Analytics of Performance and Data Quality for Mobile Edge Cloud Applications

Hong-Linh Truong
Faculty of Informatics, TU Wien, Austria
hong-linh.truong@tuwien.ac.at

Matthias Karan*
Alumnus, TU Wien, Austria
matthias.karan@gmail.com

***Abstract*—Emerging edge/fog computing models have fostered new types of applications whose software components and dependent services are provisioned across distributed edge and cloud infrastructures. The design of mobile edge cloud systems is complex, thus it is important to understand suitable deployment models and test them. Since mobile edge cloud computing and its deployments are quite new, there is a lack of techniques and knowledge about possible deployments, configurations, and performance evaluation. In this paper, we present our experiences on studying the impact of performance and data quality for mobile edge cloud systems. We use a mobile edge cloud cornering assistance (MECCA) application to examine various performance and data quality impact. In this paper, we explain how by using MECCA to test performance and data quality, we draw key issues and steps in analytics of edge cloud applications and lessons learned for mobile edge computing application testing.**

I. Introduction

Recent integration between mobile computing, edge computing and cloud computing have fostered the development of complex mobile edge cloud applications. While most existing works still focus on edge offloading applications [1], [2], many researchers have also started to address complex interactions between the edge and cloud in an integrated manner [3], [4]. From the development perspective, not only we have to examine suitable deployment models for software components and their dependent third party services in edge and cloud infrastructures but also to carry out several tasks of performance evaluation and testing to detect possible issues and to optimize the design and the deployment.

Many edge-cloud applications combine features from edge and cloud so there are various factors that influence the deployment, provisioning and analysis. In this paper, we focus on the impact of performance and data quality for edge cloud applications. Our work experiments performance and data quality for complex mobile edge cloud applications due to the lack of tools and the complexity of such applications. We focus on realistic applications in the connected vehicles domain that are suitable for edge cloud computing, and we study performance and data quality impacts in the design and deployment models. We select a mobile edge cloud cornering assistance (MECCA) application which includes features from IoT, big data services, and streaming data processing that are provisioned in edge and cloud infrastructures. Our contributions are to present steps/techniques for analytics of mobile edge cloud applications and lessons learned through real examples.

The rest of this paper is organized as follows: Section II describes our mobile edge cloud application. Section III explains methods for mobile edge cloud performance and data quality evaluation. Section IV presents experiences for performance and Section V presents experiences for data quality. We discuss the related work in Section VI. We conclude the paper and outline our future work in Section VII.

II. Overview of the Edge-Cloud Cornering Assistance (MECCA) Application

A. Architecture

To present our analytics of performance and data quality in mobile edge cloud applications, we select cornering assistance applications. Figure 1 shows the overall architecture of the mobile edge cloud cornering assistance (MECCA) application. *Clients* hosted in vehicles connect to a service registry to find a *Recommendation Service* which is responsible for recommending suitable speeds for all upcoming curves around a given location. The recommendation is based on the location data provided by the clients. To calculate a recommended speed, *Recommendation Service* needs information about upcoming curves and the current weather. An *External Database (DB)* is used to permanently store curve results and is queried, as a cache, by *Recommendation Service*. Since the database will be accessed from multiple services running on multiple nodes, it needs to be highly available, scalable and easily maintainable. In case DB has no curves stored yet, a *Detection Service* is requested to detect and calculate detailed information about upcoming curves, based on (i) the location data, indicating the location of the Clients, given by a *Recommendation Service* and (ii) map data from an external *Maps Provider*.

In MECCA, a *Service Registry* is needed to support services discovery for various service instances as we have many instances from the above-mentioned services. To enable high scalability and support the underlying edge/cloud-infrastructure, the above-mentioned services can be deployed to different containers/VM nodes. Nodes export metrics about their current resource usage to a monitor system: *Monitoring* is accessible by the other services in the system.

*Work performed when the author was a master student at TU Wien

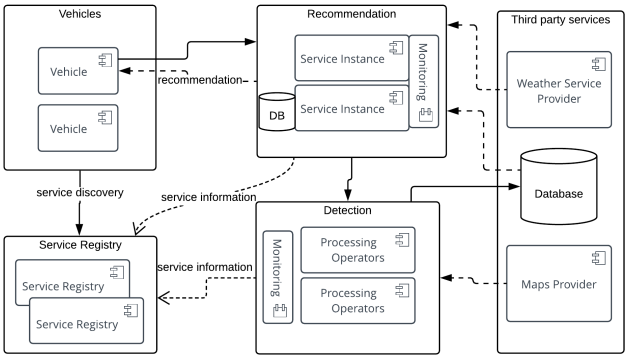


Fig. 1. Overall of main components of MECCA

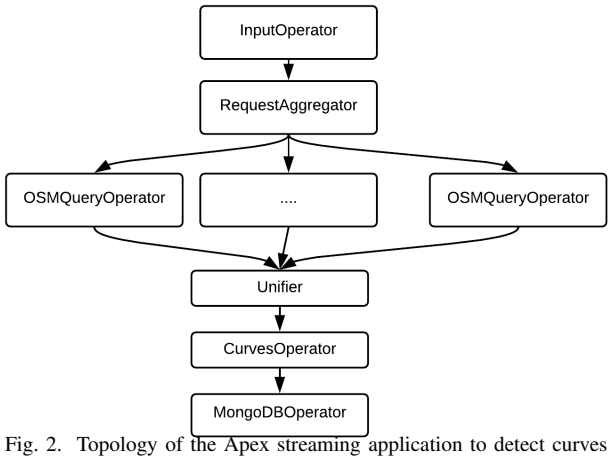


Fig. 2. Topology of the Apex streaming application to detect curves

B. Algorithms for cornering recommendations

The cornering recommendation algorithms have been implemented but in this paper we do not present the detail. Basically, we use a streaming processing to implement the detection of the curve. Figure 2 presents the topology of operators in the streaming processing, implemented with Apache Apex[1]. In particular, the operator `OSMQueryOperator` is used to query points of roads from Open Street Map (OSM – the Maps Provider). This operator can be parallelized –using partitioning techniques – and results from instances of this operator will be unified for the curve determination (`CurvesOperator`). The curve detection algorithm calculates properties of the curve including start/center-end points, radius and length. From every incoming position (specified in data tuples), it is possible that many curves can be detected and will be emitted to `MongoDBOperator` which stores them to the MongoDB database using the commons library code. To reduce the number of connections to the database, `MongoDBOperator` caches curves for a configurable time interval and then stores them in one single batch.

C. Prototype implementation

An overview of key technologies and components of MECCA[2] is given in Figure 3. *Recommendation Service* is implemented a gRPC and *Detection Service* is implemented as streaming application using Apache Apex. We use dockers as our main underlying compute nodes.

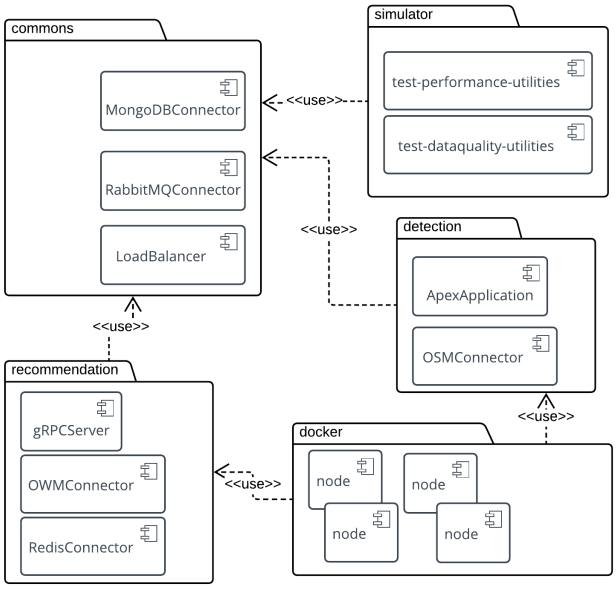


Fig. 3. Key software and infrastructural components

III. Methods for Analyzing Edge Cloud Performance and Data Quality

A. Step 1 – Deciding Edge Cloud Deployment Models

One of the challenges for a complex mobile edge cloud application is how to partition which services of the application in the edge and in the cloud infrastructures. Analysis of performance and data is important for determining deployment models and vice versa. We consider two main models:

- cloud-deployment model: all services are deployed in clouds, whereas clients are in mobile devices. We distinguish two cases: the application owner deploys his/her own services (i) not in the same data center with and (ii) in the same data center with other third party services.
- edge-cloud deployment: application services are deployed in both the edge and cloud. Similarly, application services might be co-located or not with third party services.

Note that cloud-deployment model testing is very useful for the design of edge-cloud deployment because we can detect suitable services in clouds that can be migrated to the edge or optimized for cloud deployment. That is the reason why in this paper we also focus on testing cloud-deployment.

Discussion: We recommend to test with the cloud deployment first before deciding edge-cloud deployment and testing. This helps to make better decision on where components should be deployed and which configurations are suitable for them. In many situations, like in MECCA, we have to rely on several third party services. In the production deployment, one might not be able to influence where such third party services will be deployed – as they might be fixed. Therefore, in testing performance, if possible, one should also replicate such third party services or deploy own services close to

[1]https://apex.apache.org/

[2]An open source at https://github.com/rdsea/EdgeCorneringAssistance

the third party services to examine the connectivity impact between the application under test and its third party services. The replication of many third party services, especially the infrastructural ones, can be achieved due to the availability of data and software services. In our tests, we replicate third party services when possible for cloud-only deployment.

B. Step 2 – Criteria for Evaluating Edge Cloud Deployments

There could be many different criteria for evaluating edge cloud applications. We focus on **performance** and **data quality** in this paper. For the developer and provider of edge cloud applications, the important analytics questions for evaluating edge cloud deployment would be: (i) service response time and failure between edge and cloud, (ii) accuracy of results w.r.t data accuracy and device availability and (iii) impact of network performance and reliability.

Discussion: As we focus on applications, it is important to determine application metrics that can be used to map to system metrics. Monitoring of complex applications requires multi-layer and cross-system instrumentation, monitoring tools. In this view, we found that Prometheus and Grafana are very suitable for mobile edge cloud application monitoring.

C. Step 3 – Preparing Testing Data

The first step is to prepare inputs for testing. For example, this paper, we evaluate MECCA by using tracks from the Austrian Road Safety Board (KFV) to extract real trip information, including *Timestamp*, *Latitude*, *Longitude*. The second aspect is to perform data quality experiments, one needs to have reference data. For example, we used a set of measured curves from [5] to compare again the output of algorithms. Third, we also need to prepare how we can use third party services. For example, to receive latitude and longitude values of curves for our evaluation, each curve was located manually by using the provided image and OpenStreetMaps. In case a curve had two values for the radius, we simply used the mean value. Finally, we need to prepare data related to mobile clients. For example, in MECCA, for simulating the location of the client (the vehicle), we need location data of the curves the vehicle goes. To this end, we sent listed curve locations to a public available route service called Project OSRM [6] which calculates the fastest route between the given points and returns a list of IDs of OSM nodes. To receive GPS locations from the resulting node IDs, we used the Overpass API [7].

Discussion: Similar to today's cloud applications, we expect that, in typical mobile edge cloud applications, third-party services will be utilized. Thus, we also have to study possible performance and data quality issues due to such services.

D. Step 4 – Prepare symbiotic testing

Key steps are to prepare the right setting for testing using simulation and real deployment (symbiotic systems) [8] because due to the lack of real mobile edge cloud testbeds, we should not assume that everything can be done in the real systems. In MECCA, we developed a testing application. Each test creates a set of clients, which send GPS tuples available

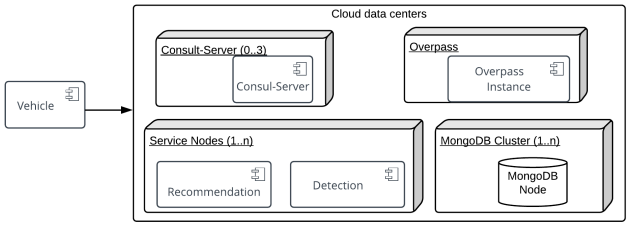


Fig. 4. Cloud-only-deployment model

.csv-files and each client accesses *Recommendation Service* via a gRPC-stub.

Discussion: There is a question of if we can rely on simulation tools for performance and data quality analytics. Although there exist some tools [9], we believe that they cannot be used to test the complexity of mobile edge cloud applications, especially w.r.t quality of data and complex algorithms depending on data from different third party services.

IV. Performance Analytics

A. Metrics

For example, in MECCA, Table I shows the metrics that are of interest for this test.

TABLE I
EXAMPLES OF METRICS USED FOR PERFORMANCE EXPERIMENTS

Metric Name	Description
response-time	Average time (in ms) between all sent recommendation requests and all received responses. This is used to measure the performance of service responses.
result-time	Average time (in ms) between all sent recommendation requests and only successful received responses that contain curves.
response-ratio	Ratio of all sent requests (recommendation & poll) to any received response. It is used to measure the reliability of messages
result-ratio	Ratio of all sent requests (recommendation only) to received curve responses. This is used to measure the accuracy of the analytics

B. Performance of cloud-only deployment model

1) Setting Testbed: We use multiple virtual machines from Google Cloud Platform (GCP) for the cloud-only deployment. Figure 4 shows the deployment configuration where all services are deployed in the same data center. To simulate a large number of clients, we emulated clients on different machines (local and cloud ones). Tables II and III present configurations of resources and services for the performance experiments.

Discussion: To keep the system independent from mobile clients and provide a more realistic setting, the clients should be deployed in different regions. The available internet speed at the cloud for clients is very fast (during the test runs it varied between 75mbps and 300mpbs). Therefore, when we need to simulate networks in the data centers we need to use different techniques. For example, the maximum network bandwidth can be throttled to Long Term Evolution (LTE) using Apple's Network Link Conditioner [10]. To give a realistic answer for mobile edge cloud applications, we should control the network with LTE capabilities only.

TABLE II
RESOURCES AND SERVICE DEPLOYMENT FOR PERFORMANCE TESTS

Name	Service	Type	Specs
Emulated Local Client		MacBook Air	8GB RAM, 2 x 1.8GHz, IntelCore i5, Network: LTE
Emulated Cloud Client	GCP	n1-standard-1	default, Network: 75-300mbps
Recommendation	GCP	n1-standard-1	default
Detection	GCP	n1-standard-4	default
OverpassAPI	GCP	n1-standard-8	default
MongoDB	GCP	n1-standard-1	default

TABLE III
SERVICES PARAMETERS FOR PERFORMANCE EXPERIMENTS

Software Component	Parameter	Value
	LocalSearchBoundingBox	6
	Poll Delay	3s
	Max Polls	3 (6)
	GRPC Timeout	5s
Recommendation	Find Curves Mode	"geohash"
	Geohash-Precision	6
	Simulate WeatherAPI	true
	WeatherBoundingBoxSize	4
Detection	AggregationTimeWindow	1s
	Aggregation BB Size	6
	OSMPartitions	5
	Overpass-Server	private
	AngleThreshold	2

TABLE IV
TEST RESULTS OF PERFORMANCE. *C/L* INDICATES THE RATIO OF CLOUD CLIENTS VERSUS LOCAL CLIENTS.

Run	Vehicles	C/L	rT	rR	resT	resR	cache
1	100	50:50	288	97%	351	87%	full
2	500	50:50	361	90%	441	78%	full
3	1000	50:50	383	82%	391	70%	full
4	1500	67:33	958	77%	1021	65%	full
5	2000	75:25	4224	11%	4342	9%	full
7	100	50:50	174	99%	1457	61%	empty
8	500	50:50	184	99%	527	64%	empty
9	1000	50:50	226	99%	/	0%	empty
10	500	0:100	322	88%	383	77%	full
11	500	0:100	321	99%	726	83%	empty

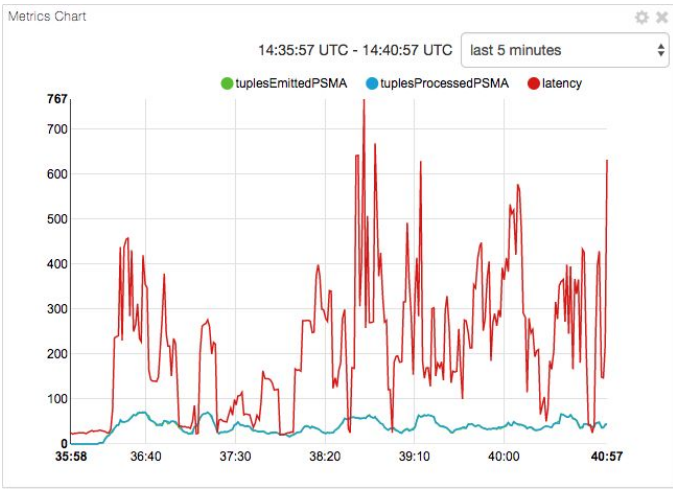


Fig. 5. Apex metrics of successful test run 4 running in the cloud with a "full-cache" with 1500 constant drivers.

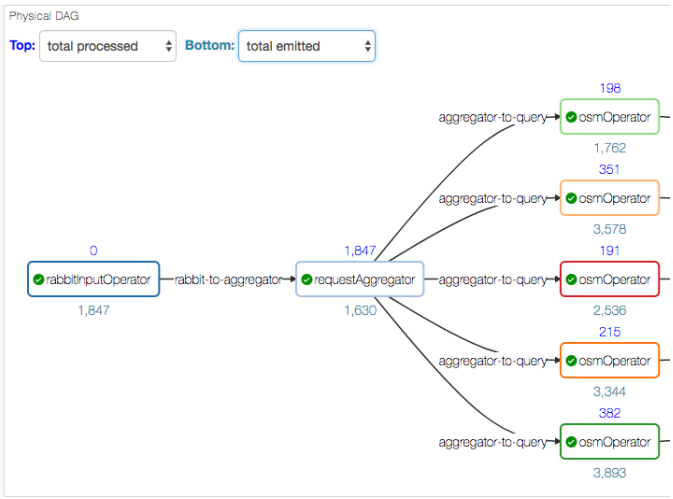
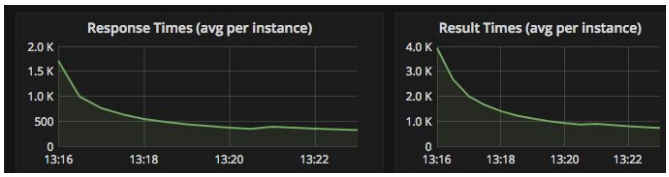


Fig. 6. Screenshot of the Apache Apex Operator Widget showing that the aggregation only reduces incoming requests by 12%, eventually causing the application to fail to respond to requests.

2) Performance with cloud-only model: **Identifying service bottleneck:** Table IV shows the increase of latency and the reduction of response- and result-rate, when more clients are added to the system. The response-rate and the result-rate drop due to the timeout of gRPC of the *Recommendation Service* whose single MongoDB instance can only handle up to 500 connection concurrently. Figure 5 shows that with caching, the streaming application at the *Detection Service* never receives more than 75 requests per second, resulting in very good latencies of only around 500 ms. With 2000 clients, we see the limits for the prototype. On average, only around 10% of the requests resulted in responses or results. The reason for the failing requests is that the single *Recommendation Service* reached its limits after around 5 minutes of execution. The gRPC-server could no longer handle requests.Test runs 7-9 show how the system performs when it is freshly deployed (empty-cache). While the response-times stay low and 99% of the requests are responded, the result times went up compared to running with full-cache. Since every request in the very beginning of the test run results in calling the detection, in the worst case, the Apex streaming application has to handle requests from every client at the same time. While the system still was able to handle up to 500 vehicles, at 1000 vehicles the detection service failed. The source of the failure is the `OSMOperator`. When many requests are incoming at the same time, the single OverpassAPI instance – a third party service – blocks the application flow.

Identifying application tasks bottleneck: Even though the `OSMOperator` was partitioned to 5 instances, still this did not resolve the bottleneck. The Apex application contains a RequestAggregator that tries to aggregate similar requests by time and location. As shown in Figure 6, in test run 9 this resulted in 1630 tuples that `OSMOperators` and OverpassAPI needed to handle. This caused extreme latencies, i.e. no tuples exited the pipeline and the *Recommendation Service* could not serve any further curves. To decrease the tuples at the Aggregator, in another test run, the TimeWindow was increased to 3 and the Geohash precision was set to 4

Run 10: 500 drivers, LTE network, "full-cache"



Run 11: 500 drivers, LTE network, "empty-cache"

Fig. 7. Comparison of result times when running 500 vehicles at LTE speed with full-cache vs. empty-cache

(causing a very large area to create aggregates). Using this configuration, at 2000 incoming tuples, the operator was able to aggregate the tuples by almost 87% to only 266. Having larger aggregates though also implies having larger results in terms of data. While the latencies of `OSMOperators` were reduced, the downstream operator to unify the results was not able to handle the amount of emitted ways and failed, causing the pipeline to be halted.

Understanding the role of cache: Test runs 7-9 showed the importance of caching. When the caches are empty, the system fails at running 500 to 1000 vehicles. Before deploying the system to a ready-to-use production, it should be considered to run the application in detection-mode first for a certain time. This can be achieved by simply providing geohash locations that are known to be frequently driven to the *Detection Service*. Another option would be to scale the third party `Overpass` to multiple high-compute instances. While this would resolve the bottleneck, if low costs are of importance, this should be only considered as a second option. With full-cache, the prototype showed very good results until test run 4, with vehicles receiving curves with an average delay of only around 1 second. This has been done in test runs 10 and 11, where a realistic scenario of 500 vehicles with LTE network was executed. As shown in Figure 7, the maximum result-time on full-cache, at the very beginning of the run, was at around 1.5 seconds. When the system was up for around 7 minutes on full-cache, this value dropped and on average vehicles received curves in 383ms. Compared to when all caches were empty, the maximum result-time was at around 4 seconds and after 7 minutes of run time dropped to an average of 726ms. When running 500 vehicles at LTE speed (full-cache), on average a vehicle receives curve results in less than 0.4 seconds. On a system start-up (empty-cache) this value almost doubles and a vehicle still receives curve responses in no more than 0.8 seconds.

Discussion: with the cloud we could utilize load balancing and caching to solve performance problems. However, if such services running in the edge, it might not be possible, because it is hard to implement load balancing and caching (on the other hand, edge systems can help reducing latency). For example, in MECCA, in case more instances of *Recommendation Service* would be available with load balancing enabled, this state would have been avoided. We see a complex dependencies among various factors, such as the design of parallel tasks, external services, etc. This requires complex design of the application and suitable deployment. For example, in MECCA, we have the bottleneck due to the third party service deployment – although we run parallel operators. one solution to fix this problem would be to also partition the downstream DetectCurves operator. When the OSMPartitions are set to 5, this would also lead to 5 additional containers for the DetectCurves operator. However, having 5 additional containers would increase the already very high memory usage even more. While on a cloud instance this could be easily achieved by buying a larger VM (but increase more costs), at the edge this is not yet a solution since computational resources are very limited. Here we also see another issue: it is not easy to use deployment to decide certain processing parts. The design of the part might be strongly associated with either cloud or edge resources. The question of interoperability in terms of execution and uploading is not straightforward like in other types of mobile-edge applications like [11]. Caching is very important and we can enable this well in case of cloud-only deployment model. However, it is much more challenging to support caching when we deploy components in both edge and cloud infrastructures.

C. Performance of edge-cloud deployment model

1) Testbed: As deploying applications to mobile edge servers is not yet possible, we emulated edge resources. For edge-cloud deployment, shown in Figure 8, we use different resources for different services being tested, depending on their capabilities. The resources run with OpenStack and the used virtual machines are configured in a way to be comparable to a *high compute* edge node that is currently used in the industry[3]. Table V specifies the VMs for running the simulated edge node and the cloud node. When summing up all resources of the VMs running in the simulated edge node, the CPU and memory specifications almost exactly match the one's specified by Cisco's *high compute* edge node. Differences are 1 additional CPU core, around 1.5 GB additional RAM and slightly more disk space of 20GB in total. To fit the specification of the simulated edge node, for the detection, an instance type of "m1.large" results in around 8GB memory, compared to 15GB in cloud-deployment tests.

Discussion: In this paper, based on performance in cloud-only deployment tests, we decide such an edge-cloud deployment but in principle there are many combinations, especially when we have quite different number of services. While we can emulate edge resources, it is important to note that the applications have to be reconfigured to suitable with edge resources. This requires the application components to be developed with this mindset. For example, to run the Apex application with the same partition size, as in the cloud-only deployment, in edge

[3]specified at https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/edge-fog-fabric/datasheet-c78-738866.html

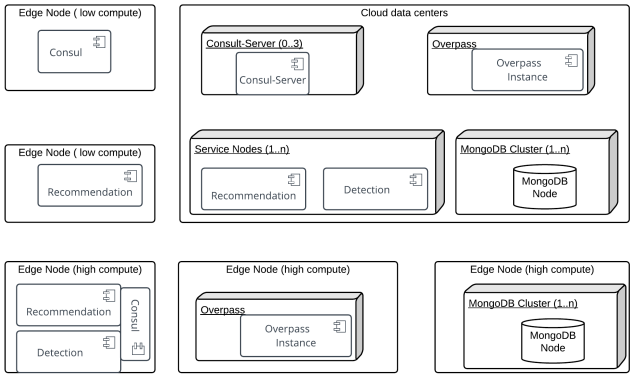


Fig. 8. Edge-cloud deployment possibilities

TABLE V
RESOURCES FOR EDGE CLOUD DEPLOYMENT

Name	Service	Type	Specs
OverpassAPI	GCP	n1-standard-8	default
Recommendation	DSG Cloud	m1.medium	CPU Cores: 2 Memory: 3.75GB DiskSpace: 40GB
Detection	DSG Cloud	m1.large	CPU Cores: 4 Memory: 7.68GB DiskSpace: 40GB
MongoDB	DSG Cloud	m1.small	CPU Cores: 1 Memory: 1.92GB DiskSpace: 40GB
Simulated Edge Node (total)	DSG Cloud		CPU Cores: 7 Memory: 13.5GB DiskSpace:120GB

nodes each operator memory was reduced from 512 MB to 256MB.

2) Evaluation of edge-cloud model: During the tests, the client deployed in a laptop was run inside the TU Wien library, hence the actual distance to the simulated edge node was around 500m. For each test run, the metrics *number-of-vehicles*, response-time rT, response-rate rR, result-time $resT$ and result-rate $resR$ will be analyzed. Test runs 1-4 executed on full-cache and 5-7 on empty-cache.

TABLE VI
TEST RESULTS OF EDGE-CLOUD DEPLOYMENT

Run	Vehicle	rT	rR	resT	resR	cache
1	100	395	99%	489	86%	full
2	500	493	84%	587	84%	full
3	1000	513	72%	584	84%	full
4	1500	951	88%	996	76%	full
5	2000	4551	8%	4612	7%	full
6	100	291	99%	1649	70%	empty
7	300	284	49%	1846	25%	empty
8	500	450	42%	/	/%	empty

Table VI shows the results of edge-cloud experiments. Until test run 5, the prototype running in the simulated edge node performed very well. Similar to cloud-only model, the maximum number of concurrent clients when running on full cache is 2000. The cloud-only deployment's response and result times are slightly better. The reason for this was the higher network bandwidth between the simulated vehicles and the servers. When running on empty cache, compared to the cloud prototype, the edge prototype already failed earlier, at 500 vehicles. The reason for this is the lower memory capacity for running the Apex application, causing `OSMOperators` to fail. This experiment shows that the prototype is able to handle between 1500 and 2000 vehicles on resource configurations that can be compared to an edge node currently used in the industry. Running the prototype on an empty-cache reduced this number to only 300 vehicles. This is due to the lower available memory for running Apache Apex to detect curves.

Discussion: Certain components will be the same in edge or cloud resources, e.g., in our case the *Recommendation Service* implemented in gRPC. However, to guarantee the benefit of edge computing, such components should be deployed in resources with enough elastic capabilities, unless the workload for them is fixed. This leads to two important aspects: certain components implemented with complex software, like our *Recommendation Service* with Apache Apex, (i) either are deployed in enough elastic edge resources, (ii) or should not be deployed in the edge, as it might cause resources contention and errors. It is important to establish a baseline of network and hardware similar to the state of the art of edge systems in order to understand performance. For example, the scenario showed that, with using current LTE network standards, the response times are almost as low as the one's measured between simulated vehicles and the cloud services. This, in principle, would not be expected in the future, real mobile edge cloud systems.

V. DATA QUALITY ANALYTICS

A. Metrics for data quality evaluation

TABLE VII
METRICS USED FOR DATA QUALITY EXPERIMENTS

Metric Name	Description
detection-rate	Percentage of successfully detected curves.
approaching-rate	Percentage of successfully classified curves as they approach while driving.
radius-error	Average error of calculated radius compared to measured radius.

Evaluating the quality of data is crucial for mobile edge cloud applications because the data delivery and collection are influenced by several factors. Table VII shows the metrics in our tests. Our simulation application to run the data quality tests reads in our predefined test-track and replays the GPS coordinates in sorted order. We tested performance using only one client at a time on the predefined track.

B. Testbed for data quality experiments

Compared to the performance experiments, hardware and software configurations for running the data-quality evaluation do not change and are the same for all following experiments, except the Overpass-Server is based on the public main server.

C. Analysis of quality of results

Our predefined test-track is simulated once from the start to the end. The monitoring client tracks the following metrics: measured radius mR, detected radius dR and radius error $rError$, shown in Table VIII. From the result, we received an

TABLE VIII
TEST RESULTS OF QUALITY OF RESULTS

ID	Detected	Approached	mR	dR	recSpeed	rError
1	true	true	286	341	78	55
4	false	false	286	/	/	/
5	true	true	245	262	68	17
7	true	true	296	459	91	163
9	true	false	352	703	112	352
11	true	true	226	260	68	34
13	true	true	246	267	69	22
15	true	true	150	162	54	12
17	true	true	229	242	66	14
19	true	true	294	360	80	66
21	true	true	250	256	68	7
23	true	true	197	191	58	6
25	true	true	161	190	58	30
27	true	true	212	292	72	80
28	true	true	107	531	97	424
29	true	false	176	439	88	263
31	true	true	108	246	66	138
33	false	false	169	/	/	/
35	true	true	220	253	67	33
39	false	false	71	/	/	/
40	true	true	93	234	65	141

average radius error of 103.06 meters. The overall detection-rate is 86% and the overall approaching-rate is 76%. A detection-rate and approaching-rate of around 80% indicates that the system performs very well when detecting curves. Having an average radius of over 100m though, in general, would suggest that the prototype cannot yet be classified as reliable when recommending speeds. To study the results in more detail, we found that the detection seems to have problems with the curves: 2,7,9,28 and 29. Curves 7 and 40 are overlapping with other curves. Overlapping curves are still very error-prone for the detection. Curve 9 is a very large curve (radius $>$ 300m). Curves with very large radius barely have angle differences between points. The curve detection algorithm does not perform well if angles are too low. Curve 29 has very few data points. If there are too few data points, the curve detection is very error-prone. Discarding these 4 problematic curves the average error of the radius drastically drops to 46.64 meters.

Discussion: When third party data is heterogeneous, the quality of results can be varying. Thus, optimization based on quality of results should consider the quality of input data.

D. Data quality due to sensing

For edge-cloud it is important to test data quality due to sensing. In the first run, inaccuracies are added to the GPS positions. This has been done by simply offsetting the GPS coordinates by a certain amount of meters. Based on existing evaluation by Zandbergen et al.[12], for this experiment we used the worst case scenario of 30 meters as upper bound for the maximum error. Thus, for each GPS coordinate, a random error between 0 and 30 meters was added. The other 3 runs simulate GPS outages at random locations. To simulate outages we simply skip tuples for a certain duration. The number of outages and their duration are configurable and were changed as follows: In test run 2 we defined 5 outages with each having a duration of 10 seconds. In the next test run 3 we increased the duration to 20 seconds. In the last test run 4 we defined 10 outages with a duration of 20 seconds.

TABLE IX
TEST RESULTS OF INACCURATE GPS

Run	Detection Rate	Approaching Rate
1	86%	67%
2	86%	76%
3	86%	76%
4	86%	67%

Table IX shows the test results when the GPS was inaccurate. When 5 outages occurred with a duration of 10 or even 20 seconds, both the detection-rate and the approaching-rate were not affected and both stayed at 86% and 76%. Only when the number of outages was doubled to 10, with each outage lasting 20 seconds, the approaching-rate slightly drops by 9% to 67%. The output measures for this specific test of course highly depend at what exact time the outages happen. Since the tests places the outages randomly (at certain intervals given by the number of outages), the results though show that again, the system is very fault-tolerant even to complete outages. Again this is achieved by using geohashes and pre-calculating curves.

Discussion: Connecting to a remote service at certain points and performing offline detection using the local cache makes the system highly fault-tolerant. For example, in general the detection is very fault-tolerant to inaccuracies and outages, because we pre-calculate curves for an area around the current location using geohashes. If vehicles would connect to a remote service to fetch curves on every GPS update, the rates would drop significantly in case of GPS errors.

E. Data quality due to network problems

TABLE X
NETWORK PROFILES WITH APPLE'S NETWORK LINK CONDITIONER [10]

Profile	Bandwidth	Delay(ms)	Packets Dropped(%)
LTE	50mbps%	50	0
Edge(2G)	240kbps%	400	0

We used two profiles in Table X to specify the downlink properties. Since the uplink values are almost the same for every profile, they are left out here. For all previous data quality experiments, LTE network conditions were used. In the first test run of this experiment, we simulate a very slow network connection using Edge (2G). In all previous test runs, both the recommendation and the central database made use of caching and already had curves in their caches. To simulate a worst-case scenario, additionally to using only Edge (2G) network, we also clear all caches. The monitoring client tracks the metrics *average-result-time* and *average-response-time*. Table XI shows the test results. If the network is slow, both detection rates and approaching rates stayed unaffected at 86% and 76%. Figure 9 shows the average response and result times when executing different runs of the experiments.

Discussion: A right pre-calculating by running services, e.g. determining curves for areas around the current location (using geohashing) in MECCA, makes the system highly tolerant against slow and even very slow network conditions. However,

TABLE XI
TEST RESULTS OF DATA QUALITY DUE TO NETWORK PROBLEMS

Run	Detection Rate	Approaching Rate	rT (ms)	resT (ms)
1	86%	76%	1127	1127
2	86%	76%	1976	5891

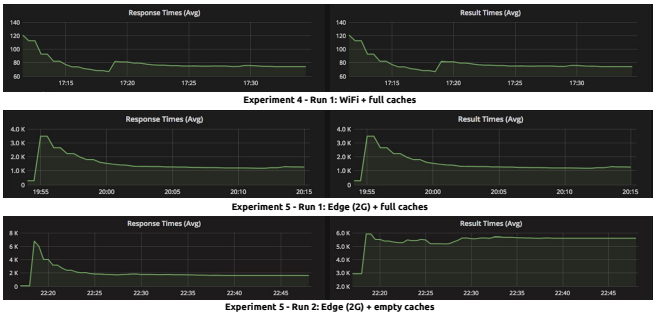


Fig. 9. Screenshot of the Prometheus monitor showing average response-and result times of selected test runs during data quality evaluation.

this means that the client should be proactive to monitor the network and to perform pre-calculating. Furthermore, this can also create wasted calculation if many tasks (e.g., curves) need to be done but unused. A tradeoff must be made for that.

VI. Related Work

Cloud benchmarking is a very popular topic with different levels of benchmarks, such as virtual machines, message brokers and database [13], [14]. However, mobile edge computing testing and benchmarking is under-researched, although edge cloud systems and applications currently are in the focus of various researchers. The work in [15] represents discussions w.r.t architecture for edge cloud systems using containers and clusters. The work in [16] surveys techniques for monitoring edge applications. Various middleware have been proposed, such as in [17]. Our focus in this paper differs as we address performance and quality of data testing.

In [18], the authors benchmarked Hadoop in small cloud configurations. While such configurations can mimic edge configuration, the work has not tested edge cloud applications in realistic edge cloud infrastructures. Many papers have focused on techniques for optimizing offloading between edge and cloud [1], [2], utilizing mobile devices for edge services [3], migrating services [4]. Our work is different as we are not focus on offloading but understanding performance and data quality impacts based on different configurations and application models. Our future work can focus on testing mobile-edge cloud where such above-mentioned techniques are also employed for services and resources.

VII. Conclusions and Future Work

In this paper, we studied various performance and data quality factors for mobile edge cloud applications. We selected MECCA as a complex application for our study. As we presented, performance and data quality depend very much on the deployment model but also the design of software. We have presented several discussions to recommend possible techniques for analytics of mobile edge cloud applications.

While our approach does not present a generic tool for performance analysis, it does bring several experiences that we could consider in order to optimize edge cloud applications and to build toolset for performance and data quality. Our future work is to focus on building a toolset for analytics of edge cloud applications that can correlate data from various layers and can be used to setup tests for such applications.

References

[1] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, October 2016.

[2] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, thirdquarter 2017.

[3] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femto clouds: Leveraging mobile devices to provide cloud service at the edge," in *2015 IEEE 8th International Conference on Cloud Computing*, June 2015, pp. 9–16.

[4] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato, "Hybrid method for minimizing service delay in edge cloud computing through vm migration and transmission power control," *IEEE Trans. Comput.*, vol. 66, no. 5, pp. 810–819, May 2017.

[5] S. Schmidl, "Untersuchung des fahrverhaltens in unterschiedlichen kurvenradien bei trockener fahrbahn," Master's thesis, Universität für Bodenkultur Wien, Mar 2011.

[6] P. OSRM, "Osrm api documentation," http://project-osrm.org/docs/v5.10.0/api/#general-options, (Accessed on 02/12/2017).

[7] O. Foundation, "Overpass api," http://wiki.openstreetmap.org/wiki/Overpass_API, 2017, (Accessed on 05/03/2017).

[8] H. L. Truong and S. Dustdar, "Principles for engineering iot cloud systems," *IEEE Cloud Computing*, vol. 2, no. 2, pp. 68–76, 2015.

[9] M. M. Lopes, W. A. Higashino, M. A. Capretz, and L. F. Bittencourt, "Myifogsim: A simulator for virtual machine migration in fog computing," in *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*, ser. UCC '17 Companion. New York, NY, USA: ACM, 2017, pp. 47–52.

[10] NSHipster, "Network link conditioner," http://nshipster.com/network-link-conditioner/, (Accessed on 02/12/2017).

[11] M. Chen, Y. Hao, Y. Li, C.-F. Lai, and D. Wu, "On the computation offloading at ad hoc cloudlet: architecture and service modes," *IEEE Communications Magazine*, vol. 53, no. 6, pp. 18–24, 2015.

[12] P. A. Zandbergen, "Accuracy of iphone locations: A comparison of assisted gps, wifi and cellular positioning," *Transactions in GIS*, vol. 13, pp. 5–25, 2009.

[13] R. Lu, G. Wu, B. Xie, and J. Hu, "Stream bench: Towards benchmarking modern distributed stream computing frameworks," in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, Dec 2014, pp. 69–78.

[14] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 143–154. [Online]. Available: http://doi.acm.org/10.1145/1807128.1807152

[15] C. Pahl and B. Lee, "Containers and clusters for edge cloud architectures – a technology review," in *Proceedings of the 2015 3rd International Conference on Future Internet of Things and Cloud*, ser. FICLOUD '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 379–386.

[16] S. Taherizadeh, A. C. Jones, I. Taylor, Z. Zhao, and V. Stankovski, "Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review," *Journal of Systems and Software*, vol. 136, pp. 19 – 38, 2018.

[17] J. a. Rodrigues, E. R. B. Marques, L. M. B. Lopes, and F. Silva, "Towards a middleware for mobile edge-cloud applications," in *Proceedings of the 2Nd Workshop on Middleware for Edge Clouds & Cloudlets*, ser. MECC '17. New York, NY, USA: ACM, 2017, pp. 1:1–1:6.

[18] M. Femminella, M. Pergolesi, and G. Reali, "Performance evaluation of edge cloud computing system for big data applications," in *2016 5th IEEE International Conference on Cloud Networking (Cloudnet)*, Oct 2016, pp. 170–175.