Are Cloudlets Necessary?

Ying Gao, Wenlu Hu, Kiryong Ha, Brandon Amos,
Padmanabhan Pillai[†], Mahadev Satyanarayanan

October 2015
CMU-CS-15-139

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[†]Intel Labs

Abstract

We present experimental results from Wi-Fi and 4G LTE networks to validate the intuition that low end-to-end latency of cloud services improves application response time and reduces energy consumption on mobile devices. We focus specifically on computational offloading as a cloud service. Using a wide range of applications, and exploring both pre-partitioned and dynamically partitioned approaches, we demonstrate the importance of low latency for cloud offload services. We show the best performance is achieved by offloading to cloudlets, which are small-scale edge-located data centers. Our results show that cloudlets can improve response times 51% and reduce energy consumption in a mobile device by up to 42% compared to cloud offload.

This research was supported by the National Science Foundation (NSF) under grant numbers IIS-1065336 and CNS-1518865. Additional support was provided by the Intel Corporation, Google, Vodafone, and the Conklin Kistler family fund. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and should not be attributed to their employers or funding sources.

Keywords: low latency, augmented reality, mobile computing, cloud computing, fog computing, edge computing, 4G LTE, cellular networks, Wi-Fi, user interaction, user experience, COMET, face recognition, RTT

1 Introduction

It is widely believed that reducing the end-to-end latency between a mobile device and its cloud services can improve user experience and extend battery life. One approach to reducing latency is to leverage small-scale edge-located data centers variously known as *cloudlets* [12], *micro data centers* [6], *fog* [1], or *mobile edge cloud* [3]. For ease of exposition we use the term "cloudlet" for this concept in the rest of this paper. Although this idea of "moving the cloud closer" is gaining momentum and industry investment, nagging questions remain about how much improvement is likely. In other words, how much benefit do cloudlets really provide, relative to just using the cloud?

In this paper, we focus on the use of cloudlets and the cloud for offloading computations from a mobile device over Wi-Fi and 4G LTE networks. On a wide range of applications, some pre-partitioned by a developer and others dynamically partitioned by a runtime system, we explore response time and energy consumption characteristics. Through these experimental results, we establish the importance of offloading as well as the importance of low latency for offload services. Finally, we report on the sensitivity of offloading performance to the interactivity level of an application.

2 Experimental Approach

The results of cloud offloading experiments are notoriously difficult to interpret because they depend on both the quality of the partitioning of the application as well as the performance characteristics of the software and hardware infrastructure. It is relatively easy to vary infrastructure characteristics such as network bandwidth in a controlled experimental setting. However, ensuring that an application partitioning is optimal for current infrastructure conditions is much more difficult. In this paper, we use an existing cloud offload tool called `COMET` [5], which represents the state-of-the-art in offloading off-the-shelf Android applications. We have made no modification to the tool, so its performance in creating optimal partitions for current conditions represents an unbiased external factor in our results.

In addition to applications that are dynamically partitioned by `COMET`, we also study applications that have been statically pre-partitioned by a developer. In these applications, the mobile client component performs sensing and user interaction, but offloads compute-intensive functionality to the server component.

Any result-based insights that are stable across this wide range of dynamically partitioned and manually pre-partitioned applications are likely to be robust and significant. We address the following questions:

- How does the location of the offloading site affect the performance of pre-partitioned mobile applications?
- Is `COMET` able to use cloudlets to consistently deliver better results relative to using the cloud?
- How does the choice of wireless network (Wi-Fi versus 4G LTE) affect offloading results?
- How does the interactivity level of an application affect performance relative to the choice of an offloading site?

Application	Request size (avg)	Response size (avg)
FACE	62 KB	< 60 bytes
MAR	26 KB	< 20 bytes
FLUID	16 bytes	25 KB

Figure 1: Characteristics of Pre-partitioned Apps

- For both dynamically and statically partitioned applications, how does the energy consumption of a mobile device vary with the location of the offloading site?

2.1 Applications

We study three statically pre-partitioned applications:

- FACE is a face recognition application that first detects faces in an image, and then attempts to identify the face from a pre-populated database. This implements the Eigenfaces method [15] using OpenCV [2], and runs on a Microsoft Windows environment. Training the classifiers and populating the database are done offline, so our experiments only consider the execution time of the recognition task on a pre-trained system.
- MAR is an augmented reality application that labels buildings and landmarks in a scene [14]. The prototype application uses a dataset of 1005 labeled images of 200 buildings. MAR runs on Microsoft Windows, and makes significant use of OpenCV library [2], Intel Performance Primitives (IPP) library, and multiple processing threads.
- FLUID is an example of physics-based computer graphics [13]. It models an imaginary fluid with which the user can interact. The FLUID server is Linux-based and is multithreaded. It runs a physics simulation for each new set of accelerometer readings from the client. For good interactive experience, the total end-to-end delay has to be below 100 ms. In our experiments, FLUID simulates a 2218-particle system with 20 ms timesteps, generating up to 50 frames per second.

Each of these applications is split into a front-end mobile app and a back-end server. For example, in FACE, an image is shipped from the mobile device to the back-end; there, a face recognition algorithm is executed and a text string of the name of the person is returned. Figure 1 shows network transfer size for both request and response for each application. For FACE and MAR, request data size is much larger than the response size because the mobile device sends an image and receives a text string as the result. In contrast, response size is larger for FLUID because it continuously receives location and pressure information for many particles.

In addition to these, we also investigate offloading standalone mobile apps designed to run on Android devices. These are the off-the-shelf applications that anyone can download at Google's Play store. We use the COMET framework to offload portions of these unmodified applications to a remote server: application threads and necessary data are migrated to the server when performing significant computation, and migrated back on any I/O operations or other interactions. We test three such applications:

Application	Total Transfer Size	# of Data Transfers
`Linpack`	$\approx$ 10 MB	1
`CPU Benchmark`	$\approx$ 80 KB	1
`PI Benchmark`	$\approx$ 10 MB	15

Figure 2: Characteristics of `COMET` Apps (# of data transfers indicates the frequency of thread migration for each run)

Smartphone (Samsung Galaxy Nexus)	Netbook (Dell Latitude 2120)
ARM® Cortex-A9	Intel® Atom™ N550
1.2 GHz 2 core	1.5 GHz, 2 cores
1 GB RAM	2 GB RAM
32 GB Flash	250 GB HDD
802.11a/b/g/n Wi-Fi	802.11a/g/n Wi-Fi

Figure 3: Hardware configuration for mobile devices

- `Linpack` for Android [10] performs numerical linear algebra computations to solve a dense N-by-N system of linear equations. This benchmark is also used in the original `COMET` paper [5].
- `CPU benchmark` [16] is a simple benchmarking tool designed for testing and comparing phone processing speeds and effects of overclocking modifications.
- `PI Benchmark` [11] measures CPU and memory speed by calculating π up to 2 million digits of precision.

All of these applications are very compute-intensive, and should benefit greatly from offloading. However, as shown in Figure 2, they differ in the amount and frequency of data transfers on `COMET`.

2.2 Experimental Setup

Figure 3 and 4 show the mobile devices, cloudlet, and cloud we use for the experiments. In the cloud (Amazon EC2), we use the most powerful VM instance available to us in terms of CPU clock speed. The instance has 8 cores (VCPUs), each with the fastest available clock rate (2.8 GHz); its memory size is more than enough for all of the tested applications (15 GB). The cloudlet is a Dell Optiplex 9010 desktop machine with 4 cores (VCPUs), whose clock is limited to 2.7 GHz. The VM instance on it has a memory size of 4 GB. By comparing a relatively weak cloudlet against more powerful EC2 cloud instances, we have deliberately stacked the deck against cloudlets. Hence, any cloudlet wins in our experiments should be considered quite meaningful.

We use two different mobile devices in our experiments. For `COMET` applications, we are limited by the fact that Android 4.1.X is the latest Android version on which `COMET` runs. For these experiments, we use a Samsung Galaxy Nexus phone that runs Android 4.1.1. For pre-partitioned applications, our choice of hardware is limited by the fact that some of our applications only run on x86 hardware. Hence, we use a Dell Latitude 2120 netbook to run all the pre-partitioned applications. Its computational power is comparable to the latest

Cloudlet VM on Dell Optiplex 9010	Cloud (Amazon AWS) c3.2xlarge instance
Intel® Core® i7-3770 2.7 GHz[†], 4 VCPUs 4 GB RAM 8 GB Virtual disk 1 Gbps Ethernet	Intel® Xeon E5-2680 v2 2.8 GHz, 8 VCPUs 15 GB RAM 160 GB SSD Amazon Enhanced Network

[†]We limit the CPU clock to 2.7 GHz and disable Turbo boost.

Figure 4: Hardware configuration for offloading sites

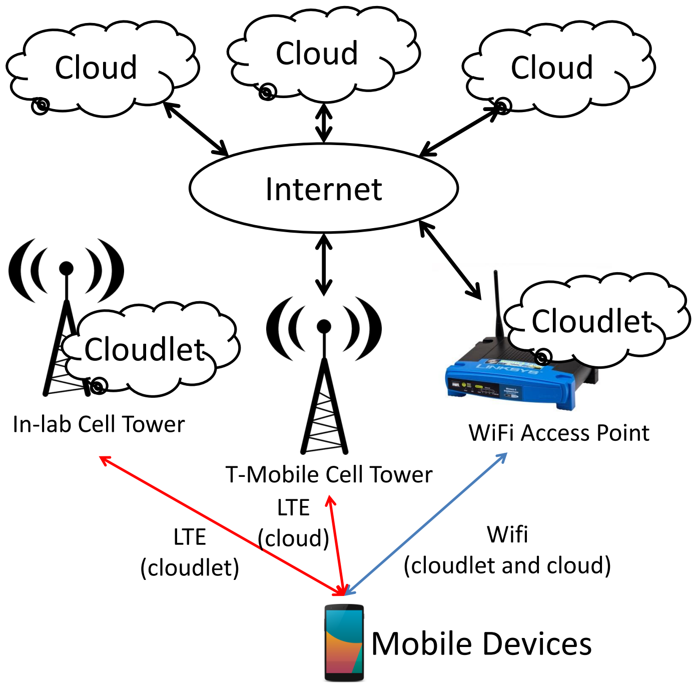


Figure 5: Network Setup for the Experiments

smartphones.

All our experiments are conducted on the CMU campus, located in Pittsburgh, PA. We note that our campus usually has good connectivity to Amazon EC2-East. We can often measure over 200 Mbps bandwidth, and under 8 ms RTT to this site. This is atypical for cloud access from a mobile device. Li et al. [8] report that average round trip time (RTT) from 260 global vantage points to their optimal Amazon EC2 instances is 74 ms, which is similar to EC2-West in our experimental setting. Thus, in interpreting the results presented in this paper, EC2-West should be regarded as typical of user experience with cloud offload.

Figure 5 illustrates the networking used in our experiments. There are four configurations, as described below.

Cloud-WiFi: The mobile device uses 802.11n to connect to a private Wi-Fi access point that is connected to the campus network via Ethernet, and thence via the Internet to an Amazon AWS site.

Cloudlet-WiFi: This is similar to the Cloud-WiFi configuration, except that network

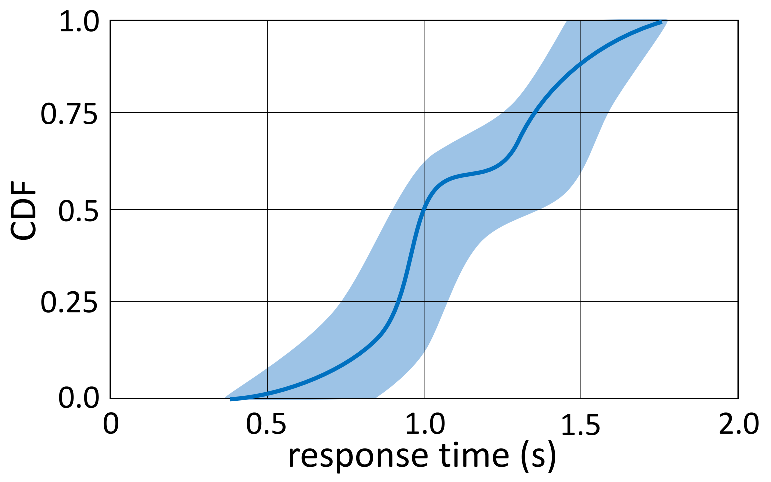


Figure 6: Example CDF plot. Best case is just under 0.5 s, median is 1.0 s, and worst case around 1.7 s. The shaded area indicates the range of the distributions measured on multiple trials. Here, median response ranges from 0.9 to 1.3 s.

traffic only needs to go as far as to the cloudlet, which is on the same Ethernet segment as the access point.

Cloudlet-LTE: Under a license for experimental use from the FCC, we have set up an in-lab 4G LTE network. This in-lab cellular network uses a Nokia eNodeB whose transmission strength is attenuated to 10 mW. Network traffic via the eNodeB is directed through a Nokia RACS gateway and local Ethernet segment to the cloudlet. Tethering on a Google Nexus 6 smartphone is used to connect the mobile devices to this cellular network. We use USB tethering for the netbook and Wi-Fi tethering for the smartphone.

Cloud-LTE: We use cellular data service on the T-Mobile 4G LTE network to reach the Internet, and thence an Amazon AWS site. Tethering on a Google Nexus 6 smartphone is used to connect the mobile devices to the T-Mobile service. We use USB tethering for the netbook and Wi-Fi tethering for the smartphone.

2.3 Visualizing the Results

In our experiments, we perform many measurements at each of our operating conditions. Because we are interested in the end-to-end response times of our applications, many different phenomena, including WiFi congestion, routing instability, WAN bandwidth limitations, and inherent data-dependent variability of application computations can affect our results. Therefore, we can expect a wide-ranging distribution of results. To fully capture these rich distributions, we present our results as a Cumulative Distribution Function (CDF) for each experimental condition as shown in Figure 6.

CDF plots provide much richer and more complete information than just reporting means and standard deviations. They make it easy to see the best-case, worst-case, median, and arbitrary percentile of response times for each experiment. In addition, our CDF plots show the observed range of variability in results over multiple runs of an experiment. As Figure 6 illustrates, the overall CDF is shown along with a shaded region that encapsulates all the observations across multiple runs. For example, the left and right edges of the shaded area at y=0.5 indicate the best and worst median response times observed. Unless indicated otherwise, we use this visualization method for all of the experimental results with three sets of experiments.

3 Experimental Results

3.1 Offloading Over WiFi

Figure 7 compares response times when offloading pre-partitioned applications to different Amazon AWS sites, or to a cloudlet. Also shown is the application response time without any offload, which is labeled as *No Offload*. This corresponds to running the server component locally on the netbook, thereby avoiding all network transmission.

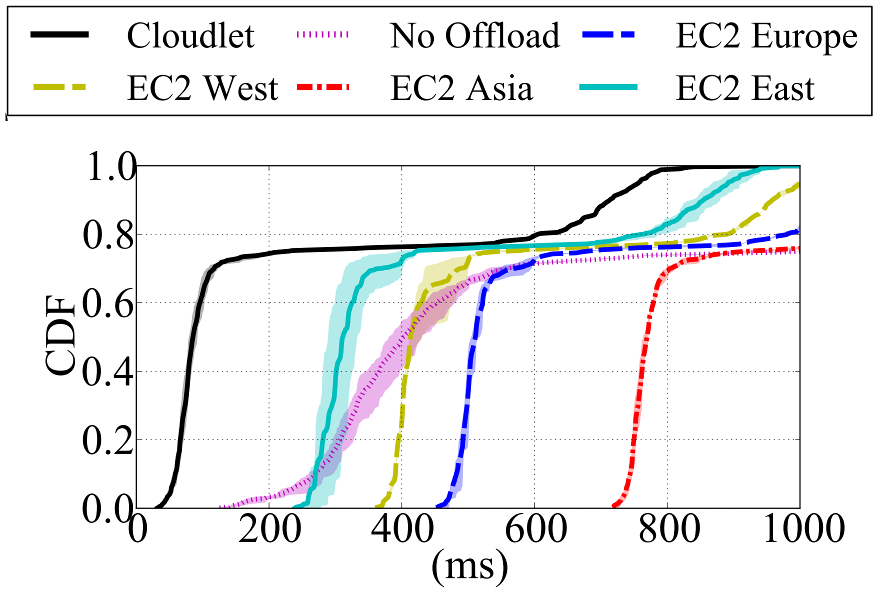
Response time clearly degrades with increasing latency to the offload site. This is clearest for `MAR` and `FLUID`, where the CDF curves shift right due to higher network latency and lower effective bandwidth to the farther offload sites. The CDFs also rise less sharply, indicating the variations in response times increase, as offloading to more remote sites is subject to greater uncertainty of WAN network conditions. These effects are also present in the `FACE` plots, but are less apparent because the application itself has a large, data-dependent variation in performance. As a result of this response time degradation, *No Offload* often provides faster response times than offloading to distant clouds. The cloudlet, on the other hand, consistently provides the fastest response times.

Figure 8 presents the results for `COMET`-based applications. These results also show the impact of offload location on response time. EC2-West, EC2-Europe, and EC2-Asia are clearly worse than EC2-East. However, these longer running applications (seconds to tens of seconds, rather than tens to hundreds of milliseconds) change the performance trade-offs between EC2-East and the cloudlet. As mentioned in Section 2.2, latency to EC2-East is on the order of 8 ms, which is not much larger than WiFi latency. As a result, the cloudlet's win over EC2-East is lost. The greater computational power of the cloud instances over the cloudlet now becomes significant, thus yielding the advantage to EC2-East.

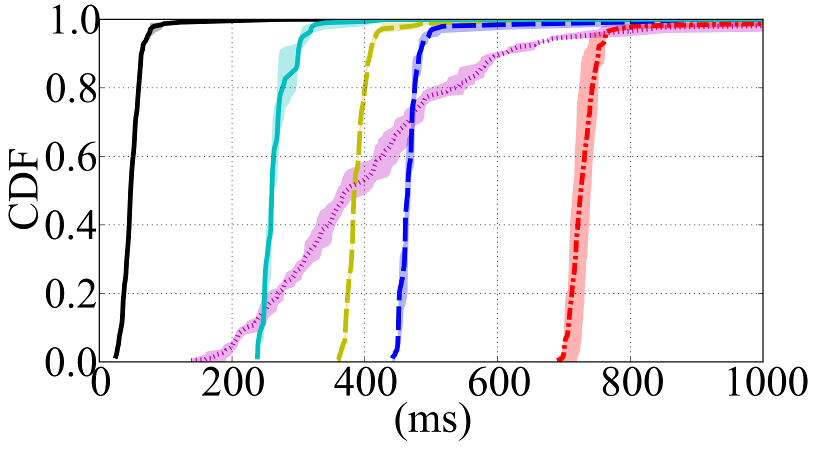
The results also show the impact of application characteristics. `CPU Benchmark` transfers very little data, so the various EC2 curves look very similar, mainly shifted by the difference in RTT to the different sites. `Linpack` transfers significantly more data, and the differences in response times between EC2 sites is much more apparent. The `PI Benchmark` transfers a similar amount of data in total, but in multiple transfers, as the processing thread is transferred back and forth between the mobile device and backend. As a result, for `PI Benchmark`, the network effects of distant clouds is magnified.

3.2 Offloading Over LTE

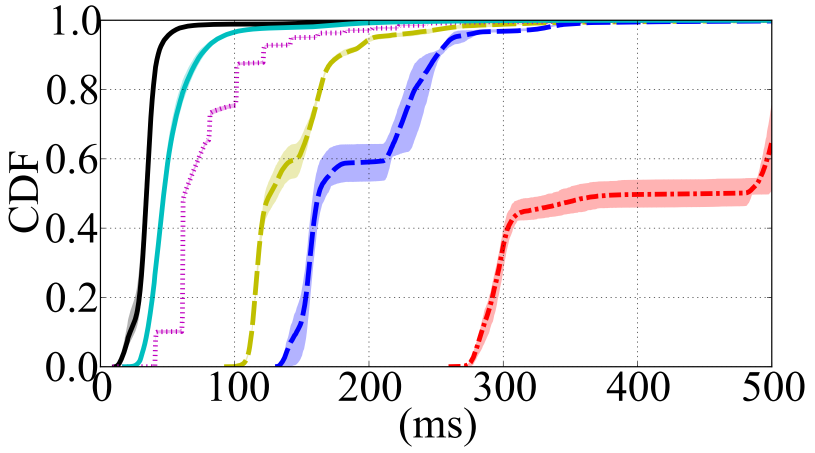
Our initial experiments with LTE showed that network latencies tend to jump between discrete values. Figure 9 shows the CDF of the observed response times over LTE from a simple echo server that gradually increases the delay before sending a reply. We would expect a straight line, indicating uniform distribution of response times, but the results show a distinct "stair-step" pattern (in-lab cell tower in Figure 9). We believe this is because



(a) FACE (300 images)

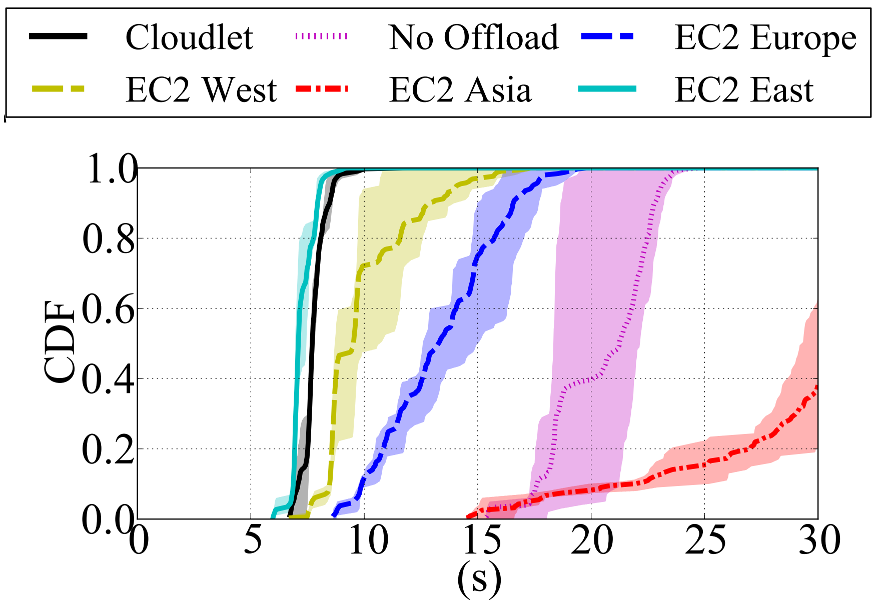


(b) MAR (100 images)

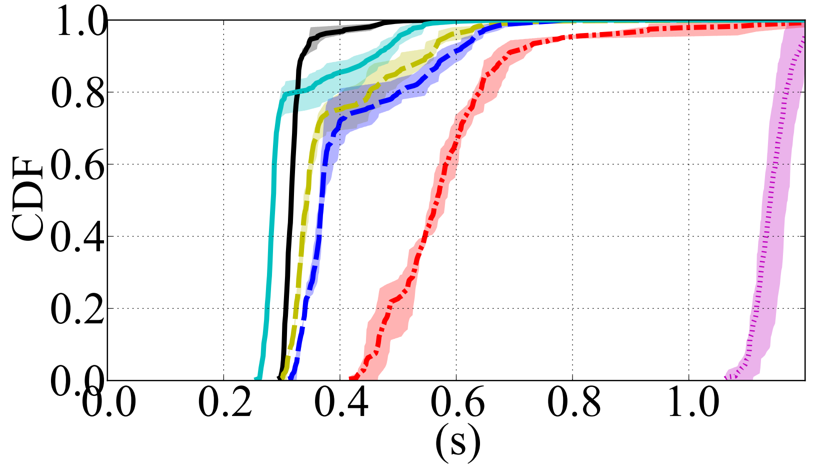


(c) FLUID (10 min run)

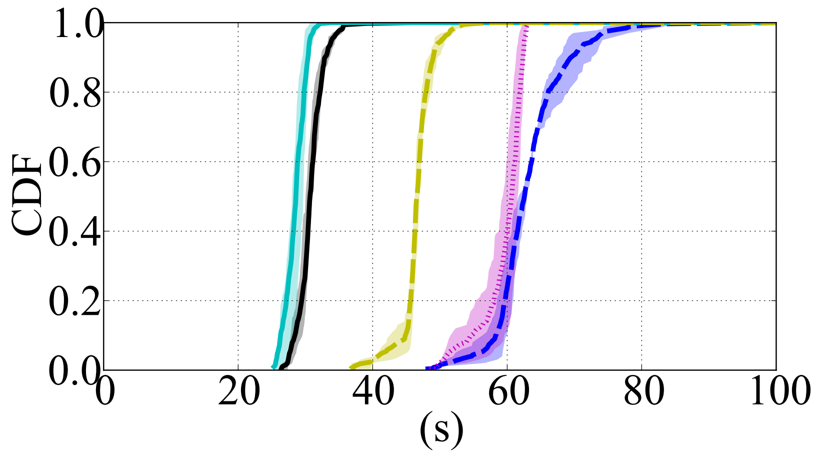
Figure 7: CDF of response times for client-server apps, using WiFi



(a) `Linpack`



(b) `CPU Benchmark`



(c) `PI Benchmark`

Figure 8: CDF of execution times for COMET apps, using WiFi

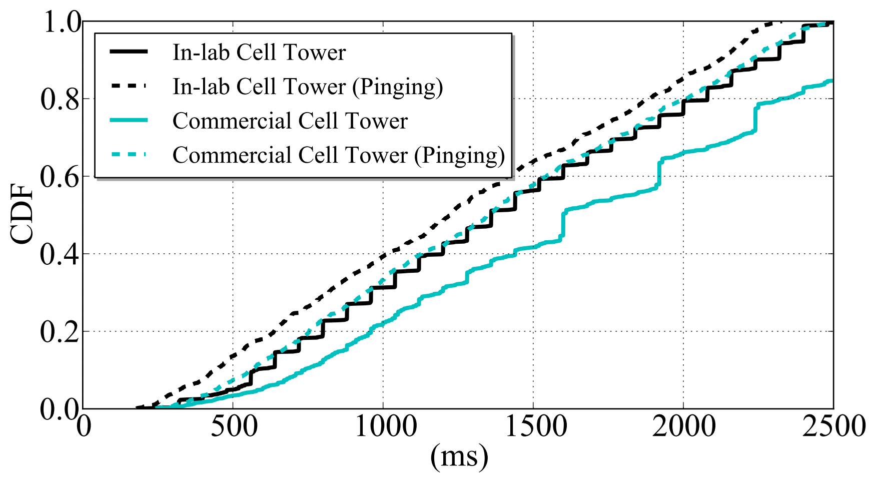


Figure 9: "Stair-steps" for LTE cases

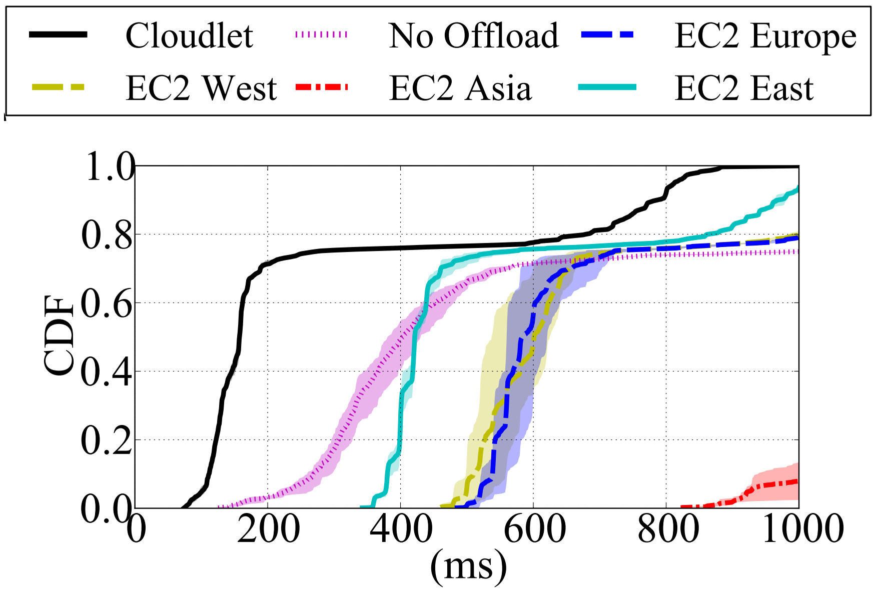
medium access in LTE, unlike in WiFi, is carefully controlled, and latencies can be quantized into discrete buckets based on the next transmission slot granted for the device. We note that the commercial LTE network shows an even more complex pattern, switching from a short to a long interval if more than a second or so has elapsed since the last transmission (commercial cell tower in Figure 9). Our experiments also show that if the mobile device performs frequent data transmissions (periodic background pings), this effect disappears. To avoid the confounding effects of latency discretization, all our LTE experiments were run concurrently with background pings at intervals of 1 ms.

Figure 10 shows the response times for three pre-partitioned applications. Overall, these results are quite similar to those from the WiFi experiments. As the LTE latencies are slightly higher, the curves are shifted to the right compared to the WiFi results. Additionally, the commercial LTE network does not have the unusually good connectivity to EC2-East datacenter that our campus network has. So, although the EC2-East response times are still the best among the cloud scenarios, they are not quite as low as in our WiFi experiments. In all cases, offloading to the cloudlet provides the best response times.

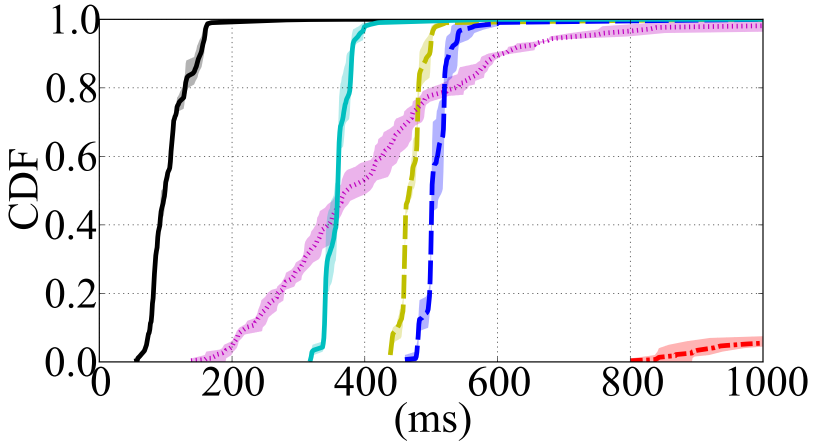
Figure 11 presents our results for `COMET`-partitioned applications. Now, the cloudlet gives the best results, followed by EC2-East, and progressively worse results for the farther offload sites. Because the `PI Benchmark` requires multiple movements of computation between the mobile device and offload site (as shown earlier in Figure 2), it is affected more by the network distance than `Linpack` even though the total data transfered is roughly the same in both cases.

3.3 Effects of Interactivity

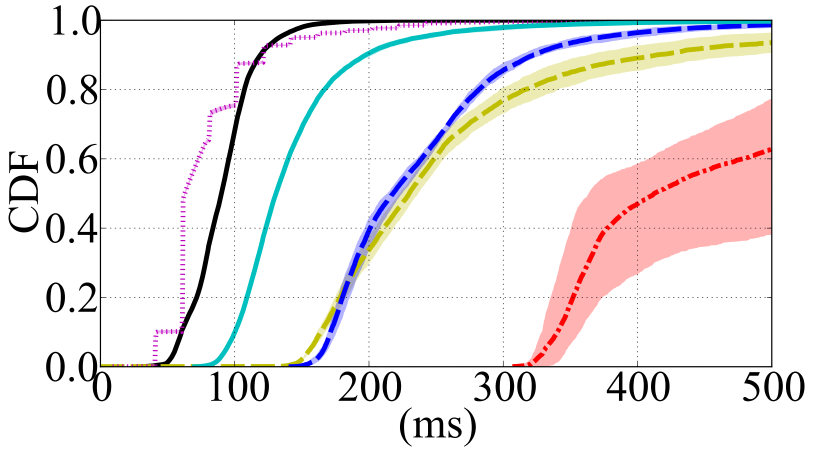
COMET is effective at offloading CPU-intensive portions of unmodified Android applications to the cloud or cloudlet. However, on any interaction with device hardware (e.g., sensor I/O, screen update, or file operation), the application threads need to migrate back to the mobile device. This can cause many thread migrations over the course of the program execution, magnifying the effects of network latency on performance, as we saw in the `PI Benchmark` results.



(a) `FACE` (300 images)

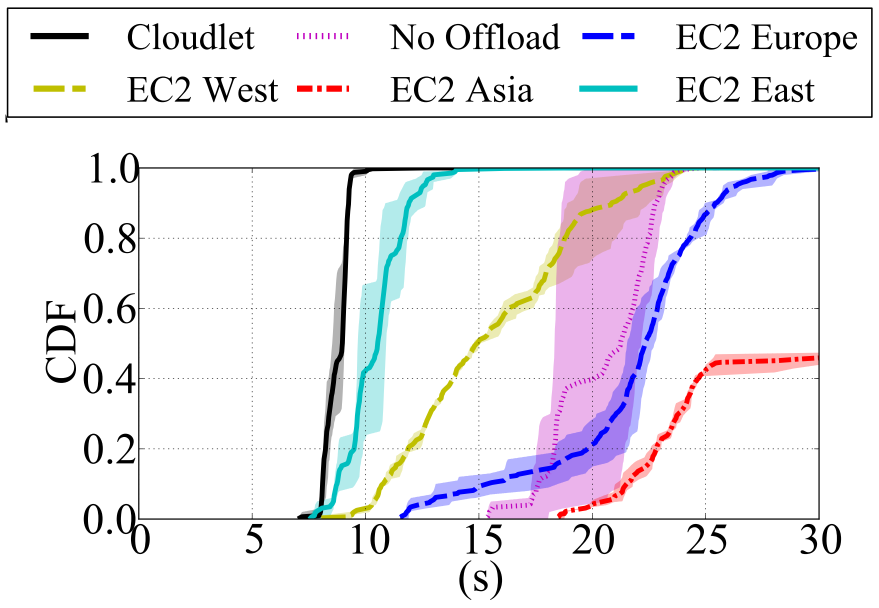


(b) `MAR` (100 images)

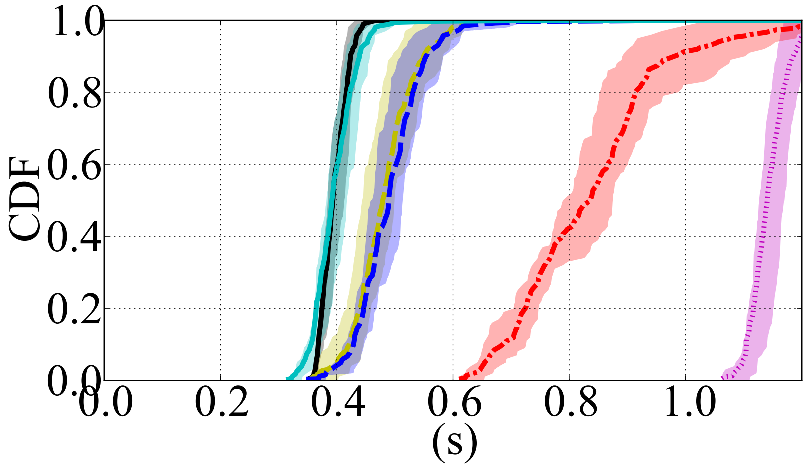


(c) `FLUID` (10 min run)

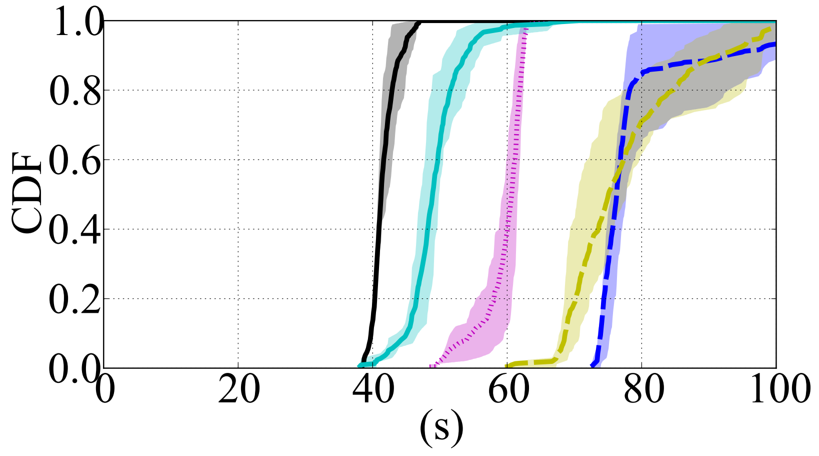
Figure 10: CDF of response times for client-server apps, using LTE



(a) `Linpack`



(b) `CPU Benchmark`



(c) `PI Benchmark`

Figure 11: CDF of execution times for COMET apps, using LTE

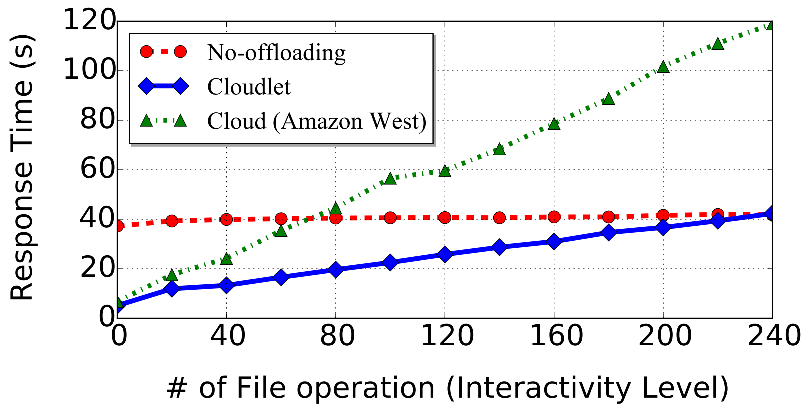


Figure 12: Effect of Interactivity on COMET offload

To study this effect of interactivity further, we create a custom CPU-intensive application that factors large integers. Our app saves intermediate results to a file after a configurable number of iterations, allowing us to vary how often migrations are triggered with COMET. Figure 12 shows how the app performance changes as the number of file operations (interactivity level) increases. The execution time without offloading remains nearly constant because the application is CPU bound on the mobile device, even with a few hundred file operations. However, cloud offloading performance is highly affected, and becomes slower than just running on the mobile device once we exceed 75 I/O operations during the execution. In contrast, offloading to a WiFi cloudlet is much less sensitive to the interactivity changes than the cloud. Until 240 file writes, it performs better than the no-offloading case. Cloudlets can greatly benefit offloading frameworks like COMET, allowing them to apply more broadly by relaxing restrictions on application interactivity. Recent research, *Tango* [4], corroborates that interactivity of an application is critical in code offloading; this effort tried to improve offloading performance for interactive apps by deterministically replaying a replica at the offloading site. In contrast, we suggest using cloudlet offload to avoid high-latency costs for migrations altogether.

3.4 Energy

In addition to improving performance, offloading computation can reduce the energy consumption on mobile devices. To see how choice of offloading site affects energy consumption, we rerun our offload scenarios while measuring the power consumption of the mobile devices. For the netbook, we remove the battery and use a WattsUp power meter [17] to log the total power draw of at the AC power cord. On the smartphone, we interpose a Monsoon Power Monitor [9] meter at the battery contacts, logging voltage and current draw during execution of our benchmarks.

The energy measurements are shown in Figure 13, along with standard deviations in parentheses. We measure average energy consumption per each offloading request. Note that these are for the WiFi experiments. Offload of any kind greatly reduces power on the mobile devices, since they no longer need to perform heavy computations. The choice of offload site does not affect power, which is about the same for cloudlet and cloud cases.

Offload		None	Cloudlet	East	West	EU	Asia
Face[†]	(J/query)	12.4 (0.5)	2.6 (0.3)	4.4 (0.0)	6.1 (0.2)	9.2 (4.1)	9.2 (0.2)
Fluid[†]	(J/frame)	0.8 (0.0)	0.3 (0.0)	0.3 (0.0)	0.9 (0.0)	1.0 (0.0)	2.2 (0.1)
MAR[†]	(J/query)	5.4 (0.1)	0.6 (0.1)	3.0 (0.8)	4.3 (0.1)	5.1 (0.1)	7.9 (0.1)
Linpack	(J/run)	40.3 (2.6)	13.0 (0.7)	13.3 (2.3)	16.9 (1.8)	18.2 (1.9)	38.1 (4.1)
CPU	(J/run)	9.6 (1.4)	5.7 (0.3)	5.9 (0.3)	5.8 (0.3)	5.9 (0.2)	6.0 (0.2)
PI	(J/run)	129.7 (2.9)	53.9 (2.1)	57.6 (1.8)	107.6 (8.6)	162.8 (18.0)	203.4 (16.7)

[†] The display is turned off during energy measurement. Figures in parentheses are standard deviations.

Figure 13: Energy consumption on mobile devices

However, execution duration does vary, so the actual energy consumed per query/frame/run is greatly affected by offload site. In all cases, cloudlet offload results in the least energy consumed per execution, while further offload sites incur increasing energy costs. In fact, in many cases, offloading to EC2-asia or EC2-Europe results in higher energy cost than no-offload case. Overall, offloading to cloudlets reduces energy consumption by 41% to 89% over running on the mobile device, and around 42% lower than offload to EC2-West, which we consider representative of the average cloud site.

4 Conclusion

Our study has shown that the choice of offloading site is important for both pre-partitioned applications and dynamic offloading frameworks like COMET. In comparison to the average cloud (EC2-west), we demonstrate that cloudlets can improve the response time by 51% and mobile energy consumption by 42%. These advantages are not limited to low-latency, one-hop WiFi networks. Even when offloading over LTE, cloudlet offloading continues to provide superior results to that of cloud.

Our results also show that offloading computation blindly to the cloud will be a losing strategy. Offloading to a distant cloud can result in even lower performance and higher energy costs than running locally on the mobile device. For interactive applications, even offloading to nearby clouds can be detrimental to performance. Such applications demonstrate most clearly that cloudlets are necessary to achieve performance and energy improvement through offloading.

With increasing interest in mobile augmented reality and cognitive assistance applications [7], which are interactive and compute-intensive, we expect cloudlets will play a greater role in mobile systems.

References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, Helsinki, Finland, 2012.

[2] G. Bradski et al. The opencv library. *Doctor Dobbs Journal*, 25(11):120–126, 2000.

[3] G. Brown. Converging Telecom & IT in the LTE RAN. White Paper, Heavy Reading, February 2013.

[4] M. S. Gordon, D. K. Hong, P. M. Chen, J. Flinn, S. Mahlke, and Z. M. Mao. Accelerating mobile applications through flip-flop replication. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 137–150. ACM, 2015.

[5] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen. COMET: Code Offload by Migrating Execution Transparently. In *Proceedings of the 10th USENIX Symposium on Operating System Design and Implementation*, Hollywood, CA, October 2012.

[6] K. Greene. AOL Flips on 'Game Changer' Micro Data Center. `http://blog.aol.com/2012/07/11/aol-flips-on-game-changer-micro-data-center/`, July 2012.

[7] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan. Towards wearable cognitive assistance. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 68–81. ACM, 2014.

[8] A. Li, X. Yang, S. Kandula, and M. Zhang. Cloudcmp: comparing public cloud providers. In *Proceedings of the 10th annual conference on Internet measurement*, pages 1–14. ACM, 2010.

[9] Monsoon Solutions. Power Monitor. `http://www.msoon.com/LabEquipment/PowerMonitor/`.

[10] Predrag Cokulov. Linpack - Android Apps on Google Play. `https://play.google.com/store/apps/details?id=rs.pedjaapps.Linpack`, 2015.

[11] Sasa D. Markovic. PI Benchmark - Android Apps on Google Play. `https://play.google.com/store/apps/details?id=rs.in.luka.android.pi`, 2015.

[12] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, 8(4):3 (Sidebar: "Help for the Mentally Challenged"), October-December 2009.

[13] B. Solenthaler and R. Pajarola. Predictive-corrective incompressible SPH. *ACM Trans. Graph.*, 28(3):40:1–40:6, July 2009.

[14] G. Takacs, M. E. Choubassi, Y. Wu, and I. Kozintsev. 3D mobile augmented reality in urban scenes. In *Proceedings of IEEE International Conference on Multimedia and Expo*, Barcelona, Spain, July 2011.

[15] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[16] Unstable Apps. CPU Benchmark - Android Apps on Google Play. `https://play.google.com/store/apps/details?id=com.unstableapps.cpubenchmark`, 2015.

[17] WattsUp. .NET Power Meter. `http://wattsupmeters.com/`.