processed, and replied within its latency bound receives 1, so a client running at 30 FPS under ideal conditions can receive a maximum utility of 30 per second. This clearly ignores variable utility of processing particular frames (e.g., differences between active and passive phases), but simplifies construction of profiles and modeling for resource allocation; we leave the complexities of modeling content-based variable utility to future work. Figure 10(b) and Figure 11(b) show the generated application profiles for FACE and POOL. We see that POOL is more efficient than FACE in using each unit of resource to produce utility. If an application needs to deliver higher utility than a single instance can, our framework will automatically launch more instances of it on the cloudlet.

5.3 Resource Allocation

Given the workload of concurrent applications running on a cloudlet, and the number of clients requesting service from each application, our resource allocator determines how many instances to launch and how much resource (CPU cores, memory, etc.) to allocate for each application based on an external policy specified by cloudlet operators. We describe an example policy below, which maximizes the system-wide total utility. Other types of policies, such as max-min fairness, can also be enforced by our allocation mechanism. In addition, we assume queueing delays are limited by the token mechanism described in the Gabriel framework [10], which limits the number of outstanding requests on a per-client basis.

As described earlier, for each application $a \in \{\text{FACE}, \text{LEGO}, \text{PING PONG}, \text{POOL}, \dots\}$, we construct a resource to utility mapping $u_a : \mathbf{r} \to \mathbb{R}$ for one instance of the application on cloudlet, where $\mathbf{r}$ is a resource vector of allocated CPU, memory, etc. We formulate the following optimization problem which maximizes the system-wide total utility, subject to a tunable maximum per-application limit:

$$\begin{aligned} \max_{\{k_a, \mathbf{r}_a\}} \quad & \sum_a k_a \cdot u_a(\mathbf{r}_a) \\ \text{s.t.} \quad & \sum_a k_a \cdot \mathbf{r}_a \preccurlyeq \hat{\mathbf{r}} \\ & 0 \preccurlyeq \mathbf{r}_a \quad \forall a \\ & k_a \cdot u_a(\mathbf{r}_a) \le \gamma \cdot c_a \quad \forall a \\ & k_a \in \mathbb{Z} \end{aligned} \tag{1}$$

In above, c_a is the number of mobile clients requesting service from application a. The total resource vector of the cloudlet is $\hat{\mathbf{r}}$. For each application a, we determine how many instances to launch — k_a, and allocate resource vector $\mathbf{r}_a$ to each of them. A tunable knob γ regulates the maximum utility allotted per application, and serves to enforce a form of partial fairness (no application can be given excessive utility, though some may still receive none). The larger γ is, the more aggressive our scheduling algorithm will be in maximizing global utility and suppressing low-utility applications. By default, we set $\gamma = 10$, which, based on our definition of utility, roughly means resources will be allocated so no more than one third of frames (from a 30FPS source) will be processed within satisfactory latency bounds for a given client.

Solving the above optimization problem is computationally difficult. We thus use an iterative greedy allocation algorithm as follows: for each application profile $u_a(\mathbf{r})$, we find the resource point that gives the highest $\frac{u_a(\mathbf{r})}{|\mathbf{r}|}$, i.e., *utility-to-resource* ratio. Denote this point as $\mathbf{r}_a^*$. We start with the application with the largest $\frac{u_a(\mathbf{r}_a^*)}{|\mathbf{r}_a^*|}$. We allocate k_a application instances, each with resource $\mathbf{r}_a^*$, such that k_a is the largest integer with $k_a \cdot u_a(\mathbf{r}_a^*) \le \gamma \cdot c_a$. If there are leftover resources, we move to the application with the next highest utility-to-resource ratio and repeat the process.

To implement resource reservation and enforcement, we leverage the `cpu-shares` and `memory-reservation` control options of Linux Docker containers. These limit a container's resource utilization only when there is contention, but allow it to use as much left-over resources as needed.

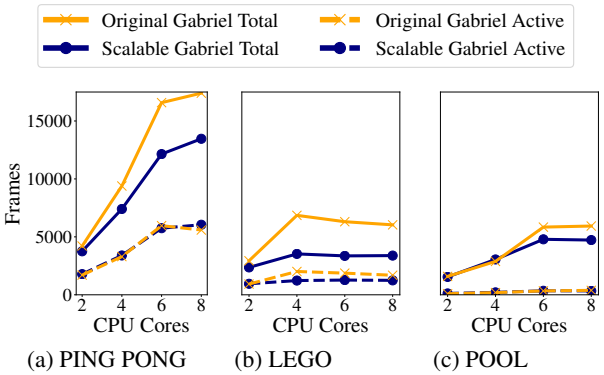


Figure 12: Effects of Workload Reduction

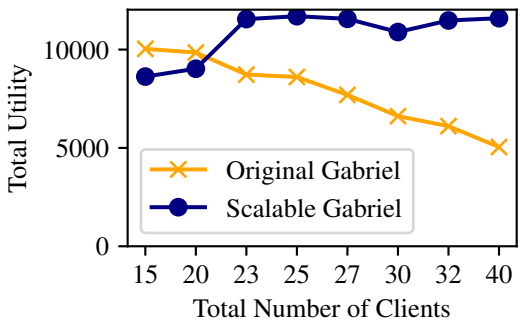


Figure 13: Total Utility with Increasing Contention

Exp #	Number of Clients					
	Total	FACE	LEGO	POOL	PING PONG	IKEA
1	15	3	3	3	3	3
2	20	4	4	4	4	4
3	23	5	5	4	4	5
4	25	5	5	5	5	5
5	27	5	6	6	5	5
6	30	5	7	6	6	6
7	32	5	7	7	7	6
8	40	8	8	8	8	8

Table 4: Resource Allocation Experiments

6 Evaluation

We use five WCA applications, including FACE, PING PONG, LEGO, POOL, and IKEA for evaluation [4] [3]. These applications are selected based on their distinct requirements and characteristics to represent the variety of WCA apps. IKEA and LEGO assist users step by step to assemble an IKEA lamp or a Lego model. While their 2.7-second loose latency bound is less stringent than other apps, the significance of their instructions is high, as a user could not proceed without the instruction. On the other hand, users could still continue their tasks without the instructions from FACE, POOL, and PING PONG assistants. For POOL and PING PONG, the speed of an instruction is paramount to its usefulness. For example, any instruction that comes 105ms after a user action for POOL is no longer of value.

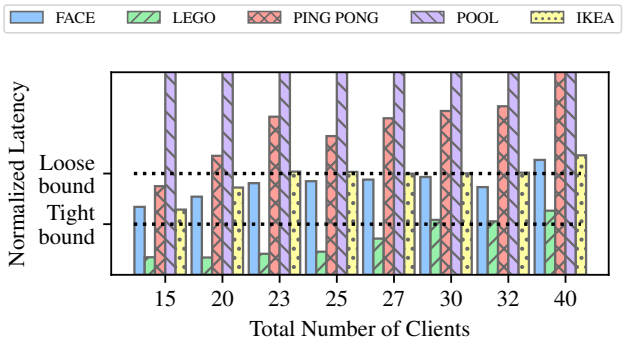
6.1 Effectiveness of Workload Reduction

We first evaluate the effectiveness of all of the workload reduction techniques explored in Section 4. For this set of experiments, we use a single application and a static resource allocation. We use four Nexus 6 mobile phones as clients, connecting to a cloudlet over a Wi-Fi link. We run PING PONG, LEGO, and POOL applications one at a time with 2, 4, 6, and 8 cores available on the server. Figure 12 shows the total number of frames processed with and without workload reduction. Note that although the offered work is greatly reduced, the processed frames for active phases of the application have not been affected. Thus, we confirm that we can significantly reduce cloudlet load without affecting the critical processing needed by these applications.

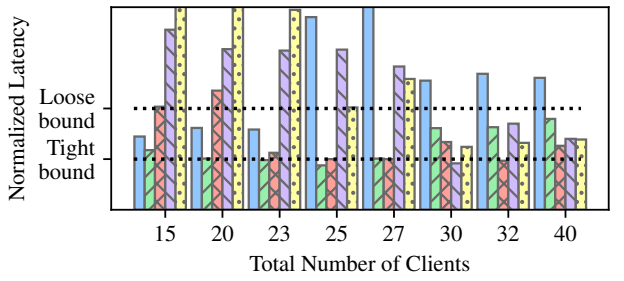
6.2 Effectiveness of Resource Allocation

We next evaluate resource allocation on a server machine with 2 Intel® Xeon® E5-2699 v3 processors, totaling 36 physical cores running at 2.3 Ghz (turbo boost disabled) and 128 GB memory. We dedicate 8 physical cores (16 hyperthreads) and 16 GB memory as cloudlet resources using cgroup. We run 8 experiments with increasing numbers of clients across five concurrent applications with a total of 15 to 40 clients. The breakdown of the number of clients used for each experiment is given in Table 4. We use offline-generated application profiles discussed in Section 5 to optimize total system utility. Figure 13 shows how the system-wide total utility changes as we add more clients to the workload, under the original Gabriel approach and the scalable Gabriel approach. We see that original Gabriel's total utility drops more than 40% as contention increases, since every client contends for resources in an uncontrolled fashion. All applications suffer, but the effects of increasing latencies are vastly different among different applications. In contrast, scalable Gabriel maintains a high level of system-wide utility by differentially allocating resources to different applications based on their sensitivity captured in the utility profiles.

Figure 14 and Figure 15 provide insights into how scalable Gabriel strikes the balance. Latencies are better controlled as resources are dedicated to applications with high utility, and more clients are kept within their latency bounds. Of course,



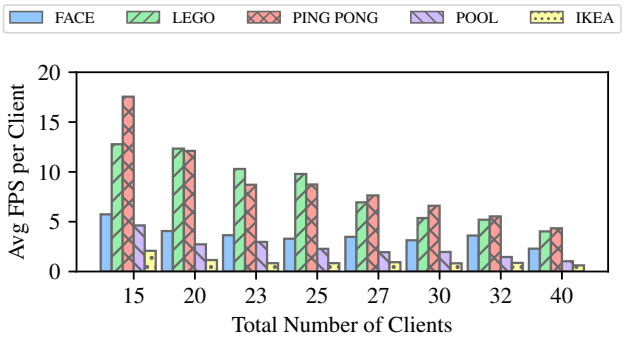
(a) Original Gabriel



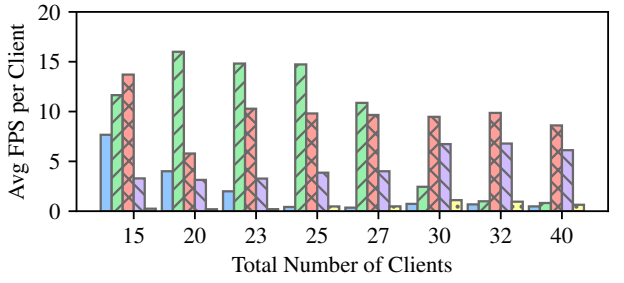
(b) Scalable Gabriel

Note: These are 90th percentiles of response latency, normalized by per-application tight and loose bounds [4]

Figure 14: Effects of Resource Allocation on Latency

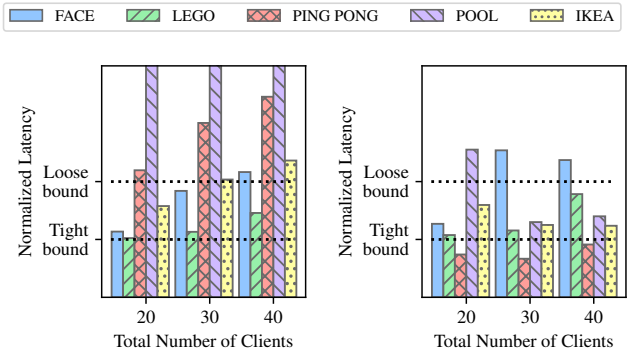


(a) Original Gabriel



(b) Scalable Gabriel

Figure 15: Effects of Resource Allocation on Throughput



(a) Original Gabriel (b) Scalable Gabriel

Note: These are 90th percentiles of response latency, normalized by per-application tight and loose bounds [4]

Figure 16: Combined Effects on Response Latency

with higher contention, fewer frames per second can be processed for each client. Original Gabriel degrades applications in an undifferentiated fashion. Scalable Gabriel, in contrast, tries to maintain higher throughput for some applications at the expense of the others, e.g. LEGO up to 25 clients.

6.3 Effects on Guidance Latency

We next evaluate the combined effects of workload reduction and resource allocation in our system. We emulate many users running multiple applications simultaneously. All users share the same cloudlet with 8 physical cores and 16 GB memory. We conduct three experiments, with 20 (4 clients per app), 30 (6 clients per app), and 40 (8 clients per app) clients. Each client loops through pre-recorded video traces with random starting points. Figure 16 and Fig 17 show per client frame latency and FPS achieved. The first thing to notice is that concurrently utilizing both sets of techniques does not cause conflicts. In fact, they appear to be complementary and latencies remain in better control than using resource allocation alone.

The previous plots consider per request latencies. The ultimate goal of our work is to maintain user experience as much as possible and degrade it gracefully when overloaded. For WCA applications, the key measure of user experience is guidance latency, the time between the occurrence of an event and the delivery of corresponding guidance. Figure 18 shows boxplots of per-application guidance latencies for the concurrent application experiments above. The red line denotes the application-required loose bound. It is clear that our methods control latency significantly better than the baseline. Scalable Gabriel is able to serve at least 3x the number of

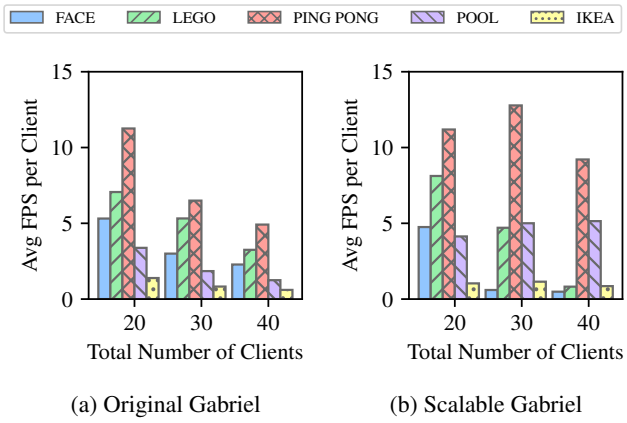


Figure 17: Combined Effects on Throughput

clients when moderately loaded while continuing to serve half of the clients when severely loaded. In these experiments, the utility is maximized at the expense of the FACE and IKEA application, which provides the least utility per resource consumed. At the highest number of clients, scalable Gabriel sacrifices the LEGO application to maintain the quality of service for the other two. This differentiated allocation is reflected in Figure 19. In contrast, with original Gabriel, none of the applications are able to regularly meet deadlines.

7 Related Work

Although edge computing is new, the techniques for scalability examined in this paper bear some resemblance to work that was done in the early days of mobile computing, and more recent cloud management work.

Odyssey [25] and extensions [8] proposed upcall-based collaboration between a mobile's operating system and its applications to adapt to variable wireless connectivity and limited battery. Exploration of tradeoffs between application fidelity and resource demand led to the concept of *multi-fidelity applications* [32]; such concepts are relevant to our work, but the critical computing resources in our setting are those of the cloudlet rather than the mobile device.

Several different approaches to adapting application fidelity have been studied. Dynamic sampling rate with various heuristics for adaptation have been tried primarily in the context of individual mobile devices for energy efficiency [19, 21, 22, 35]. Semantic deduplication to reduce redundant processing of frames have been suggested by [12, 13, 17, 38]. Similarly, previous works have looked at suppression based on motion either from video content [20, 23] or IMUs [15]. Others have investigated exploiting multiple deep models with accuracy and resource tradeoffs [11, 16]. While most of these efforts

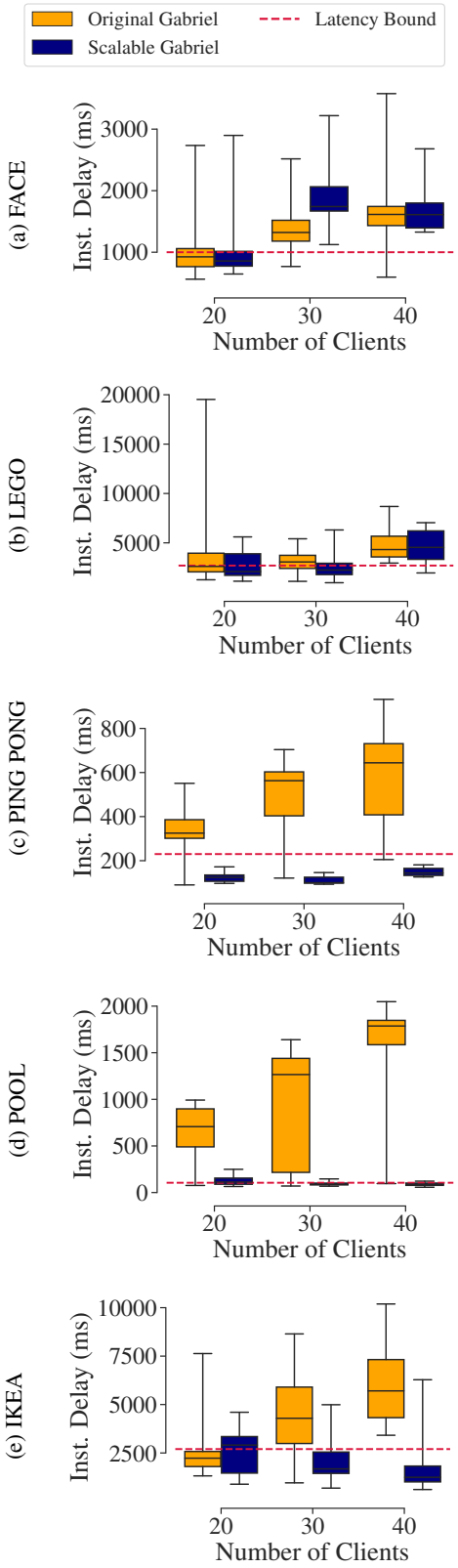


Figure 18: Guidance Latency

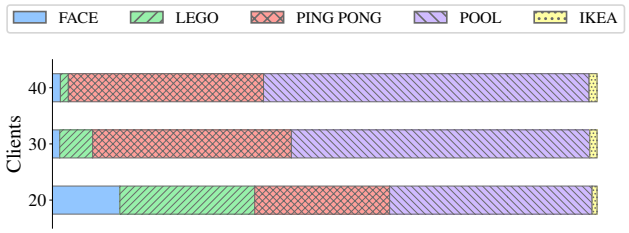


Figure 19: Fraction of Cloudlet Processing Allocated

were in mobile-only, cloud-only, or mobile-cloud context, we explore similar techniques in an edge-native context.

Partitioning workloads between mobile devices and the cloud have been studied in sensor networks [24], throughput-oriented systems [6, 36], for interactive applications [2, 27], and from programming model perspectives [1]. We believe that these approaches will become important techniques to scale applications on heavily loaded cloudlets.

Dynamic resource allocation schemes on the cloud for video processing have been well explored [9, 18, 33]. More recently, profile-based adaptation of video analytics [14, 16, 37] focused on throughput-oriented analytics applications on large clusters or in the cloud. In contrast, our goals focus on interactive performance on relatively small edge deployments.

8 Conclusion and Future Work

More than a decade ago, the emergence of cloud computing led to the realization that applications had to be written in a certain way to take full advantage of elasticity of the cloud. This led to the concept of "cloud-native applications" whose scale-out capabilities are well matched to the cloud, as well as tools and techniques to easily create such applications.

The emergence of edge computing leads to another inflection point in application design. In particular, it leads to "edge-native applications" that are deeply dependent on attributes such as low latency or bandwidth scalability that can only be obtained at the edge. However, as this paper has shown, edge-native applications have to be written in a way that is very different from cloud-native applications if they are to be scalable.

This is the first work to show that cloud-native implementation strategies that focus primarily on dynamic scale-out are unlikely to be effective for scalability in edge computing. Instead, edge-native applications need to adapt their network and cloudlet resource demand to system load. As the total number of Tier-3 devices associated with a cloudlet increases, the per-device network and cloudlet load has to decrease. This is a fundamental difference between cloud-native and edge-native approaches to scalability.

In this paper, we explore client workload reduction and server resource allocation to manage application quality of service in the face of contention for cloudlet resources. We demonstrate that our system is able to ensure that in overloaded situations, a subset of users are still served with good quality of service rather than equally sharing resources and missing latency requirements for all.

This work serves as an initial step towards practical resource management for edge-native applications. There are many potential directions to explore further in this space. We have alluded to some of these earlier in the paper. One example we briefly mentioned is dynamic partitioning of work between Tier-3 and Tier-2 to further reduce offered load on cloudlets. In addition, other resource allocation policies, especially fairness-centered policies, such as max-min fairness and static priority can be explored when optimizing overall system performance. These fairness-focused policies could also be used to address aggressive users, which are not considered in this paper. While we have shown offline profiling is effective for predicting demand and utility for WCA applications, for a broader range of edge-native applications, with ever more aggressive and variable offload management, online estimation may prove to be necessary. Another area worth exploring is the particular set of control and coordination mechanisms to allow cloudlets to manage client-offered load directly. Finally, the implementation to date only controls allocation of resources but allows the cloudlet operating system to arbitrarily schedule application processes. Whether fine-grained control of application scheduling on cloudlets can help scale services remains an open question.

ACKNOWLEDGEMENTS

We thank our shepherd, Eyal de Lara, and the anonymous reviewers for their guidance. This research was supported in part by the National Science Foundation (NSF) under grant number CNS-1518865, by the National Science Foundation Graduate Research Fellowship Program under Grant Nos. DGE1252522 and DGE1745016, and by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0051. Additional support was provided by Intel, Vodafone, Deutsche Telekom, Verizon, Crown Castle, Seagate, VMware, MobiledgeX, and the Conklin Kistler family fund. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the view(s) of their employers or the above-mentioned funding sources.

REFERENCES

[1] Rajesh Krishna Balan, Mahadev Satyanarayanan, So Young Park, and Tadashi Okoshi. 2003. Tactics-based remote execution for mobile computing. In *Proceedings of the 1st international conference on Mobile systems, applications and services*. ACM, 273–286.

[2] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. 2015. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. ACM, 155–168.

[3] Zhuo Chen. 2018. *An Application Platform for Wearable Cognitive Assistance*. Ph.D. Dissertation. Carnegie Mellon University.

[4] Zhuo Chen, Wenlu Hu, Junjue Wang, Siyan Zhao, Brandon Amos, Guanhang Wu, Kiryong Ha, Khalid Elgazzar, Padmanabhan Pillai, Roberta Klatzky, Dan Siewiorek, and Mahadev Satyanarayanan. 2017.

An Empirical Study of Latency in an Emerging Class of Edge Computing Applications for Wearable Cognitive Assistance. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. Fremont, CA.

[5] Luis M. Contreras and Diego R. Lopez. 2018. A Network Service Provider Perspective on Network Slicing. *IEEE Softwarization* (January 2018). https://sdn.ieee.org/newsletter/january-2018/a-network-service-provider-perspective-on-network-slicing.

[6] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. 2010. MAUI: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 49–62.

[7] Zak Doffman. 2018. Battlefield 2.0: How Edge Artificial Intelligence is Pitting Man Against Machine. *Forbes* (November 2018).

[8] Jason Flinn and Mahadev Satyanarayanan. 1999. Energy-aware Adaptation for Mobile Applications. In *Proceedings of the Seventeenth ACM Symposium on Operating systems Principles*. Charleston, SC.

[9] Tom ZJ Fu, Jianbing Ding, Richard TB Ma, Marianne Winslett, Yin Yang, and Zhenjie Zhang. 2015. DRS: dynamic resource scheduling for real-time analytics over fast streams. In *2015 IEEE 35th International Conference on Distributed Computing Systems*. IEEE, 411–420.

[10] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. 2014. Towards Wearable Cognitive Assistance. In *Proceedings of ACM MobiSys*.

[11] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. 2016. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *ACM MobiSys'16*. ACM, 123–136.

[12] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B Gibbons, and Onur Mutlu. 2018. Focus: Querying large video datasets with low latency and low cost. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 269–286.

[13] Wenlu Hu, Brandon Amos, Zhuo Chen, Kiryong Ha, Wolfgang Richter, Padmanabhan Pillai, Benjamin Gilbert, Jan Harkes, and Mahadev Satyanarayanan. 2015. The Case for Offload Shaping. In *Proceedings of HotMobile 2015*. Santa Fe, NM.

[14] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. 2018. Videoedge: Processing camera streams using hierarchical clusters. In *Symposium on Edge Computing (SEC)*. IEEE, 115–131.

[15] Puneet Jain, Justin Manweiler, and Romit Roy Choudhury. 2015. Overlay: Practical mobile augmented reality. In *MobiSys'15*. ACM, 331–344.

[16] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 253–266.

[17] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. Noscope: optimizing neural network queries over video at scale. *VLDB'17* 10, 11 (2017), 1586–1597.

[18] Ahmed S Kaseb, Anup Mohan, and Yung-Hsiang Lu. 2015. Cloud resource management for image and video analysis of big data from network cameras. In *2015 International Conference on Cloud Computing and Big Data (CCBD)*. IEEE, 287–294.

[19] Nicholas D Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T Campbell. 2010. A survey of mobile phone sensing. *IEEE Communications* 48, 9 (2010), 140–150.

[20] Kiron Lebeck, Eduardo Cuervo, and Matthai Philipose. [n. d.]. Collaborative Acceleration for Mixed Reality. ([n. d.]).

[21] Konrad Lorincz, Bor-rong Chen, Geoffrey Werner Challen, Atanu Roy Chowdhury, Shyamal Patel, Paolo Bonato, Matt Welsh, et al. 2009. Mercury: a wearable sensor network platform for high-fidelity motion analysis.. In *SenSys*, Vol. 9. 183–196.

[22] Konrad Lorincz, Bor-rong Chen, Jason Waterman, Geoff Werner-Allen, and Matt Welsh. 2008. Resource aware programming in the pixie os. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*. ACM, 211–224.

[23] Saman Naderiparizi, Pengyu Zhang, Matthai Philipose, Bodhi Priyantha, Jie Liu, and Deepak Ganesan. 2017. Glimpse: A programmable early-discard camera architecture for continuous mobile vision. In *MobiSys'17*. ACM, 292–305.

[24] Ryan Newton, Sivan Toledo, Lewis Girod, Hari Balakrishnan, and Samuel Madden. 2009. Wishbone: Profile-based Partitioning for Sensornet Applications.. In *NSDI*, Vol. 9. 395–408.

[25] Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, J. Eric Tilton, Jason Flinn, and Kevin R. Walker. 1997. Agile Application-Aware Adaptation for Mobility. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*. Saint-Malo, France.

[26] John Oakley. 2018. Intelligent Cognitive Assistants (ICA) Workshop Summary and Research Needs. https://www.nsf.gov/crssprgm/nano/reports/ICA2_Workshop_Report_2018.pdf.

[27] Moo-Ryong Ra, Anmol Sheth, Lily Mummert, Padmanabhan Pillai, David Wetherall, and Ramesh Govindan. 2011. Odessa: enabling interactive perception applications on mobile devices. In *MobiSys'11*. ACM, 43–56.

[28] Tiernan Ray. 2018. An Angel on Your Shoulder: Who Will Build A.I.? *Barron's* (February 2018).

[29] Mahadev Satyanarayanan. 2004. Augmenting Cognition. *IEEE Pervasive Computing* 3, 2 (April-June 2004).

[30] Mahadev Satyanarayanan and Nigel Davies. 2019. Augmenting Cognition through Edge Computing. *IEEE Computer* 52, 7 (July 2019).

[31] Mahadev Satyanarayanan, Guenter Klas, Marco Silva, and Simone Mangiante. 2019. The Seminal Role of Edge-Native Applications. In *Proceedings of the 2019 IEEE International Conference on Edge Computing (EDGE)*. Milan, Italy.

[32] Mahadev Satyanarayanan and Dushyanth Narayanan. 1999. Multi-Fidelity Algorithms for Interactive Mobile Applications. In *Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DialM)*. Seattle, WA.

[33] Krisantus Sembiring and Andreas Beyer. 2013. Dynamic resource allocation for cloud-based media processing. In *Proceeding of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM, 49–54.

[34] Sherry Stokes. 2018. New Center Headquartered at Carnegie Mellon Will Build Smarter Networks To Connect Edge Devices to the Cloud. https://www.cmu.edu/news/stories/archives/2018/january/conix-research-center.html.

[35] Narseo Vallina-Rodriguez and Jon Crowcroft. 2012. Energy management techniques in modern mobile handsets. *IEEE Communications Surveys & Tutorials* 15, 1 (2012), 179–198.

[36] Shanhe Yi, Zijiang Hao, Qingyang Zhang, Quan Zhang, Weisong Shi, and Qun Li. 2017. Lavea: Latency-aware video analytics on edge computing platform. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 15.

[37] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. 2017. Live video analytics at scale with approximation and delay-tolerance. In *NSDI'17*.

[38] Tan Zhang, Aakanksha Chowdhery, Paramvir Victor Bahl, Kyle Jamieson, and Suman Banerjee. 2015. The design and implementation of a wireless video surveillance system. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM, 426–438.