# Cloudlet Caches: Managing State for a Microservices Based Edge Computing Platform

Joseph DeChicchis
Duke University, December 13, 2018

## I. Introduction

We have seen tremendous growth in cloud technologies over the past decade. The limited compute and storage available on end user devices have led to the development of established cloud offerings such as Microsoft Azure, Amazon Web Services, and Google Cloud. All of these public cloud offerings provide a *platform for cloud computing*. Public cloud providers leverage their established infrastructure and proprietary technological knowledge to abstract away the difficult distributed systems problems, enabling end users to take advantage of cloud storage and compute solutions with minimal need for each cloud application to reinvent complex algorithms.

Although the cloud has led to the development of many new services, the limitations of the cloud have become increasingly apparent as we have tried to develop new applications. Specifically, bandwidth and latency issues are particularly difficult to overcome in the world of mobile-cloud communication. In addition, one can argue that the cloud leads to the issue of a single point of failure. That is, because of the limited number of super-scale data centers, a failure in one can lead to a service outage in an entire geographic region. These limitations have fueled a recent re-focusing on edge computing technologies. Research has shown that cloudlets [1] can aid in real-time video denaturing [2], cognitive assistance applications [3], and VR/AR [4]. Although there are many benefits to a distributed cloudlet infrastructure, there is currently no compelling *platform for edge computing* for managing cloudlets and easily deploying services. For edge computing to reach its full potential, it is imperative that full service platforms be developed for the edge world just as they were developed for the cloud computing era.

## II. A Microservices Based Edge Computing Platform

I propose a geo-distributed microservices based edge computing platform as the ideal mechanism to manage distributed cloudlet infrastructures. Figure 1 shows several regions in a potential cloudlet platform deployment. Note that a production grade platform will have thousands of regions. Figure 2 illustrates what a cloudlet region may look like in Durham, NC where each colored dot represents a microservice instance. A production platform will have many more (potentially hundreds) cloudlets and run more microservice instances on each cloudlet. It is important to point out that cloudlets could be a dedicated server room, a wireless base station, a self-driving car with an onboard server, and much more.

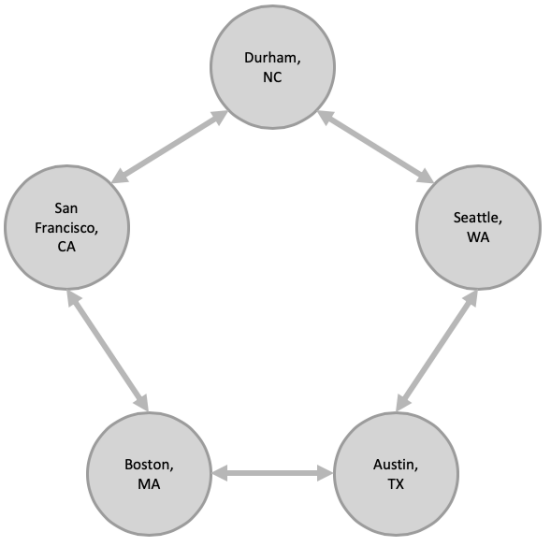


Fig. 1. Potential cloudlet regions. A production grade platform will have thousands of such regions around the world.

In this proposed edge computing platform, the developer would simply have to write and deploy microservices and the system would handle load balancing, replication, failover, system updates, health monitoring, and security and privacy. Using a microservices approach is beneficial because it inherently breaks an application into independent components. For example, a game may have a User microservice which keeps track of a user's information (e.g. username, email address) and achievements (e.g. score, level progress). Such a game may also have a Graphics Compute microservice which renders the game level. In such a scenario, the User microservice may run in the cloud because it is not bound by tight latency requirements whereas the Graphics Compute microservice may run on the edge (i.e. distributed cloudlet platform) because it is sensitive to latency constraints. Furthermore, the system would detect and predict the load (e.g. users near X, Y, and Z cloudlets in regions A and B use the game frequently) and intelligently place replicas of the Graphics Compute microservice on the appropriate cloudlets.

Building a geo-distributed microservices based edge computing platform presents many challenges: detecting cloudlet failure, managing failover, intelligent placement of microservice instances (load balancing), replication, providing a mechanism for end users to quickly find the closest instance of a

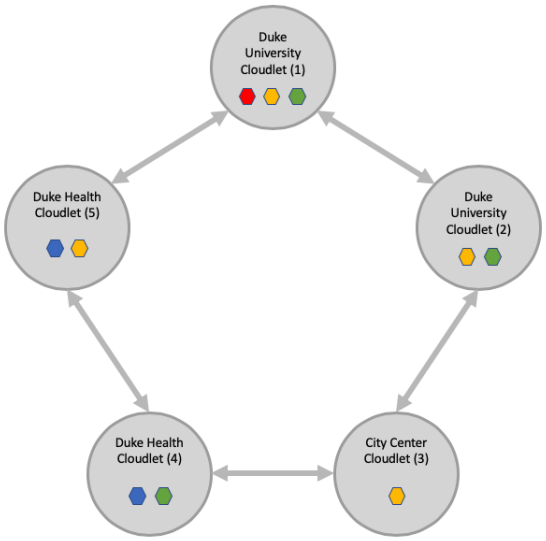


Fig. 2. A possible cluster of cloudlets in a cloudlet region (in this case Durham, NC). Each colored dot represents a microservice instance.

microservice (discovery), health monitoring, system updates, and security and privacy. Furthermore, such a platform would have to be able to support both stateful and stateless microservices. While stateless microservices are simple to implement (e.g. a Graphics Compute microservice simply receives a compute request, processes it, and returns the result), there are many challenges to implementing stateful microservices on the edge. Some questions which arise when considering a state management platform for the distributed cloudlet edge are:

- Is state replicated within the cloudlet, between cloudlets, or both?
- How do we ensure strong consistency on the edge?
- Would state transfer between cloudlets and the cloud? If so, how?
- What state is maintained on the edge and what state is maintained in the cloud?

I present cloudlet caches in the next section as a potential solution for managing state for stateful microservices in an ideal edge computing platform.

## III. Cloudlet Caches

The benefits of edge largely amount to reducing latency. This raises the question of whether we can reduce read/write latency for clients when communicating with the cloud. Currently, whenever a client makes a read or write request, they must communicate with servers which are potentially very far away, resulting in long response times. Caching reads and write on geographically closer cloudlets could potentially alleviate this problem. Although there are currently systems such as CDNs which cache static data, caching dynamic data is a much more challenging problem because we must ensure data consistency. I call cloudlets which cache dynamic data (i.e. both reads and writes) cloudlet caches and argue that such cloudlet caches are the ideal state management solution for an edge computing platform with stateful microservices.

There are two insights which support the notion of cloudlet caches:

- Eventual consistency is sufficient for many workloads.
- Strict ordering does not need to be enforced for many workloads.

Consider, for example, a social network. When someone makes a new post, everyone in the world does not need to see it instantaneously (i.e. within milliseconds). It is acceptable as long as everyone sees the new post within a few seconds. When a comment is posted, it is not important to know that comment A was posted 5 milliseconds before commend B. Instead, it is sufficient if the system can deduce, perhaps through timestamps, that comment A was posted a second before comment B. Of course, not all systems behave in this way. An airline booking system must have a strict ordering of purchases, and its system must display the most up-to-date seat availability. Otherwise, an airline may have some angry customers calling their call centers because a customer was not able to purchase a seat which appeared empty on their website. Nonetheless, I believe cloudlet caches could be useful for real-world applications because there are many systems which can operate without the guarantees of strong consistency and strict ordering.

Cloudlet caches would provide state management for a microservices based edge computing platform by exposing APIs that microservice developers can use to make calls to persisted, replicated, and cached storage. That is, irrespective of where the microservice is running (i.e. on a cloudlet or in the cloud), the platform would guarantee the fault tolerance and eventual consistency of any storage read/write commands. This is similar to the approach Service Fabric takes by providing Reliable Collections [5]. The next section describes several approaches to how one may implement state management for cloudlet caches.

Figure 3 illustrates the vertical layout of a two-level cloudlet cache architecture. All state is eventually consistent, with the cloud acting as the arbiter of consistency (i.e. the cloud runs some sort of consensus protocol). Each hexagon represents state for a given microservice. In this scenario, the $L_2$ caches would run the logic for microservices and each microservice would interface with the cloudlet cache state management system through APIs. The cloudlet cache edge computing platform would ensure writes are propagated to the cloud. The cloud keeps a store of which cloudlets are caching data and selectively broadcast updates to reduce polling overhead. Furthermore, each cloudlet is fault tolerant (i.e. cloudlets consist of clusters of servers) so a client which writes to a cloudlet does not have to wait for the data to be committed to the cloud (i.e. writes to cloudlets are successful once it's committed within the cloudlet cluster).

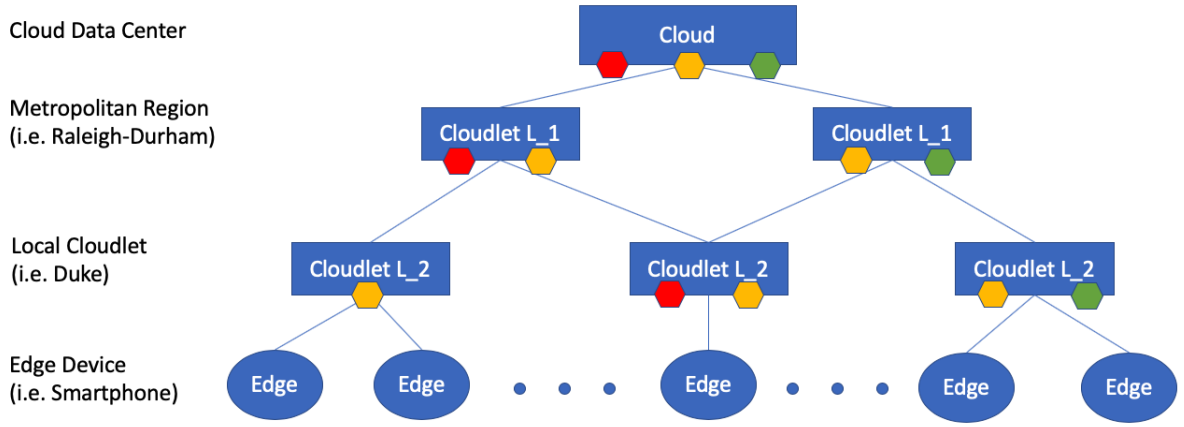


Fig. 3. The vertical layout of a two-level cloudlet cache architecture. All state is eventually stored in the cloud, and cloudlets at each layer cache state for the lower layers. Each hexagon represents state for a certain microservice.

## IV. Managing State on Cloudlet Caches

Various consensus protocols have traditionally been used to achieve consistency. Consensus protocols take advantage of the fact that given a deterministic state machine, we can achieve consensus if we can ensure that state transitions are performed on each replica in the same order [6]. Several algorithms have been proposed to solve this problem.

The popular Raft [7] consensus protocol is an easy to understand and implement version of the Paxos [6] consensus protocol. Like many implementations of Paxos, Raft is a Multi-Paxos system which means there is a single leader that clients must communicate with. Having a single leader greatly simplifies the algorithm. However, things do not have to be this way. Based on the Single-Decree Paxos model, it is possible to construct a system which achieves consensus even though writes and reads can happen at any replica. Such a system has several benefits such as being able to load balance among replicas and achieving greater availability. EPaxos [8] is a Paxos variant which allows for such a scenario. Whether a client can communicate with a single leader or multiple replicas can greatly affect scalability.

Two goals must be achieved to build a cloudlet based dynamic caching system. One is that the cloudlet itself must be highly available. The other is that data on each cloudlet and the cloud must be eventually consistent. This paper investigates the performance tradeoffs between using Raft and EPaxos to ensure consistency across cloudlets and the cloud.

Figure 4 presents the traditional single-leader multiple-client approach where the cloud maintains consistency using Raft (architecture 1). Figure 5 shows a slightly different multiple-"leaders" multiple clients approach where the cloud maintains consistency using EPaxos (architecture 2). In both of these cases no cloudlets are being used to cache data. Figure 6 presents the cloudlet caching model were clients connect to cloudlets instead of the cloud (architecture 3). In this case, the cloud implements Raft so all cloudlets must communicate with the same leader. Figure 7 illustrates the cloudlet caching model as well but cloudlets can communicate with any cloud replica (architecture 4).

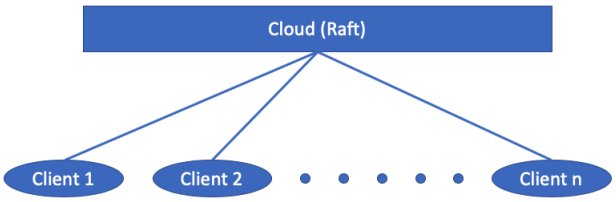


Fig. 4. Architecture 1: Scenario where there is no cloudlet caching and the cloud achieves consensus using Raft. Client must communicate with single leader.

Note that although a cloudlet cache architecture could have multiple layers, as shows in figure 3, this paper only considers one layer cloudlet cache architectures for evaluation purposes. Furthermore, the main reason Raft and Paxos are being considered as potential consensus algorithms for this paper is to evaluate whether the cloud having only a single leader presents a more significant bottleneck than a cloud which has multiple-"leaders" which clients can communicate with. Although the cloud would ultimately be a bottleneck in any cloudlet cache deployment, the negative effect of a central cloud on system throughput may be lessened by leveraging various consensus protocols.

## V. Evaluation Methodology

I implemented the four architectures presented in the previous section to evaluate how each of them impact system throughput. The core algorithms of both Raft [7] and EPaxos [8] were used to implement a consistent key-value store in C#. Certain optimizations such as snapshotting and features required in production environments such as cluster reconfigurations were not implemented. The key value store supported GET, SET, and DELETE operations.

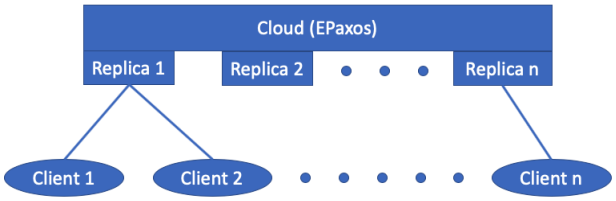


Fig. 5. Architecture 2: Scenario where there is no cloudlet caching and the cloud achieves consensus using EPaxos. Client can communicate with any replica.

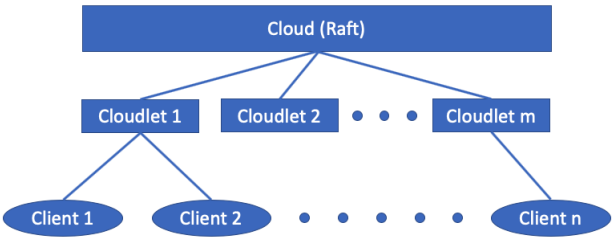


Fig. 6. Architecture 3: Scenario where there is cloudlet caching and the cloud achieves consensus using Raft. Clients can communicate with any cloudlet. Each cloudlet must communicate with single leader.

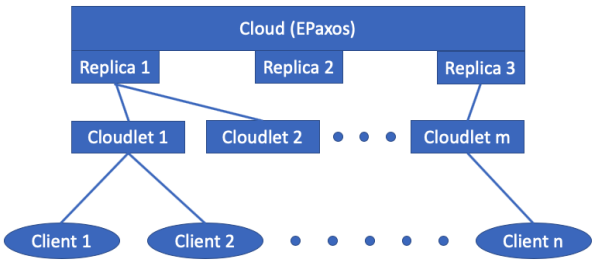


Fig. 7. Architecture 4: Scenario where there is cloudlet caching and the cloud achieves consensus using EPaxos. Clients can communicate with any cloudlet. Each cloudlet can communicate with any replica.

Each cloud cluster, both Raft and EPaxos, consisted of three nodes. Clients were implemented as a wrapper around a cache which essentially forwarded requests to the cloud if data was not available in the local cache and returned data to the client if the data was in the local cache (clients did not implement clusters for evaluation purposes).

All evaluations were conducted with two clients. Two cloudlet caches were used to evaluate architectures three and four. In the case where Raft was used for the cloud cluster all clients and cloudlets communicated with the single leader (architectures one and three). In the case were EPaxos is used for the cloud cluster, each client and cloudlet cache communicated with a distinct node (architectures two and four). Each client communicated with a distinct cloudlet cache for both cases.

Throughput was measured as the number of requests a client completed per second. Clients made consecutive requests (i.e. a client sent a new request as soon as a previous one finished). A client request consisted of a key, value, and request type. The key for a given request was randomly chosen among 26 uppercase characters. The value of a request was a random four-character string consisting of uppercase characters. The type of operation for a given request was also randomly chosen. Requests were made for a duration of 5 minutes and throughput recorded every second.

The following latencies were simulated:

- Communication between client and cloud: 70 ms.
- Communication between client and cloudlet: 30 ms.
- Communication between cloudlet and cloud: 50 ms.
- Communication between nodes in a cloud cluster: 5ms.

Evaluations were conducted on a 2017 MacBook Pro and using two Raspberry Pies. For each architecture, evaluations were conducted solely on the MacBook Pro (scenario A) and using a MacBook Pro and one or two Raspberry Pies (scenario B). In scenario A, the clients, cloudlet caches, and cloud ran as separates threads in a single process. In scenario B, the clients ran on the 2017 MacBook Pro, and the cloudlet caches and cloud each ran on a separate Raspberry Pie.

## VI. Results

Throughput (requests completed per second) was plotted as a function of time for each of the evaluations. These plots one for each of the eight evaluations, two per architecture are presented in figures 8, 9, 10, 11, 12, 13, 14, and 15. Table I summarizes the mean throughput and variance for each of the eight evaluation based on an aggregate of client 0 and client 1 data.

Throughput was greater when running the clients, cloudlet caches, and cloud all on the MacBook Pro. Using Raspberry Pies for the cloudlet caches and cloud reduced throughput as expected since Raspberry Pies are much more resource constrained than a MacBook Pro and communicating over the network adds more variance in communication latency. In addition, using cloudlet caches increased throughput for both Raft and Paxos as expected since communication latency between the client and cloudlet cache is lower than communicating with the cloud once data is in the cache (it takes longer for clients to retrieve data if the data is not cached since the client has to send a request to the cloudlet then the cloudlet has to send a request to the cloud).

The first thing to note is that the variance in throughput was low on all evaluations but when running cloudlet caches and the cloud on Raspberry Pies (figures 13 and 15). This can be explained by performance limitations of the Raspberry Pies since such large variances were not observed when only using the MacBook Pro to run both the cloudlet caches and cloud. The Raspberry Pies are probably unable to handle the volume of requests as throughput increases (i.e. as number of requests the Raspberry Pi has to handle increases). In addition, there may have been some network issues which lead to higher variance in these cases.

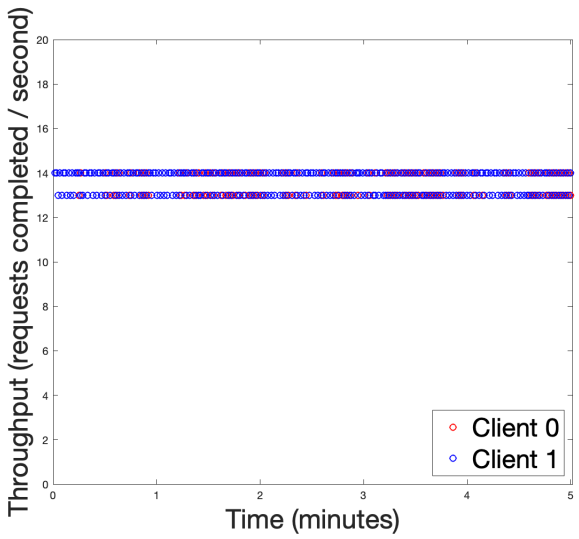


Fig. 8. Architecture 1 (Raft cloud which clients communicate directly with) evaluated on a MacBook Pro. $\mu = 13.56$ and $\sigma = 0.50$ (calculated based on aggregate of client 0 and client 1 throughput).

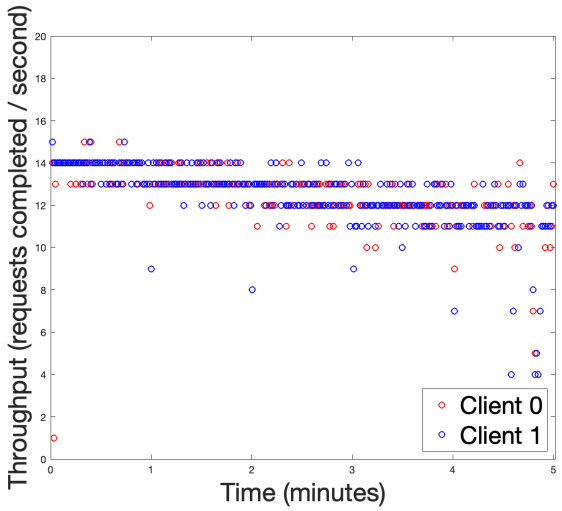


Fig. 9. Architecture 1 (Raft cloud which clients communicate directly with) evaluated on a MacBook Pro (clients) and Raspberry Pi (cloud). $\mu = 12.42$ and $\sigma = 1.59$ (calculated based on aggregate of client 0 and client 1 throughput).

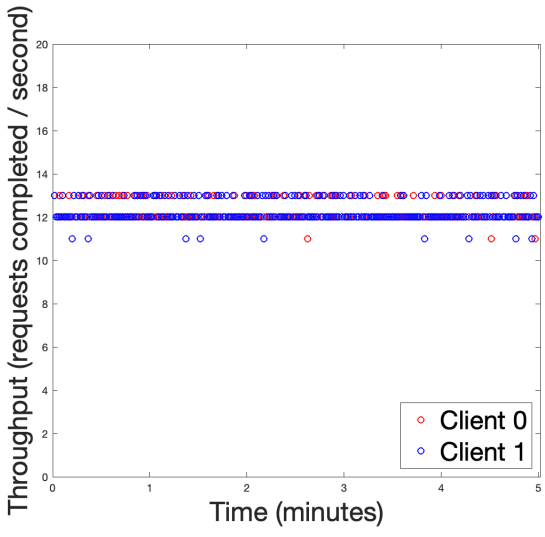


Fig. 10. Architecture 2 (EPaxos cloud which clients communicate directly with) evaluated on a MacBook Pro. $\mu = 12.27$ and $\sigma = 0.49$ (calculated based on aggregate of client 0 and client 1 throughput).

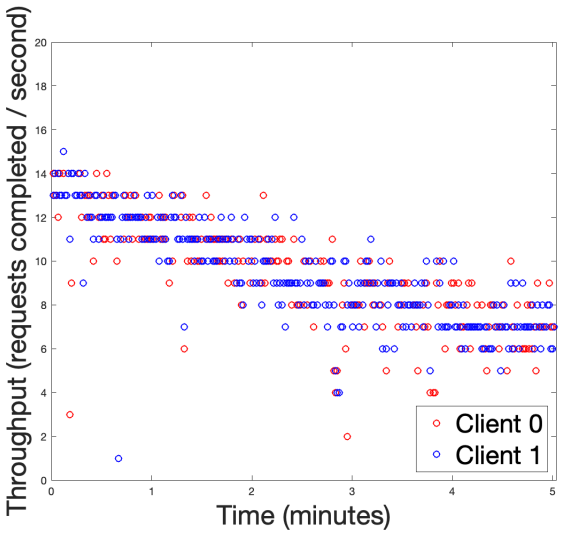


Fig. 11. Architecture 2 (EPaxos cloud which clients communicate directly with) evaluated on a MacBook Pro (clients) and Raspberry Pi (cloud). $\mu = 9.33$ and $\sigma = 2.44$ (calculated based on aggregate of client 0 and client 1 throughput).

Another interesting thing to note is that performance on the Raspberry Pies seemed to decrease over time (figures 9 and 11). This too can probably be explained by the resource limitations of the Raspberry Pies.

Overall, the results indicate that cloudlet caches increase system throughput. However, there was not a significant performance difference between using Raft or EPaxos as the consensus algorithm.

## VII. Discussion

Although the results from the previous section indicate that cloudlet caches have the potential to increase system throughput, it is surprising that using EPaxos did not increase throughput compared to using Raft as the consensus protocol. One would imagine that using a more distributed consensus protocol such as EPaxos would help balance requests (i.e. help reduce the cloud bottleneck) and increase overall system performance.

Using EPaxos may not have increased throughput since its strengths are based largely on its ability to balance load on a wide area network. In wide area networks, clients in the United States may be able to access a data center in Virginia much faster than a client in Japan can since in a traditional consensus protocol (e.g. Raft), the cluster of machines are all located in the Virginia data center. However, EPaxos enables clusters

| Evaluation Architecture | Mean Throughput (requests completed / second) $\mu$ | Variance $\sigma$ |
|---|---|---|
| Arch 1 (Raft without cloudlet caches), only MackBook | 13.56 | 0.50 |
| Arch 1 (Raft without cloudlet caches), MackBook & Pi | 12.42 | 1.59 |
| Arch 2 (EPaxos without cloudlet caches), only MackBook | 12.27 | 0.49 |
| Arch 2 (EPaxos without cloudlet caches), MackBook & Pi | 9.33 | 2.44 |
| Arch 3 (Raft with cloudlet caches), only MackBook | 29.71 | 1.08 |
| Arch 3 (Raft with cloudlet caches), MackBook & Pies | 25.33 | 15.26 |
| Arch 4 (EPaxos with cloudlet caches), only MackBook | 29.56 | 0.99 |
| Arch 4 (EPaxos with cloudlet caches), MackBook & Pies | 17.95 | 19.80 |

TABLE I
MEAN THROUGHPUT AND VARIANCE FOR THE EIGHT EVALUATIONS OVER A 5 MINUTE PERIOD (CALCULATED BASED ON AGGREGATE OF CLIENT 0 AND CLIENT 1 THROUGHPUT).

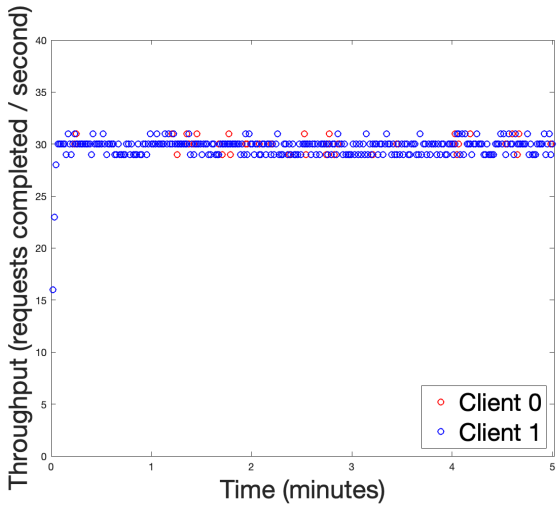


Fig. 12. Architecture 3 (Raft cloud and cloudlet caches which clients communicate with) evaluated on a MacBook Pro. $\mu = 29.71$ and $\sigma = 1.08$ (calculated based on aggregate of client 0 and client 1 throughput).

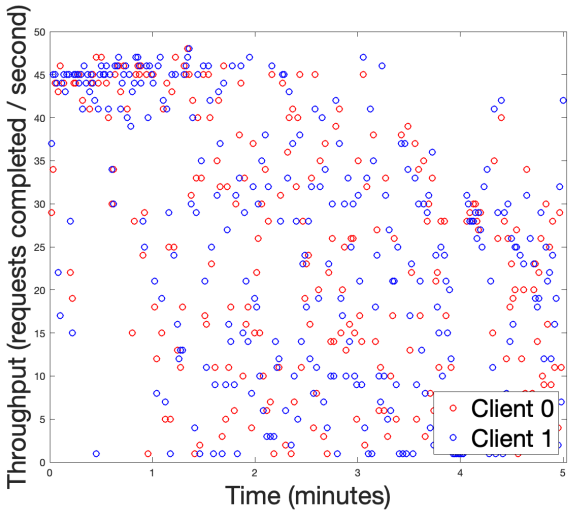


Fig. 13. Architecture 3 (Raft cloud and cloudlet caches which clients communicate with) evaluated on a MacBook Pro (clients), a Raspberry Pi (cloudlet caches), and another Raspberry Pi (cloud). $\mu = 25.33$ and $\sigma = 15.26$ (calculated based on aggregate of client 0 and client 1 throughput).

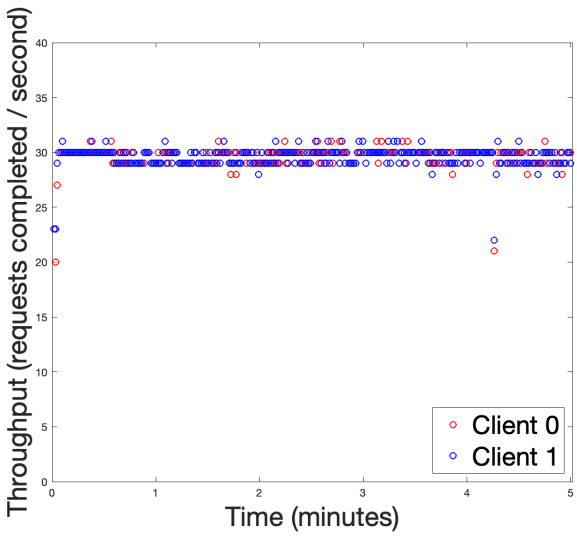


Fig. 14. Architecture 4 (EPaxos cloud and cloudlet caches which clients communicate with) evaluated on a MacBook Pro. $\mu = 29.56$ and $\sigma = 0.99$ (calculated based on aggregate of client 0 and client 1 throughput).

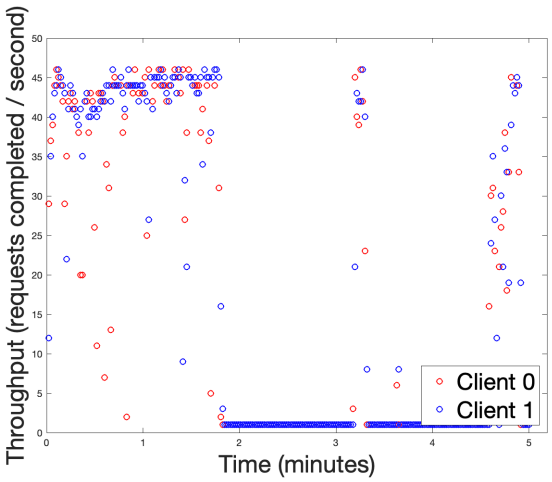


Fig. 15. Architecture 4 (EPaxos cloud and cloudlet caches which clients communicate with) evaluated on a MacBook Pro (clients), a Raspberry Pi (cloudlet caches), and another Raspberry Pi (cloud). $\mu = 17.95$ and $\sigma = 19.80$ (calculated based on aggregate of client 0 and client 1 throughput).

with good performance even if the machines are spread out between the United States and Japan, hence reducing access latencies for users in Japan. The advantages of EPaxos were

probably not observed since the evaluation for this paper assumes that cloud clusters could communicate quickly over a local network (i.e. only 5ms delays between nodes). Furthermore, this result implies that the potential load balancing characteristic of EPaxos compared to Raft is negligible when communication between nodes is fast. Moreover, consensus algorithms do not need to operate on a wide area network when cloudlet caches are used since cloudlet caches help reduce access latencies for users.

## VIII. Related Work

### A. State Management

Managing state within a distributed cluster of machines is largely a solved problem. Raft [7], a simplified version of the (infamous) Paxos [6] algorithm, achieves leader election and consensus through a distributed election mechanism and log replication. ZooKeeper [9] was developed to abstract away many of the difficult distributed systems problems which arise in distributed high-performance compute scenarios. EPaxos [8] was developed as a leaderless consensus protocol for wide area networks. However, none of these systems were developed for the edge scenario and the programming models they motivate do not directly translate from compute clusters in the cloud to the distributed edge world.

The federation and replication components of Service Fabric [5] solve common distributed computing problems such as leader election and state replication. While its core algorithms were developed to support edge scenarios [10], Service Fabric has matured into a system for managing state in the cloud.

### B. Microservices Platform

Service Fabric is the most comprehensive microservices management platform on the market. It supports full lifecycle management of applications composed of both stateless and stateful microservices [5]. However, the specific microservices programming model Service Fabric has adopted over time is not meant for edge computing. Nonetheless, its core algorithms are agnostic to higher levels of state management abstraction, making it a good candidate for developing state management on the edge.

Kubernetes [11] are a popular container orchestration system. Although Kubernetes embraces the idea of containerized microservices, it is not a full fledge microservices management platform like Service Fabric. Instead, Kubernetes is purely a container orchestration system and other tools have to be deployed on top of it to provide state management and other useful features which a microservices platform should have built into it. Even though some have argued that Kubernetes should be used as a base for managing containers on the edge [12]. I believe this is the wrong approach since one would want a microservices based edge computing platform to provide more than just container orchestration on the edge.

### C. Edge Computing Platforms

OpenStack++ [13] was developed to enable faster testing of edge computing applications by extending the OpenStack [14] platform to support rapid provisioning of VM images, VM migration across cloudlets, and cloudlet discovery by mobile clients. However, OpenStack++ does not aim to build a true platform for edge computing which abstracts issues such as replication, state management, health monitoring, and much more.

Akamai Technologies has a Cloudlet Applications [15] offering, but the applications are pre-made edge scenarios which customers can easily plug into and they do not support the execution and management of truly custom applications.

### D. IoT Platforms

There are several IoT management and deployment platforms such as AWS Greengrass [16], Azure IoT Edge [17], and EdgeX [18]. However, these IoT platforms do not aim to provide a platform for edge computing in the cloudlet space. Instead, they focus on deploying services on IoT devices, and managing communication between IoT devices and the cloud.

### E. State Management on the Edge

CloudPath [19] is an idea which is similar to cloudlet caches. The authors propose the use of multiple levels of caches which edge devices can communicate with. However, the authors try to guarantee strict ordering through timestamps which cannot be done with total confidence due to clock skews. In addition, CloudPath suggests using polling to ensure data freshness which is an unnecessary overhead. Cloudlet caches avoid these issues by not guaranteeing strict ordering and broadcasting updates.

## IX. Future Work

Future work relating to cloudlet caches should investigate how much of a bottleneck the cloud may be and whether adding more cloudlet cache layers improves or hinders performance. In addition, evaluations should be conducted using real-world traces over longer periods of time.

There is much work to be done relating to a microservices based edge computing platforms. Although cloudlet caches present a potential solution for managing state on the edge, there may be better approaches which increase throughput or provide stronger consistency guarantees without degrading performance. Moreover, complicated problems such as load balancing, replication, discovery, health monitoring, system updates, and security and privacy would have to be addressed for an edge computing platform to be production ready.

## X. Conclusion

We are desperately in need of an edge computing platform with a mature ecosystem for developers similar to the cloud computing platforms we see in production environments today. Taking advantage of containerized microservices is the best solution to an edge computing platform and current technologies for microservice and container management as well as consensus could potentially be used to manage such a platform and provide an abstracted state management system for stateful microservices.

I introduced cloudlet caches as one potential solution for managing state on the cloudlet edge. Evaluations of the four architectures presented above showed that cloudlet caches have the potential to increase overall system throughput for an edge computing platform. However, somewhat surprisingly, Raft, the more traditional Multi-Paxos consensus algorithm, performed similarly to the more distributed leaderless EPaxos consensus algorithm indicating that there may not be a need for novel consensus protocols to avoid the cloud bottleneck in a cloudlet cache deployment.

## REFERENCES


[1] Mahadev Satyanarayanan et al. 2009. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing* 8, 4 (October 2009), 14-23.

[2] Christopher Streiffer, et al. 2017. "ePrivateeye: to the edge and beyond!," *Proceedings of the Second ACM/IEEE Symposium on Edge Computing* (SEC '17). ACM, New York, NY, USA, Article 18, 13 pages.

[3] Zhuo Chen, et al. 2017. "An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance," *Proceedings of the Second ACM/IEEE Symposium on Edge Computing* (SEC '17). ACM, New York, NY, USA, Article 14, 14 pages.

[4] X. Hou, Y. Lu and S. Dey, "Wireless VR/AR with Edge/Cloud Computing," *Proceedings of the 26th IEEE International Conference on Computer Communication and Networks*, Vancouver, Canada, July 2017.

[5] Gopal Kakivaya, et al. 2018. "Service fabric: a distributed platform for building microservices in the cloud," *Proceedings of the Thirteenth EuroSys Conference* (EuroSys '18). ACM, New York, NY, USA, Article 33, 15 pages.

[6] Leslie Lamport. 1998. *The part-time parliament*. ACM Trans. Comput. Syst. 16, 2 (May 1998), 133-169.

[7] Diego Ongaro and John Ousterhout. 2014. "In search of an understandable consensus algorithm," *Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference* (USENIX ATC'14), Garth Gibson and Nickolai Zeldovich (Eds.). USENIX Association, Berkeley, CA, USA, 305-320.

[8] Iulian Moraru, David G. Andersen, and Michael Kaminsky. 2013. *There is more consensus in Egalitarian parliaments*. In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13). ACM, New York, NY, USA.

[9] Patrick Hunt, et al. 2010. "ZooKeeper: wait-free coordination for internet-scale systems," *Proceedings of the 2010 USENIX conference on USENIX annual technical conference* (USENIXATC'10). USENIX Association, Berkeley, CA, USA, 11-11.

[10] Gopal Kakivaya explaining the overall architecture of Service Fabric. Available: https://www.youtube.com/watch?v=MrfcP6dS6mU. [Accessed Oct. 1, 2018].

[11] Kubernetes. Available: https://kubernetes.io. [Accessed Dec. 11, 2018].

[12] Edge Computing and the Cloud-Native Ecosystem. Available: https://thenewstack.io/edge-computing-and-the-cloud-native-ecosystem/. [Accessed Dec. 11, 2018].

[13] Ha, Kiryong and Mahadev Satyanarayanan. *OpenStack++ for Cloudlet Deployment*. 2015.

[14] OpenStack. Available: https://www.openstack.org. [Accessed Oct. 1, 2018].

[15] Akamai's Cloudlet Applications. Available: https://www.akamai.com/us/en/products/web-performance/cloudlets/. [Accessed Oct. 1, 2018].

[16] AWS Greengrass. Available: https://aws.amazon.com/greengrass/. [Accessed Oct. 1, 2018].

[17] Microsoft's Azure IoT Edge. Available: https://azure.microsoft.com/en-us/services/iot-edge/. [Accessed Oct. 1, 2018].

[18] EdgeX. Available: https://www.edgexfoundry.org. [Accessed Oct. 1, 2018].

[19] Seyed Hossein Mortazavi, et al. 2017. "Cloudpath: a multi-tier cloud computing framework," *Proceedings of the Second ACM/IEEE Symposium on Edge Computing* (SEC '17). ACM, New York, NY, USA, Article 20, 13 pages.