SparkEdgeEmu: An Emulation Framework for Edge-Enabled Apache Spark Deployments

Chapter · August 2023

DOI: 10.1007/978-3-031-39698-4_11

4 authors:

Moysis Symeonidis
University of Cyprus
19 PUBLICATIONS **151** CITATIONS
SEE PROFILE

Demetris Trihinas
University of Cyprus
41 PUBLICATIONS **500** CITATIONS
SEE PROFILE

George Pallis
University of Cyprus
125 PUBLICATIONS **3,802** CITATIONS
SEE PROFILE

Marios D. Dikaiakos
University of Cyprus
235 PUBLICATIONS **3,890** CITATIONS
SEE PROFILE

SparkEdgeEmu: An Emulation Framework for Edge-enabled Apache Spark Deployments

Moysis Symeonides[1], Demetris Trihinas[2], George Pallis[1], and Marios D. Dikaiakos[1]

[1] Department of Computer Science, University of Cyprus
{msymeo03, pallis, mdd}@ucy.ac.cy
[2] Department of Computer Science, University of Nicosia
trihinas.d@unic.ac.cy

Abstract. Edge Computing emerges as a stable and efficient solution for IoT data processing and analytics. With big data distributed engines to be deployed on edge infrastructures, users seek solutions to evaluate the performance of their analytics queries. In this paper, we introduce SparkEdgeEmu, an interactive framework designed for researchers and practitioners who need to inspect the performance of Spark analytic jobs without the edge topology setup burden. SparkEdgeEmu provides: (i) parameterizable template-based use cases for edge infrastructures, (ii) real-time emulated environments serving ready-to-use Spark clusters, (iii) a unified and interactive programming interface for the framework's execution and query submission, and (vi) utilization metrics from the underlying emulated topology as well as performance and quantitative metrics from the deployed queries. We evaluate the usability of our framework in a smart city use case and extract useful performance hints for the Apache Spark code execution.

Keywords: Edge Computing · Internet of Things · Big Data.

1 Introduction

The proliferation of the Internet of Things (IoT) has led to an explosion in installations of IoT devices and in the amount of IoT-generated data. Recent reports estimate that by 2025 IoT will comprise around 41 billion devices in operation worldwide, producing on a daily basis about 80ZB of data [9]. However, typical IoT devices do not have adequate storage capacity and processing power to perform analytics tasks. Thus, application designers and operators recognized the need to offload IoT data into more powerful computing platforms placed at the proximity of IoT data sources, to cope with processing, storage, and latency requirements of "earthbound" applications. This has led to the emergence of the Edge Computing paradigm, which offers in-place processing, data transfer minimization, and on-time result calculation by deploying analytic tasks on edge-driven compute nodes. A typical edge node, however, does not have the capacity to host and run demanding analytics jobs that arise in many application use cases. Therefore, big data engines, like Apache Spark, provide implementations, which distribute computation and data to multiple edge nodes, and take advantage of their aggregate capacity. These implementations hide the complexity arising from machine communication, task scheduling, fault tolerant operation, etc., behind higher-level abstractions for performing queries on IoT data [20]. However, predicting the resource needs and the performance behavior of analytic queries running on edge nodes that are deployed on a wide geographic area, exposed to possibly sub-optimal environmental conditions with limited computing resources and often unstable network connectivity, is a challenging endeavor [17, 19]. It is expensive and time-consuming to develop, configure, test, and reproduce the conditions of large-scale, physical testbeds; consequently, testing and performance evaluation become major barriers for edge processing.

To alleviate the difficulties of a physical testbed, users attempt to evaluate the performance of their tasks via emulation frameworks [11, 3], which mimic the conditions and effects of a physical deployment on the deployed services. Even if emulators achieve near-to-real conditions for the deployed services, users have to describe, containerize, and configure these services. Furthermore, emulation frameworks usually provide modeling toolkits that users require to define every emulated node and its properties manually, including processing capabilities, network configurations, deployed services, etc. Then, users need to evaluate the performance of the submitted exploratory queries on scattered datasets by extracting quantitative and utilization metrics from the deployed big data distributed engine and the underlying infrastructure [13]. Considering that the majority of analytic platform users are data scientists, they are not aware of distributed engine deployment, configuration, and monitoring.

To address these challenges, we introduce SparkEdgeEmu, a framework for interactive performance and bottleneck analysis of Apache Spark jobs deployed over emulated edge testbeds. Users only need to fulfill use case templates, leaving the framework to bootstrap the emulated testbed, deploy Spark services, inject the respective datasets, and capture monitoring metrics for post-execution performance analysis. The main contributions of this work are: (i) the *Modeling Abstractions* for parameterizable templates of scalable edge topologies, through which users select the respective use case and its parameters, as well as the topology's compute and network resources, (ii) an *Open-source Implementation*[3] of the SparkEdgeEmu that translates the use case model to a large-scale emulated Apache Spark testbed, providing multi-host scalability, inherited from its underlying emulator [19], query- and topology-level metrics for post-experiment performance evaluation and bottleneck identification analysis, which consequently helps in the query optimization process, and (iii) an *Experimental Study* of analytic queries executed on a city-scale testbed that uncovers hidden insights of the queries performance and the Apache Spark footprint.

The rest of the paper is structured as follows: Section 2 describes the related work. Sections 3 and 4 show the framework and its implementation details, respectively. Section 5 presents the experiments and Section 6 concludes the paper.

[3] https://www.github.com/UCY-LINC-LAB/SparkEdgeEmu

2 Related Work

To evaluate the performance of a system, users typically embrace benchmarking suites that provide both workloads and monitoring capabilities. There is a plethora of benchmarking tools related to big data analytics. For instance, Yahoo YCSB [7] provides a range of diverse tools for DBMS benchmarking. Moreover, SparkBench [15] introduces ML, graph computation, SQL queries, and streaming workloads for Apache Spark. A study of latency and throughput between big data streaming engines is presented in [6], while Karimov et al. [13] provide a novel definition for the latency of stateful operators, a method to measure it and decouple the underlying systems from the driver that controls the experimentation to have fair results. An edge-oriented and automated approach is proposed in [10]. The authors introduce BenchPilot, a framework that is capable of performing repeatable and reproducible experiments on Edge micro-DCs. Even if the benchmarking studies alleviate the difficulties of analytic workload creation and experimentation, *they consider an already deployed edge infrastructure, which is sometimes unrealistic during the design phase.*

To create realistic testing conditions without facing the cost and the configuration effort of a real edge cluster, users turn to emulation frameworks. Frameworks like FogBed [8] and EmuEdge [22] notably expand network emulators (e.g., MiniNet [14]) to provide fog/edge resource and network heterogeneity. Interestingly, Beilharz et al. introduce Marvis [3], a hybrid testbed that combines simulated events with emulated infrastructure for evaluating distributed IoT applications. The system integrates the ns-3 network simulator with virtualization technologies and domain-specific simulators (e.g., traffic simulators). However, these solutions inherit the restrictions of the network emulators like strict modeling (i.e., the configuration of routers, gateways, IP masks) and limited scalability. To tackle these issues, a series of emulation frameworks introduce distributed cloud technologies via multi-host overlay networks and virtualization technologies. For example, MockFog [11] is a fog emulator that is deployable on AWS and OpenStack clusters, provides the required heterogeneity, and enables users to inject network faults at run-time. Moreover, other container-based Fog and 5G emulation frameworks [16, 19, 21] offer multi-host scalability, realistic emulation via ad-hoc topology updates, automated service deployment, and emulation monitoring. However, *none of the above solutions are focused on Spark analytic queries, leaving users to handle the barrier of containerization, configuration, deployment, and monitoring of the distributed processing engines.*

3 System Overview

The time-consuming infrastructure setup required for studying analytic queries' performance on the edge increases product time-to-market and turns analysts' attention away from the actual purpose of query performance evaluation. To ease understanding, let us consider a use case where a data scientist wants to evaluate the performance of his/her analytic queries on a realistic smart city

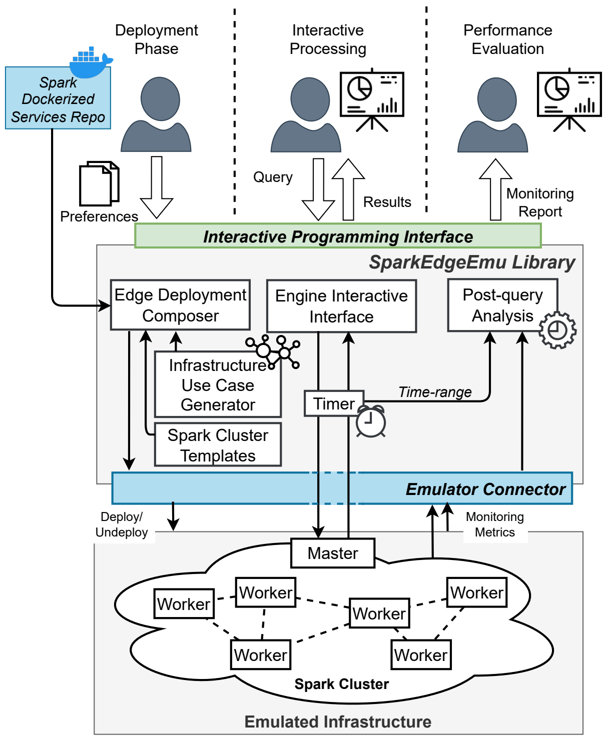

Fig. 1: System Overview

edge deployment. In particular, the data scientist needs an installed city-scale edge infrastructure along with a deployed big data engine to submit his/her queries and monitor their performance. However, operators are almost impossible to provide a ready-to-use infrastructure from the beginning of a project. Thus, users can only evaluate the performance of the analytic queries in a local virtualized environment or a rented Cloud cluster. This results in an error-prone performance that may over- or under-estimate the edge capabilities.

Contrary to this approach, users can embrace the SparkEdgeEmu Framework. The high-level overview of the framework is depicted in Figure 1. The on-boarding starts with the selection of a predefined use case via the framework's *modeling abstractions*, along with the definition of parameters like the number and the density of edge nodes, processing, and network QoS, which are based on statistical distributions extracted from real-world deployments [1, 4, 5, 18]. For instance, for a smart city use case, users select the number of neighborhoods, the number of compute devices per neighborhood, cloud servers, and their capabilities and connectivity characteristics. Then, users submit the parameterized model to the platform via the *Interactive Programming Interface*.

With the parameters at hand, the framework uses the *Edge Deployment Composer (EDC)* to construct an in-memory data structure that keeps a set of deployable elements, with each element being a materialized view of the user's preferences. To do that, the EDC invokes the *Infrastructure Use Case Generator* and provides the infrastructure parameters to it. The generator transforms the parameters into statistically generated edge topologies and returns them to the EDC. Then, EDC retrieves the *Spark Cluster Templates* and fills them with service-level parameters, such as network addresses and node resources. With both templates and infrastructure descriptions ready, EDC enriches the topologies with the Apache Spark services, IoT datasets, and placement preferences. The output of EDC is a statistically generated ready-to-use deployment description. The system propagates the description to the *Emulator Connector*, which instantiates the emulated infrastructure, deploys the services, and retrieves both the emulated infrastructure metrics and metrics from the deployed Spark cluster. In this paper, we opted not to utilize a distributed storage system such as HDFS for distributing the IoT datasets. Instead, we introduce a shareable folder on each node where the framework stores the corresponding IoT data files[4]. Regarding Apache Spark services, the framework offers an online repository that contains docker images which include the required executable artifacts and binaries.

When the emulation is ready and the Spark cluster deployed, the users can submit analytical tasks through the framework's *Interactive Programming Interface*. Specifically, users execute the analytical tasks as code blocks via the programming abstractions, and the system records the starting and ending timestamps of the respective code block, which may perform multiple sequential analytic queries. When they are finished, users are aware of the duration of the queries and retrieve the captured metrics via the Post-query Analysis module.

The *Post-query Analysis* module requests the underlying infrastructure utilization metrics and big data engine statistics from the Emulator Connector, filtered by the code block's starting and ending timestamps. Finally, users may perform high-level analysis on the retrieved metrics generating a more clear overview of the submitted queries' performance.

4 Implementation Aspects

This section presents the framework's implementation aspects for SparkEdgeEmu key components.

Modeling Abstractions. There are several emulators that provide high-level modeling abstractions [19, 16]. However, their users need to describe every single compute node and its network connections, which makes the design of a large-scale deployment challenging. To bridge the gap between scalability and expressivity, we introduce high-level template-based infrastructure and use case modeling abstractions. Model 1.1 depicts an example of system's modeling. Specifically, the users introduce the types of the devices (`devices_types`) and connections (`connection_types`). For the devices, users have to specify the `name` of the device, the `processor`'s capabilities (including `cores` and `clock_speed`),

[4] Possible issues regarding storage, like security concerns, are out of our scope. However, we plan to introduce an emulated distributed storage as a future extension.

the device's `memory`, and the `disk` capabilities (e.g., `technology`, `size`, `read` & `write` throughput, etc.). Moreover, a connection type has an identifier (`name`), and `uplink` & `downlink` QoS that include `data_rate`, `latency`, and `error_rate`.

```
infrastructure:
   devices_types:
   - name: small-vm
     processor:
       cores: 4
       clock_speed: 1.5GHz
     memory: 4GB
     disk:
       technology: SSD
       size: 32GB
       read: 95MB/s
       write: 90MB/s
     ....
   connection_types:
   - name: 5G
     downlink:
       data_rate: 90MBps
       latency: 2ms
       error_rate: 0.1%
     uplink: ....
usecase:
   usecase_type: smart_city
   parameters:
       num_of_regions: 3
       num_of_devices_per_region: 7
       edge_devices: [rpi3b, nuc]
       edge_connection: 5G
       cloudlets: [small-vm]
```

Model 1.1: Infrastructure & Use Case Parameters

We note that SparkEdgeEmu provides out-of-the-box profiles for popular edge devices, e.g., raspberries (`rpi3b`), and connections, e.g., `4G`, `5G`, and `wifi` standards, which users use without having to define them again. Finally, the `usecase` primitive materializes a randomized edge deployment. Specifically, use case includes a `usecase_type` that defines the selected template and a set of parameters, through which users configure the template. For instance, in Model 1.1, the use case refers to a `smart_city` template and users set its parameters, such as number of regions, devices per region and their types, the network type, etc.

Interactive Programming Interface. Next, SparkEdgeEmu users start the experimentation utilizing a Python-based programming interface (e.g., Code 1.1). The users execute the SparkEdgeEmu functions locally, while the framework handles communication with the underlying emulator and the emulated Apache Spark cluster. For the emulation, users submit the described use case creating a connector object (lines 1-4), and deploy the use case to the underlying emulation framework (line 5). In addition, our choice of Python enables us to leverage PySpark, a Python library that facilitates connectivity to an Apache Spark cluster. By instantiating an Apache Spark session object, users can submit their Spark code to the cluster simply by specifying the IP address of the master. Within the SparkEdgeEmu programming interface, we have implemented the "*create_spark_session*" function, which generates a session object with a pre-configured emulated Spark Master IP (lines 7-9). Through this function, users identify the Spark connection's configurations (e.g., executors' processing capabilities) and get an Apache Spark session object, which interactively communicates with the underlying emulated Apache Spark cluster. When users execute queries in code-blocks under the "*with connector.timer()*" statement (line 10), the system captures the duration of their execution. In this way, the system keeps the starting and ending point of the code block (lines 10-14) and can retrieve metrics from this period (line 15). The metrics include statistics from the deployed Apache Spark queries, e.g., the number of tasks (line 16) and metrics from the emulated infrastructure, e.g., CPU utilization (line 17).

```
1  connector = EmulatorConnector(
2                  controller_ip = '...',
3                  usecase = 'usecase.yaml'
4              )
5  connector.deploy()
6
7  spark_session = connector.create_spark_session(
8          app_name = 'app-1', configs = { ... } )
9
10 with connector.timer():
11     df = spark_session.read.csv('data.csv')
12     df.groupBy('DOLocationID') \
13       .agg({'driver_pay':'avg'}).collect()
14     ....
15 monitoring_data = connector.get_metrics()
16 monitoring_data['rpi3_b_0'].tasks.plot()
17 monitoring_data['rpi3_b_0'].cpu_util.plot()
```

Code 1.1: Programming Interaction Primitives

Infrastructure Use Case Generator. To materialize the infrastructure generator, we adopt and extend Ether [18], which is a framework for synthesizing plausible edge infrastructure configurations from a set of reference use cases, which are grounded on empirical data, including smart cities [4], Industrial IoT deployment [5], mobile edge clouds and vehicular networks [1]. Developers can utilize Ether's programming primitives and building blocks to create reusable edge infrastructure configurations and topologies. In our case, the Infrastructure Generator translates the use case modeling abstractions into Ether's programming primitives, and Ether creates an in-memory graph keeping all required information for both networks and compute nodes. However, Ether does not have all network or compute properties that our system needs. For example, Ether defines processing power as CPU cycles without considering the number of cores. For the latter, we extend Ether's node abstraction to encapsulate also CPU's number of cores and clock frequency. Moreover, an Ether-enabled use case is framed in a geospatial context through which users evaluate the effects of geographically distributed edge systems. In such a setup, the nodes are distributed across a region by following a specific distribution, e.g., uniform or log-normal. Ether generates the respective connectivity for the nodes and produces

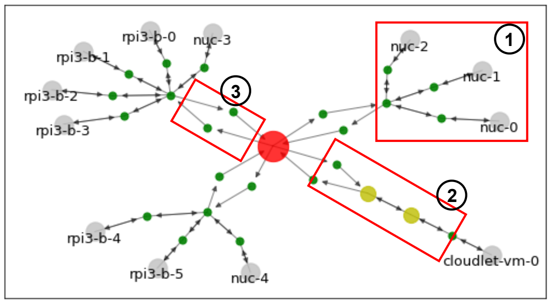

Fig. 2: Ether's Visualization for Smart City Use Case

the underlying network fabric. We upgrade the latter functionality by introducing realistic wireless signal quality models for 5G MIMO channels [21]. Figure 2 shows a representation of an auto-generated smart city use case with three neighborhoods and one cloudlet. The placement of nodes in a neighborhood follows lognormal distribution, as is highlighted in [2, 18], thus, each neighborhood has a different number of nodes. For instance, rectangle ① depicts a neighborhood with three Intel's Next Unit of Computing (NUC) nodes [12], while others have more nodes or include RPi3s. The yellow ② and green circles ③ depict network components, like switches, and uplink/downlink connections, respectively.

Emulation & Deployment. With the edge topology created, the *Edge Topology Composer* is responsible for the creation of the underlying emulation framework model. Specifically, the system fills the Spark templates with proper parameters to generate the Spark services and place them on the auto-generated topology. There are two types of templates for a Spark Cluster, one for the master node and one for the workers' nodes. In these templates, the Edge Topology Composer provides properties, like topology nodes' network addresses, hostnames, other Spark parameters, etc. When the templates are ready, Edge Topology Composer explores the in-memory graph of the infrastructure generator, utilizing Ether's exploration methods for graph and node-to-node analytics like, node's properties identification, uplink and downlink network QoS, link capacity, etc. Thus, the composer keeps the compute nodes' capabilities, identifies the network QoS links among the edge nodes, and forms the underlying emulation model by utilizing the Emulator Connector.

In our prototype, we create a connector for the Fogify emulation framework [19]. Fogify provides a programmable way to produce its model, multi-host scalability, and a less than 10% performance difference between emulation and physical infrastructure. Fogify's connector creates the emulation model and submits it through Fogify's API. Fogify validates the submitted description and translates the model specification to a running multi-host emulated environment. The framework utilizes container-based cluster orchestrators (e.g., docker swarm) to ensure the instantiation, and constraining of the services on the containerized emulated environment. Moreover, Fogify Agents, which are internal services of the framework, apply the respective network QoS and monitor the

emulated instances. To this end, the output is a Spark cluster deployed on the emulated edge infrastructure that awaits for incoming user's queries.
Monitoring Metrics. By invoking the emulation connector, SparkEdgeEmu retrieves monitoring metrics from the underlying emulation topology after the execution of a batch of analytics queries. Specifically, the Fogify emulator offers a wide range of infrastructure utilization metrics, including CPU utilization, memory usage, network traffic, disk I/Os, etc. However, users do not only require metrics from the underlying infrastructure but also metrics and statistics from the running big data engine, e.g., the average execution time per task or the number of assigned tasks for a specific cluster node. Moreover, holistic metrics, like the overall execution latency or the overall consumed resources, are also important for the performance evaluation of analytics queries. For that reason, we extended Fogify's monitoring system to store metrics from a running deployed Apache Spark cluster. Specifically, Fogify's monitoring subsystem periodically polls the internal monitoring API of the deployed Apache Spark cluster and saves the retrieved measurements. The spark-related metrics refer to each worker and include (i) assigned, completed, and failed tasks, (ii) JVM memory, (iii) cached data size, (vi) CPU time, (v) per-task duration, and so on. Finally, SparkEdgeEmu offers methods for exposing these metrics in a unified manner through which users can combine, process, and analyze them.

5 Experimental Study

Next, we examine the use case of smart city deployment on Edge computing topology and analytic queries.
Topology, Workload and IoT Data. For the topology generation, we utilize the model of the smart city use case as introduced at Model 1.1 and the exemplary code snippet of Code 1.1. For the parameters of the use case, we set the number of neighborhoods equals to 3, the average number of edge devices in each neighborhood to 7, including Pi4 (4GB), Pi3b+ raspberries, and NVIDIA Jetson Nanos. Except for the edge devices, we also introduce a cloudlet server with 8 CPUs@2.4GHz and 8GB memory. The generated topology includes 22 edge nodes and one cloudlet, with the first neighborhood having 4xRPi4, 2xRPi3b, and 1xJetson-Nano, the second neighborhood having 1xRPi4, 4xRPi3b, and 3xJetson-Nano, and, finally, the third neighborhood including 1xRPi4, 2xRPi3b, and 4xJetson-Nano. As a representative dataset, we utilize a publically available and real-world dataset comprised of For-Hire Vehicle ("FHV") trip routes in the first half of 2022 from New York city [5]. Each vehicle is equipped with an IoT tracking device to record 24 metrics for each route, including charged amount, tip amount, pickup/dropoff location, etc. We set the dataset to be stored and distributed over the edge nodes, and we submit three analytic queries on it with their descriptions to be on Table 1. All queries include multiple stages with the first stage digesting the input IoT data parquet [6] files. Each trial is repeated 10 times with final results depicting the overall measurements of all runs. All

[5] https://goo.gl/X9rCpq [6] https://parquet.apache.org/

Query	Description
`Q1`	The average of payment grouped by the drop-off location
`Q2`	The number of trips (count) per company
`Q3`	The overall amount of tips that passengers provided

Table 1: Submitted Queries

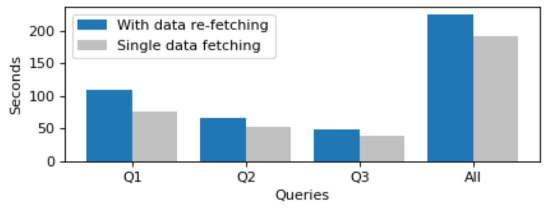

Fig. 3: Queries Execution Duration

experiments are conducted with SparkEdgeEmu to be run on a server with 48cores@2.450GHz and 176GB memory.

5.1 Experiments & Results

Code-block Performance Evaluation Differences. Firstly, we evaluated the performance of the queries (Table 1) deployed on the emulated topology. We examine each separate query but also all queries together as a code-block. Furthermore, we examine also how the data fetching influences the performance of the deployed queries. To evaluate the latter, we re-fetch the data at the beginning of each code-block execution, while to avoid the re-fetching, we retrieve the data once and keep them in memory. Figure 3 illustrates the average execution time of ten runs of each configuration. We observe that the execution time follows the same order (`Q1`>`Q2`>`Q3`) independently of the data-fetching approach. If we evaluate the semantics of the queries, we easily recognize that the `Q1` is a group-by query that performs average, while `Q2` is again a group-by query but only counts the data points. Intuitively, the averaging of a batch of data is heavier than a simple count. Furthermore, the group-by is performed in a different field, with the cardinality of drop-off locations being much higher than the number of car-sharing companies. For similar reasons, it is reasonable `Q3` to be the most light-weight query in execution. According to the data fetching, it influences the execution time in all experiments with an average overhead of about 25%. The latter indicates that Apache Spark does not identify possible optimizations in data-fetching during the repeatable executions and users should be aware of that. In conclusion, *SparkEdgeEmu Framework helps in performance analysis and performance bottleneck identification. During the experimentation, we highlight that the performance of group-by queries is characterized by the number of key elements and aggregation function, while Apache Spark seems to be unaware of data re-fetching and re-computations.*

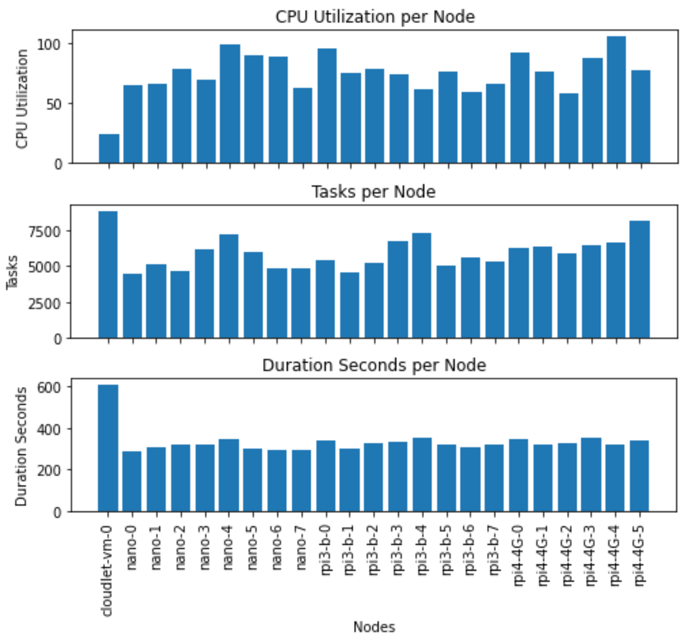

Fig. 4: CPU-related Metrics of the Experiment

For the rest of the experiments, we use metrics captured from the execution of all queries without data re-fetching.

CPU & Analytic Tasks. SparkEdgeEmu helps users to identify also the workload placement and nodes' utilization of the underlying cluster through its wide range of monitoring metrics. For instance, Figure 4 depicts three bar charts from workload-related metrics, namely, the emulated node CPU utilization, the assigning tasks of Apache spark, and the cumulative duration in seconds that Apache Spark considers. Interestingly, we observe that the cloudlet VM was underutilized during the experimentation even if Apache Spark was assigned to it for most of the tasks. Moreover, Spark measured that cloudlet workers spent much more CPU time (Duration Seconds) than edge devices. As benchmarking efforts have already identified [10], *distributed processing big data engines tend to assign more tasks to more powerful nodes, while these nodes usually are underutilized in Edge topologies.*

Memory Consumption. Another metric that influences the performance of the Apache Spark cluster is the consumed memory of the cluster's nodes. Figure 5 illustrates the consumed memory in bytes (x10^8) and the utilization percentage for every emulated node. We have to note here that we keep the default Apache Spark parameters in all experiments, so the default memory that an engine's worker can utilize is 1GB. One can clearly identify that all nodes have about 700-750MB occupied memory except for raspberries 3b which have about 400-500MB. Since RPi3b has only 1GB of memory, the average percentage of

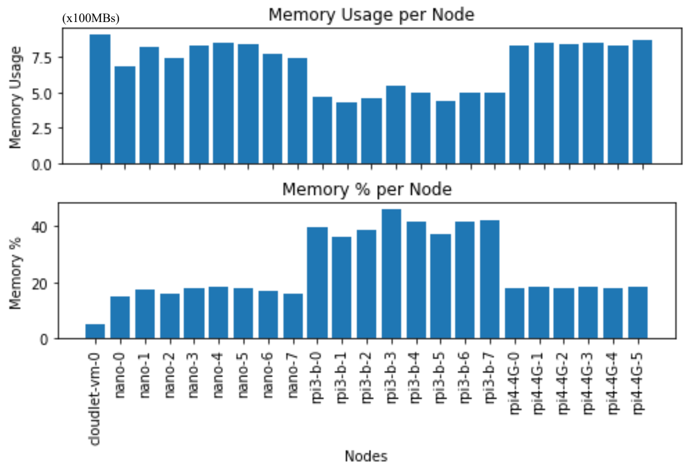

Fig. 5: Memory-related metrics of the Experiment

occupied memory is 40% for this device. In summary, *Apache Spark occupied less memory on edge memory-constrained devices, even if it assigns to them a similar number of tasks as the other Edge nodes (Fig. 4).*

Network Traffic & Shuffling Data. Figure 6 depicts network-related data extracted from both emulated infrastructure and Apache Spark cluster. Specifically, the first plot highlights the network traffic (both incoming and outgoing) in bytes (x10^7), while the second and the third plot illustrate the bytes generated from Spark's Shuffling Read and Write, respectively. Apache Spark generates shuffling data among different stages (usually when join or group operator is performed). So, the *Shuffle Write* metric illustrates how many bytes are generated from a local stage and should be transferred to another operator, while *Shuffle Read* metric is how many bytes are consumed by the current worker. An interesting observation is that the size of the network traffic captured by the emulator is higher than the traffic between the stages captured by the Apache Spark engine. The extra traffic among the cluster nodes could be health-check and the task-assigning messages that the engine uses to keep the cluster and processing alive. To this end, *the health-check and the task-assigning messages contribute a non-negligible extra overhead in network traffic.*

6 Conclusion & Future Work

In this paper, we introduce SparkEdgeEmu, an interactive framework that facilitates in performance evaluation and assessment of analytic processing on Edge-enabled Apache Spark clusters. It provides a unified interface through which data analysts can: (i) create auto-generated scenario-based Edge topologies, (ii) materialize the topologies into a real-time emulation with a deployed Apache Spark cluster, (iii) submit analytic tasks and observe its results, (iv) inspect and moni-

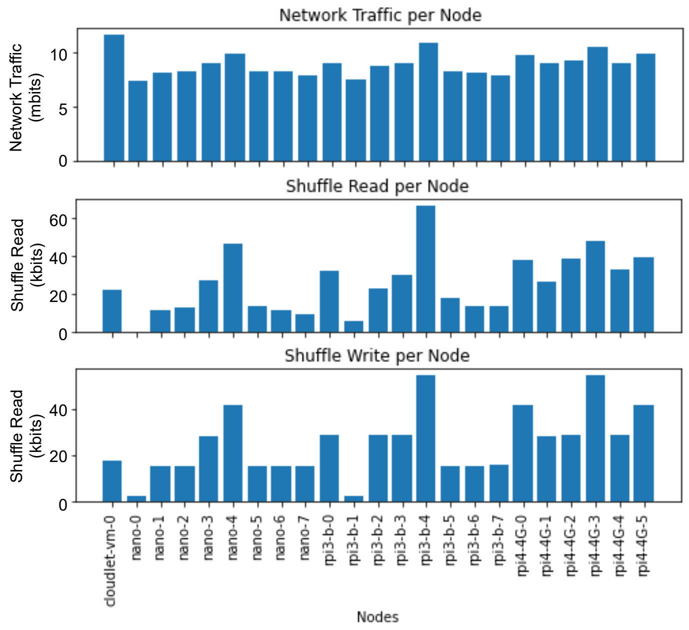

Fig. 6: Network-related metrics of the Experiment

tor the execution of the map-reduce tasks, and (v) perform post-experimentation analysis on the captured measurements. Furthermore, we provided implementation details about the framework's programming abstractions, infrastructure generation, underlying emulation, and monitoring metrics extraction. Finally, we evaluated the useability of our approach via a representative city-scale use case and performed a wide analysis of IoT data and deployment performance.
Future Work. We plan to add more underlying emulators and test their accuracy by comparing them to real-world deployments. To evaluate them properly, we'll deploy edge devices in different locations, install Apache Spark on them, and collect utilization and performance metrics. Next, we will use the same parameters in our system and compare the metrics we collected with the emulated results. Moreover, to enhance the realism of our deployment, we plan to replace the data shareable folder with a distributed storage emulation, such as HDFS.

References

1. 5G Automotive Association: C-ITS vehicle to infrastructure services: How C-V2X technology completely changes the cost equation for road operators. White paper 2019.
2. Austria, S.: Federal ministry for climate action, environment, energy, mobility, innovation and technology. `https://www.senderkataster.at/`
3. Beilharz, J., Wiesner, P., Boockmeyer, A., Brokhausen, F., Behnke, I., Schmid, R., Pirl, L., Thamsen, L.: Towards a staging environment for the internet of things. In:

2021 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops). pp. 312–315 (2021)
4. Catlett, C.E., Beckman, P.H., Sankaran, R., Galvin, K.K.: Array of things: A scientific research instrument in the public way: Platform design and early lessons learned. In: Proceedings of the 2nd International Workshop on Science of Smart City Operations and Platforms Engineering. p. 26–33. ACM (2017)
5. Chen, B., Wan, J., Celesti, A., Li, D., Abbas, H., Zhang, Q.: Edge computing in iot-based manufacturing. IEEE Communications Magazine **56**(9), 103–109 (2018)
6. Chintapalli, S., Dagit, D., Evans, B., Farivar, R., Graves, T., Holderbaugh, M., Liu, Z., Nusbaum, K., Patil, K., Peng, B.J., Poulosky, P.: Benchmarking streaming computation engines: Storm, flink and spark streaming. In: IEEE IPDPSW (2016)
7. Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking cloud serving systems with ycsb. In: SoCC. ACM (2010)
8. Coutinho, A., Greve, F., Prazeres, C., Cardoso, J.: Fogbed: A rapid-prototyping emulation environment for fog computing. In: IEEE ICC (May 2018)
9. Dignan., L.: Iot devices to generate 79.4zb of data in 2025, says idc, 2019. `https://bit.ly/3MVTY15`
10. Georgiou, J., Symeonides, M., Kasioulis, M., Trihinas, D., Pallis, G., Dikaiakos, M.D.: Benchpilot: Repeatable & reproducible benchmarking for edge micro-dcs. In: Proceedings of the 27th IEEE ISCC (2022)
11. Hasenburg, J., Grambow, M., Bermbach, D.: Mockfog 2.0: Automated execution of fog application experiments in the cloud. IEEE TCC **11**(01), 1–1 (apr 2021)
12. Intel: Intel nuc for edge compute. `https://www.intel.com/content/www/us/en/products/docs/boards-kits/nuc/edge-compute.html`
13. Karimov, J., Rabl, T., Katsifodimos, A., Samarev, R., Heiskanen, H., Markl, V.: Benchmarking distributed stream data processing systems. In: IEEE ICDE (2018)
14. Lantz, B., Heller, B., Mckeown, N.: A network in a laptop: Rapid prototyping for software-defined networks. In: In ACM SIGCOMM HotNets Workshop (2010)
15. Li, M., Tan, J., Wang, Y., Zhang, L., Salapura, V.: Sparkbench: A comprehensive benchmarking suite for in memory data analytic platform spark. In: ACM International Conference on Computing Frontiers (2015)
16. Nikolaidis, F., Chazapis, A., Marazakis, M., Bilas, A.: Frisbee: A suite for benchmarking systems recovery. In: Proceedings of the 1st Workshop on High Availability and Observability of Cloud Systems. HAOC (2021)
17. Rathijit, S., Abhishek, R., Alekh, J.: Predictive Price-Performance Optimization for Serverless Query Processing. International Conference on Extending Database Technology, EDBT (2023)
18. Rausch, T., Lachner, C., Frangoudis, P.A., Raith, P., Dustdar, S.: Synthesizing plausible infrastructure configurations for evaluating edge computing systems. In: 3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20). USENIX Association (Jun 2020)
19. Symeonides, M., Georgiou, Z., Trihinas, D., Pallis, G., Dikaiakos, M.D.: Fogify: A fog computing emulation framework. In: IEEE/ACM SEC (2020)
20. Symeonides, M., Trihinas, D., Georgiou, Z., Pallis, G., Dikaiakos, M.: Query-Driven Descriptive Analytics for IoT and Edge Computing. In: Proceedings of IEEE International Conference on Cloud Engineering (IC2E 2019) (2019)
21. Symeonides, M., Trihinas, D., Pallis, G., Dikaiakos, M.D., Psomas, C., Krikidis, I.: 5g-slicer: An emulator for mobile iot applications deployed over 5g network slices. In: IEEE/ACM IoTDI (2022)
22. Zeng, Y., Chao, M., Stoleru, R.: Emuedge: A hybrid emulator for reproducible and realistic edge computing experiments. In: IEEE ICFC (2019)