BenchPilot: Repeatable & Reproducible Benchmarking for Edge Micro-DCs

Joanna Georgiou*, Moysis Symeonides*, Michalis Kasioulis*, Demetris Trihinas†,
George Pallis*, Marios D. Dikaiakos*

* Department of Computer Science
University of Cyprus
Email: { jgeorg02, msymeo03, mkasio01, gpallis, mdd }@cs.ucy.ac.cy

† Department of Computer Science
University of Nicosia
Email: trihinas.d@unic.ac.cy

***Abstract*—Micro-Datacenters (DCs) are emerging as key enablers for Edge computing and 5G mobile networks by providing processing power closer to IoT devices to extract timely analytic insights. However, the performance evaluation of data stream processing on micro-DCs is a daunting task due to difficulties raised by the time-consuming setup, configuration and heterogeneity of the underlying environment. To address these challenges, we introduce BenchPilot, a modular and highly customizable benchmarking framework for edge micro-DCs. BenchPilot provides a high-level declarative model for describing experiment testbeds and scenarios that automates the benchmarking process on Streaming Distributed Processing Engines (SDPEs). The latter enables users to focus on performance analysis instead of dealing with the complex and time-consuming setup. BenchPilot instantiates the underlying cluster, performs repeatable experimentation, and provides a unified monitoring stack in heterogeneous Micro-DCs. To highlight the usability of BenchPilot, we conduct experiments on two popular streaming engines, namely Apache Storm and Flink. Our experiments compare the engines based on performance, CPU utilization, energy consumption, temperature, and network I/O.**

***Index Terms*—Edge Computing, Internet of Things**

I. Introduction

The number of interconnected embedded devices is rapidly exploding and transforming our physical world into a cyber-physical ecosystem, the Internet of Things (IoT), with reports claiming the number of "things" will reach 41 billion by 2025 [1]. These devices generate a vast amount of data that needs to be processed to produce useful insights. However, processing on IoT devices is not always an option due to their inherent limitations, such as poor reliability, restrictive processing power and storage capacity [2]. On top of that, the limited bandwidth and increased latency of IoT-to-Cloud connections make the offloading of processing to centralized Cloud data centers a sub-optimal solution.

Edge Computing is promoted as a reasonable alternative for analyzing upstream and downstream IoT data, on behalf of cloud services [3]. Micro-DCs are key enablers for IoT data processing and analytic pipelines while also playing a significant role in 5G deployments [4]. An Edge micro-DC consists of a small cluster of co-located servers that create a shareable pool of resources across different users and applications. In contrast to warehouse-scale cloud data centers, micro-DCs are usually on-premise deployments assembling limited computing resources with increased heterogeneity, e.g., farms of single-board ARM-based micro-controllers with limited capabilities (1-4 cores @ 1.5GHz with 1-4GB RAM) and servers with increased processing power (e.g., 2-8 cores @ 4GHz with 8-32 GB RAM) [2]. The emergence of micro-DCs as platforms for deploying IoT-related applications raises the need to evaluate platform and application performance under various configurations and workload conditions. This is a challenging task because of the large space of alternative configurations that typically arise in such context and affect system and application performance in distributed infrastructures [5].

Typically, the evaluation of a system is performed through systematic benchmarking, which is a procedure that methodologically stresses a system under test, and, at the same time, observes its reactions through a thorough analysis of an expansive list of quantitative and qualitative performance metrics [5], [6]. The well-established benchmarks from the data management and cloud computing community [7]–[12] usually provide only workloads and data generators. The latter leads to the enforcement of end-users to manage burdens, such as the management of pre-deployment dependencies, hardware, and software configurations. Moreover, they do not offer provisions for diverse hardware architectures, e.g., ARM processors. Consequently, users have to rebuild the executables to be compatible with the underlying infrastructure. Additionally, state-of-the-art studies focus on application performance metrics without taking into account infrastructure metrics, e.g. energy consumption, while their metric collection processes are not extensible. Thus, there is a need for understandable and comparable performance indicators that capture both application and edge-oriented metrics. The lack of a comprehensive benchmarking suite becomes an extra pain for academic and industrial researchers since they need to build ad-hoc tools for quantifying the performance of their analytic tasks.

To tackle all these difficulties, we introduce BenchPilot, an open-source auto-benchmarking framework focusing on Edge micro-DCs. BenchPilot simplifies and accelerates the setup and evaluation of experiments that target edge infrastructures. It performs repeatable studies, captures hardware and software measurements, and, in general, allows users to focus on analyzing the monitoring data without dealing with complex and

978-1-6654-9792-3/22/$31.00

time-consuming setups. Even though, for now, BenchPilot's workloads focus on the evaluation of streaming distributed processing engines (SDPEs), anyone can easily extend it to support other kinds of workloads as well. For this work, we perform an extensive evaluation study on two state-of-the-art SDPEs on a micro-DC to highlight BenchPilot's applicability. Utilizing well-known streaming workloads [10], we reveal hidden insights for SDPEs performance, latency, resources' utilization, and energy consumption. In a nutshell, the main contributions of this paper are:

- An **open-source and extensible benchmarking framework**[1] that facilitates automated instantiation, repeatable benchmarking, monitoring of performance and infrastructure, post-experimentation analysis, and generally every barrier that a researcher faces during the performance evaluation. BenchPilot utilizes cloud technologies, like infrastructure-as-a-code and containerization, to alleviate difficulties that are raised from benchmark setup, and to abstract the micro-DCs hardware's heterogeneity.
- A **publicly available repository of a containerized streaming workload**[2], which is used by the framework out-of-the-box to evaluate the performance of micro-DCs. This workload is based on the widely-used Yahoo Streaming Workload [10]. Since the workloads are decoupled from the BenchPilot framework, researchers are able to use, extend, or update them according to their needs.
- A **comprehensive analysis of state-of-the-art SDPEs deployed on a micro-DC**. Our study compares the performance of Apache Storm & Flink, and edge nodes' utilization metrics, like CPU utilization, network traffic, etc., using as a testbed of 5 heterogeneous nodes.

The rest of the paper is structured as follows: Section II presents the related work, Section III introduces the BenchPilot Framework, and describes its implementation aspects. A comprehensive experimentation is illustrated in Section IV, and finally, Section V concludes the paper.

II. Related Work

Edge Computing engineers usually utilize experimentation tools to compare different frameworks, applications, and edge infrastructures. For instance, Fog and 5G emulators [2], [4] provide toolsets for topology emulation, application deployment, and extracting monitoring metrics from applications and emulated infrastructures. FlockAI [13] introduces an ML testing suite with a designing interface that enables creating custom algorithms, evaluating their performance and energy footprint via simulation. DeFog [12], which was one of the first Fog-oriented benchmarking projects, offers a methodology for benchmarking on Fog deployments by examining applications, including object classification, text-to-speech conversion, and online gaming. However, *these tools do not provide deployment on physical nodes or they are only focused on microservices or ML applications*.

Other than the evaluation tools described above, there are plenty of benchmarking studies in the data management community. For instance, HiBench [7] introduces big data workloads for ML, batch processing, and graph analytics, while Yahoo YCSB [8] offers a set of artifacts for DBMS benchmarking. Additionally, SparkBench [9] offers ML, graph computation, SQL queries, and streaming workloads for Spark, while a streaming comparison in latency and throughput between engines was presented in [10], in which the authors implemented a full data pipeline to reproduce real-world scenarios with high realism. Karimov et al. [11] gave a strict definition for the latency of stateful operators and a method to measure it. To have fair results, they decoupled the underlying systems from the driver controlling the experiments. Despite the plethora of benchmarking studies, *little or no provision is made in containerization, automation of deployment, and infrastructure monitoring in these efforts.*

There are only a few attempts to build a framework that automates the evaluation process. One example is Plug and Play Bench (PAPB) [14], which tries to facilitate big data benchmarking by utilizing containers to run on the underlying execution environment. PAPB proof-of-concept implementation uses batch and ML workloads of the HiBench benchmark suite, deployed on public cloud providers. Moreover, Frisbee [15] performs chaos engineering experiments on Kubernetes-deployed containerized applications. It provides a declarative language for failure injection and deployment, while also captures measurements from running services. These systems are promising but *focus only on Cloud without considering the heterogeneity raised by Edge clusters, and do not provide data-intensive streaming workloads.*

III. The BenchPilot Framework

A. System Overview

To date, operators and researchers need to evaluate the performance of their Edge infrastructures by performing repeatable studies with representative workloads. For that reason, users either need to build these workloads, which could be extremely time-consuming, or utilize already created benchmarking suites. Following the second approach, users must setup a batch of software dependencies, configure the underlying hardware and software infrastructure, and install monitoring tools that may yield unnecessary pain, turning the user's concentration away from the actual purpose of performance evaluation. BenchPilot helps users conduct performance evaluation studies by automating the benchmarking process, capturing performance, infrastructure, and application measurements, and facilitating the analysis of the captured metrics. The framework aids both experienced and amateur users to rigorously evaluate their micro-DCs, without dealing with the complex and time-consuming benchmarking.

To ease understanding, let us consider a scenario where an operator wants to analyze streaming data on a self-hosted micro-DC. The user does not know which Streaming Distributed Processing Engine (SDPE) is the most efficient for the micro-DC's underlying hardware, so it is necessary to make multiple deployments, perform repeatable trials, and analyze the generated results. Contrary to this approach, the user can

[1] https://github.com/UCY-LINC-LAB/BenchPilot
[2] https://hub.docker.com/repository/docker/benchpilot/benchpilot

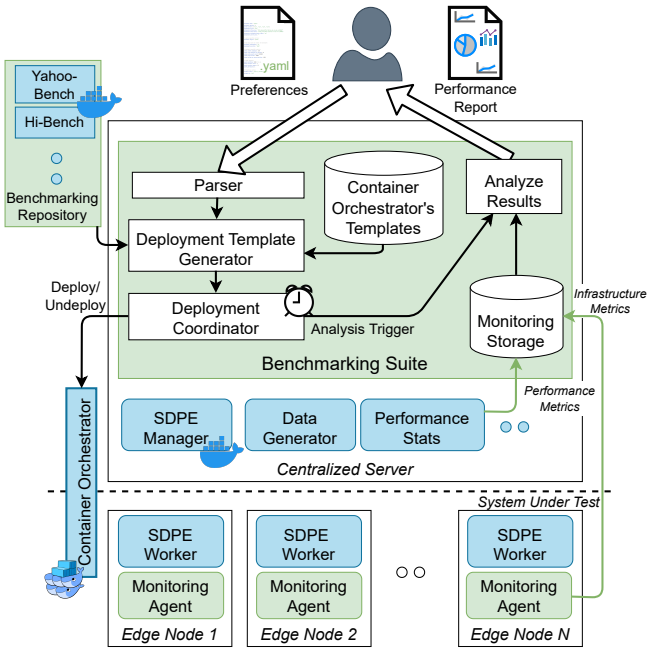


Fig. 1: The Overview of BenchPilot Framework

follow the BenchPilot benchmarking procedure. Starting from the **micro-DC bootstrapping**, the BenchPilot user only needs to execute a parameterizable installation script of Benchpilot on every DC node. The script installs all necessary software dependencies across the micro-DC and downloads the required workload docker images. This step is required only for the first time of the BenchPilot installation or in case of a hardware update, e.g., introducing a new device.

Then, the user initiates the BenchPilot benchmarking procedure, as depicted in Figure 1. The pipeline starts with the user submitting a set of experiments described in BenchPilot's high-level **descriptive language** for experimentation scenarios. Through the declarative specification, the user is able to describe the experimentation's parameters, including specific workload parameters, such as the application under test, data injection rate, etc., and infrastructure parameters, like placement preferences, parallelization, etc. The framework parses and evaluates the user's preferences, and propagates them to the **Deployment Template Generator**. The responsibility of the Deployment Template Generator is to construct an in-memory data structure that keeps a set of deployment elements, where each element is a materialized view of the user's preferences. To do that, the Deployment Template Generator invokes the container orchestrator's templates, which can be parameterized accordingly for the SDPE's services, and combines them with the containerized streaming workloads from the BenchPilot's repository, that came from open-source benchmarking suites. The output is a list of ready-to-use experiment descriptions, which will be used by the Deployment Coordinator for driving the execution on the micro-DC.

Specifically, for each experiment description, the **Deployment Coordinator** generates a deployment description and transports it to the **Container Orchestrator**, which is responsible for managing the deployment, scaling, and networking of containerized services, and is materialized by orchestration frameworks, such as Docker Swarm or Kubernetes. When all services are running, the Deployment Coordinator submits the respective executable to the underlying SDPE manager, e.g. Spark Master, initiates the data generator to start producing data that engine workers will consume, and observes the experiment execution. Afterward, it records the starting and ending experiment's timestamps, and propagates a new deployment to the Container Orchestrator. When all experiments are over, users can start the **Post-Experimentation Analysis**, in which the user requests the monitored metrics of each execution from the monitoring storage based on the provided experiments' starting/ending timestamps. Users are able to apply high-level analytic models to the retrieved metrics of each experiment. With the performance measurements and the generated model-based metrics, users can have a clear overview of their deployments.

B. Implementation Aspects

This subsection provides the implementation aspects about the workload containerization, the experimentation modeling, the benchmarking execution, and the monitoring stack.

Workload Containerization: is a virtualization technique that abstracts the bare-metal infrastructure and enables rapid application development with negligible performance impact [16]. BenchPilot utilizes Docker and its ecosystem as a containerization engine. In the Docker ecosystem, users describe their services through Dockerfiles, which contain every required instruction and environment variable for the service's execution. Users need to compile their Dockerfiles to create the respective Docker images and then execute them on the host machine as Docker Containers. Analogous to inheritance in OOP, a Docker image can inherit another Docker image if the user specifies the latter as the base image in the Dockerfile. Taking advantage of this functionality, BenchPilot introduces the BenchPilot base image, which includes all necessary packages and dependencies required by other BenchPilot images. Then, for each encapsulated workload and SDPE, BenchPilot creates specific images. In order to cope with hardware heterogeneity, arising for instance in micro-DC that comprise both x64 and ARM architectures, BenchPilot provides compiled images for both architectures and consequently its Dockerfile compilation can generate images compatible to the target hardware. Furthermore, to facilitate the dissemination of the images, the Docker ecosystem introduces Docker image registries, public or private, capable of hosting, indexing, and managing Docker images. The most popular public registry is the DockerHub, so we upload every image artifact there to be publicly available to the research and industry community.

Experimentation Modeling: BenchPilot provides users a high-level model description that follows YAML representation to describe multiple and repeatable experiments. Description 1 depicts an example of a modeling specification. Users start the modeling by providing a set of experiment objects existing under the *experiments* YAML section. Each experiment is separated into three subsections, namely, (i) the *workload*,

```
experiments:
  - workload:
      name: "marketing-campaign"
      repetition: 1
      duration: "600s"
      parameters:
        campaigns: 100
        tuples_per_second: 1000
        capacity_per_window: 10
        time_divisor: 10000
    cluster:
      nodes: ["nc1","rpi0","rpi1","rpi2","rpi3"]
    engine:
      name: "spark"
      parameters:
        batchtime: 1000ms
        executor_cores: 1
        executor_memory: 1G
        partitions: 5
  - workload:
      name: "marketing-campaign"
                    ....
```

Description 1: Example Experiment Modeling

(ii) the *cluster*, and (iii) the *engine*. Into the workload subsection, users select the workload of their choice through the *name* property (e.g., marketing-campaign), the *repetition* that indicates how many times the workload will be executed, the *duration* that the workload will run at each execution, and a set of *parameters*, which are provided by the respective workload. For instance, the marketing-campaign workload parameters include the number of stored *campaigns*, how many tuples will be produced per second (*tuples_per_second*), and so on. Then, the user defines the *cluster* that describes a subset of the micro-DC *nodes*. The modeling materializes the definition of the nodes via a list of their hostnames. Engine workers will be deployed during the experiment on these nodes. Then, users specify the *engine*, which represents the SDPE along with the framework's *parameters*. For instance, the first experiment of Description 1 will be running on a spark cluster with the streaming analytic job *batchtime* to be 1000ms, the workers will start executors equipped with 1 core (*executor_cores*) and 1G memory (*executor_memory*), and the incoming data stream will be re-partitioned into 5 executors (*partitions*). Finally, users similarly define all the experiments of their interest.

Benchmarking Execution: Template Generator and Deployment Coordinator subcomponents are responsible for the translation of experiment description to sequential containerized workload executions. Specifically, Template Generator creates a deployable template for the underlying Container Orchestrator. State-of-the-art container orchestrator frameworks, e.g., Docker Swarm or Kubernetes, allow users to define the placement policies of their services. Taking advantage of this, BenchPilot defines the placement of benchmark services and the worker's node of the SDPE based on the given hostnames of the DC nodes. For our prototype, we use Docker Swarm as the underlying container orchestrator. However, it is possible to integrate other docker orchestrators (like Kubernetes) by extending the placement policy interface of the BenchPilot platform. Furthermore, Template Generator passes SDPE's parameters through in-line configurations and container environmental variables to the containerized benchmark services. Then, the Deployment Coordinator submits one by one the descriptions to the underlying container orchestrator, records the timestamps of starting and finishing points of each experiment, and destroys the deployment when all the experiments are finalized. Finally, the Deployment Coordinator outputs the starting and finishing points of each experiment to the user, and can perform time range queries on the monitoring stack.

Monitoring Stack: BenchPilot offers a transparent, from the application under test, monitoring stack capable of extracting various infrastructure utilization metrics, including CPU, Memory, and Network Utilization. To achieve this, BenchPilot instantiates a containerized monitoring agent on every node. The agent inspects system information (e.g., performance pseudofiles and cgroup files) and extracts the required metrics in a non-intrusive way. The agent starts various probes, one for each sub-component (e.g., cgroup probe, OS probe, etc.), and exposes an API through which a centralized monitoring server retrieves the data periodically and stores them to the monitoring storage. Furthermore, the monitoring agent offers probes for external resources as well. For instance, in our testbed, we expose energy measurements to the monitoring agent through SNMP protocol, while we also configure the extraction of workload and SDPE specific metrics. From the implementation perspective, we have selected Netdata[3] a widely known and used monitoring tool, and Prometheus[4], an open-source and popular monitoring server, for our stack. By default, Prometheus uses indexed files to store its data but also offers a list of possible pluggable backends. We choose InfluxDB[5] as the monitoring storage backend because it minimizes the data retrieval time by applying index-based optimizations on time-series datasets.

IV. Experimentation

This section showcases the utility of BenchPilot through repeatable experiments that adopt an open, popular and realistic workload to compare the performance and utilization of two state-of-the-art Streaming Distributed Processing Engines, Storm and Flink, on a representative micro-DC testbed.

A. Experiment Setup

Testbed. The testbed consists of the SDPE worker nodes and the BenchPilot client. The worker nodes include a Dell PowerEdge R610 server (*nc1*) featuring 12cores@2.4GHz with 12GB RAM and four Raspberry Pi 4 Model B (*rpi0-rpi3*) devices, equipped with a quad core ARM Cortex-A72@1.5GHz and 4GB RAM. The *nc1* server is connected to a Smart Power Distribution Unit (SPDU) and *rpi0-rpi3* to Meross Wi-Fi Smart Plugs[6], so that the BenchPilot monitoring tool can retrieve power and energy consumption data every 5s via the SNMP protocol. In turn, the BenchPilot client resides on a VM with 16 vCPUs and 16GB RAM, alongside the Prometheus server and the data generator.

Workload. For this evaluation experiment, we adopt the open and popular Yahoo Streaming Benchmark [10] that has

[3] https://www.netdata.cloud/ [4] https://www.prometheus.io/
[5] https://www.influxdata.com/ [6] https://www.meross.com/product/3/article/

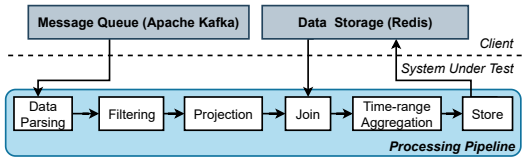


Fig. 2: Overview of Yahoo Streaming Workload

been containerized and extended to support parameterization through BenchPilot experiment templates and thus, without the need to change and re-compile coded artifacts. In particular, the workload features an application as a data processing pipeline with multiple and diverse steps that emulate insight extraction from marketing campaigns. Figure 2 depicts the workload pipeline that includes: (i) incoming advertising traffic data, (ii) filtering, (iii) projection of unnecessary values, (iv) combination with existing information, and (v) storing of the results. All data generated by the data generator are extracted from the pipeline through a message queue (Apache Kafka), while an in-memory database (Redis) stores intermediate data and the generated results. The workload supports execution on the examined SDPEs, while the data rate and number of campaigns are parameterizable. To evaluate performance, a post-experiment script is made available that computes the processing latency based on the campaign event timestamps. With this, latency is automatically computed for each batch of events and timestamped metrics are stored in the BenchPilot monitoring stack for real-time and post-experiment analysis.

B. Experiments & Results

We evaluate the performance and resource utilization of Flink compared to Storm embedding; we configure Storm with two alternative schedulers, namely its default scheduler and R-Storm. The default Storm scheduler adopts a fairness strategy for operator placement, while R-Storm is resource-aware [17], attempting to optimize operator placement by scheduling more operators on powerful workers so that the inter-communication overhead during data shuffling is reduced. The default parameterization for the rest of the software components (i.e., Kafka, Redis) has not been altered with the exception of the number of Kafka partitions set to 5 and equivalent to the number of workers to achieve maximum parallelization during data injection. For each experiment run, the workload is injected for 10 minutes (600 seconds) with BenchPilot configured to monitor the execution environment for a period of 150 seconds before load injection (warm-up period) and 50 seconds afterwards (cool-down period). Finally, we must elaborate that our goal is not to fully optimize the performance of the SDPEs but rather showcase BenchPilot's usability via interesting insights for SDPEs deployed on edge micro-DCs.

Performance Analysis: In this experiment run, we execute the workload under different profiles: (i) **lightweight**, featuring 50k tuples per second data generation rate and 50k campaigns; and (ii) **heavyweight**, in which we introduce 500k tuples per second and 50k campaigns. Figure 3 depicts for both profiles the processing latency as box plots (without outliers) as generated by BenchPilot during the post-analysis. We observe

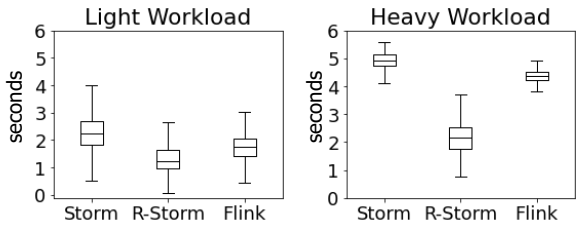
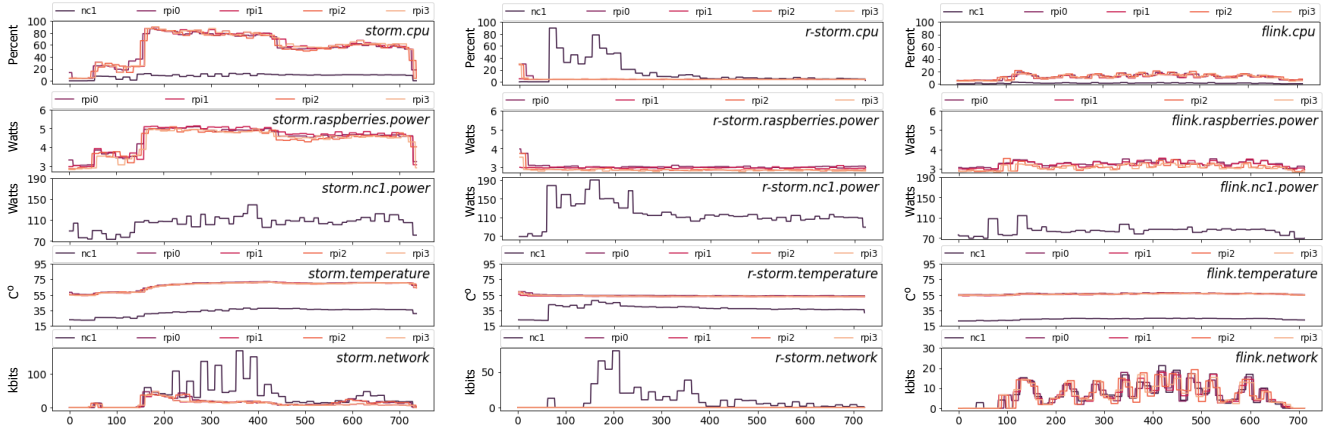


Fig. 3: Latency Box-Plot from BenchPilot Post-Analysis

that Storm has better performance in both experiments with the scheduler not playing an important role for the light workload. However, for the heavy workload, we observe how crucial it is for micro-DCs to adopt a resource-aware scheduler especially in cases with significant resource heterogeneity. In conclusion, *operator placement and parameter tuning of a SDPE is crucial to amplify performance, especially in heavy load executions*.

Next, Fig. 4 depicts resource utilization metrics exposed by BenchPilot for the SDPEs during the heavy load experiments. **CPU utilization**: First, we observe that for Storm (*storm.cpu*), the Pi's present significant cpu usage ranging from 60%-80%. On the contrary, R-Storm favors *nc1* and hence, the cpu usage of the server is between 20%-80% while the Pi's are under-utilized. To understand why the *r-storm.cpu* is initially $\sim$70% and then drops to $\sim$20% one just needs to inspect the workload, to understand that Storm initially retrieves and caches the data needed for the join operation. So, the workload uses more resources at the beginning of the batch, causing the minimization of execution time and resource utilization when using the R-Storm placement. In turn, the Flink testbed seems to be under-utilized, in contrast to default Storm, and despite not embedding a resource-aware scheduler (nc1 actually has lower cpu than the Pi's) it's ability to handle efficiently iterative tasks (i.e., joins) with delta increment computations, fits extremely well for this workload. Thus, *powerful nodes tend to be underutilized in a heterogeneous realm, and default schedulers and parameters of SDPEs do not tackle this issue.* Furthermore, *sophisticated underlying implementation techniques, like delta increment computations, may have a significant impact on infrastructure resource utilization, as well as, workload processing latency.*

Energy Consumption & Temperature: are correlated with CPU utilization, especially in the case of edge devices. The energy consumption, as the literature for edge devices indicates, can be computed using a regression model of the processor utilization (i.e., P_{active}, P_{idle}) [18]. In all runs, nc1 consumes most of the micro-DC's energy, with more than 80% of total energy consumed during the experiments. The latter could be expected since nc1's P_{active} and P_{idle} are much higher than an edge device's. The *storm/r-storm/flink.temperature* is the average temperature of all cores of a node measured in Celsius. The measured temperatures of the edge devices are much more elevated than the ones nc1 has. Intuitively, one understands that the server has its own cooling system while edge devices do not have any cooling provision. In conclusion, *the energy consumption of a cluster is dominated by the powerful servers, even if they are underutilized, while*



(a) Apache Storm (b) Apache R-Storm (Resource-aware) (c) Apache Flink

Fig. 4: SDPE Resource Utilization for Heavy Workload

the observed high-temperature of edge devices could cause sustainability and failure issues in an edge micro-DC.

Network Traffic: Lastly, there is the network traffic, which is the sum of incoming and outgoing data volume in each interval. For Storm (*storm.network*), even if nc1 is underutilized, the network traffic indicates that an underutilized server can process much more tuples than a fully utilized edge device. On the contrary, Flink's traffic (*flink.network*) does not change among the nodes. Finally, R-Storm (*r-storm.network*) transfers traffic only through nc1 that has all deployed tasks, as it is expected. In conclusion, *the placement of the streaming operators on nodes dictates the network traffic in micro-DCs.*

V. Conclusion

In this work, we introduced BenchPilot, a framework that facilitates the repeatable, reproducible, and rapid benchmarking and experimentation on Edge micro-DCs. BenchPilot features a high-level declarative language for customizable benchmarking, seamless micro-DC deployment via containerization technologies, and an extensible monitoring stack. Focusing on SDPEs, we provided a containerized streaming workload from a real-world marketing application, and we performed repeatable trials on different SDPEs (Apache Storm & Flink) that are deployed on a representative Edge micro-DC. Our results illustrate that (i) parameter tuning of SDPEs is crucial to amplify performance and to minimize network traffic among nodes, with resource-aware Apache Storm to have the best performance in all trials, while Apache Flink needs fewer resources from the underlying nodes, (ii) powerful servers tend to be underutilized in most of the experiments, and (iii) energy consumption and nodes' temperature are correlated with the CPU utilization, with the overall energy consumption of micro-DCs to be dominated by the powerful servers. Our future work includes the encapsulation of more streaming workloads, evaluation of more SDPEs (e.g., Apache Spark), and the deployment of BenchPilot on diverse Edge micro-DCs.

Acknowledgement. This work is partially supported by the EU Commission through RAINBOW 871403 (ICT-15-2019-2020) project, the Cyprus Research and Innovation Foundation through COMPLEMENTARY/0916/0916/0171 project, and from RAIS (Real-time analytics for the Internet of Sports), Marie Skłodowska-Curie Innovative Training Networks (ITN), under grant agreement No 813162.

References

[1] "L. dignan. (2019), iot devices to generate 79.4zb of data in 2025, says idc," https://www.zdnet.com/article/iot-devices-to-generate-79-4zb-of-data-in-2025-says-idc/.

[2] M. Symeonides, Z. Georgiou, D. Trihinas, G. Pallis, and M. D. Dikaiakos, "Fogify: A fog computing emulation framework," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*, 2020, pp. 42–54.

[3] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[4] M. Symeonides, D. Trihinas, G. Pallis, M. D. Dikaiakos, C. Psomas, and I. Krikidis, "5g-slicer: An emulator for mobile iot applications deployed over 5g network slices," in *IEEE/ACM IoTDI*, 2022.

[5] B. Varghese, N. Wang, D. Bermbach, C.-H. Hong, E. D. Lara, W. Shi, and C. Stewart, "A survey on edge performance benchmarking," *ACM Comput. Surv.*, 2021.

[6] M. D. Dikaiakos, "Grid benchmarking: Vision, challenges and current status," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 1, pp. 89–105, 2007.

[7] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The hibench benchmark suite: Characterization of the mapreduce-based data analysis," in *IEEE ICDEW*, 2010.

[8] 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: Association for Computing Machinery, 2010.

[9] M. Li, J. Tan, Y. Wang, L. Zhang, and V. Salapura, "Sparkbench: A comprehensive benchmarking suite for in memory data analytic platform spark," in *ACM International Conference on Computing Frontiers*, 2015.

[10] S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng, and P. Poulosky, "Benchmarking streaming computation engines: Storm, flink and spark streaming," in *IEEE IPDPSW*, 2016.

[11] J. Karimov, T. Rabl, A. Katsifodimos, R. Samarev, H. Heiskanen, and V. Markl, "Benchmarking distributed stream data processing systems," in *IEEE ICDE*, 2018.

[12] J. McChesney, N. Wang, A. Tanwer, E. de Lara, and B. Varghese, "Defog: Fog computing benchmarks," in *ACM/IEEE SEC, 2019*.

[13] D. Trihinas, M. Agathocleous, K. Avogian, and I. Katakis, "Flockai: A testing suite for ml-driven drone applications," *Future Internet*, vol. 13, no. 12, 2021.

[14] S. Ceesay, A. Barker, and B. Varghese, "Plug and play bench: Simplifying big data benchmarking using containers," in *IEEE BigData*, 2017.

[15] F. Nikolaidis, A. Chazapis, M. Marazakis, and A. Bilas, "Frisbee: A suite for benchmarking systems recovery," in *Proceedings of the 1st Workshop on High Availability and Observability of Cloud Systems*, ser. HAOC, 2021.

[16] R. Morabito, "Virtualization on internet of things edge devices with container technologies: A performance evaluation," *IEEE Access*, 2017.

[17] B. Peng, M. Hosseini, Z. Hong, R. Farivar, and R. Campbell, "R-storm: Resource-aware scheduling in storm," in *Proceedings of the 16th Annual Middleware Conference*, ser. Middleware. ACM, 2015.

[18] D. Trihinas, G. Pallis, and M. Dikaiakos, "ADMin: adaptive monitoring dissemination for the internet of things," in *IEEE INFOCOM*, 2017.