ComB: A Flexible, Application-Oriented Benchmark for Edge Computing

Simon Bäurle
Technical University of Munich
simon.baeurle@tum.de

Nitinder Mohan
Technical University of Munich
mohan@in.tum.de

ABSTRACT

Edge computing is an attractive platform where applications, previously hosted in the cloud, shift parts of their workload on resources closer to the users. The field is still in its nascent stages with significant ongoing innovation in small form-factor hardware designed to operate at the edge. However, the increased hardware heterogeneity at the edge makes it difficult for application developers to determine if their workloads will operate as desired. Simultaneously, edge providers have to make expensive deployment choices for the "correct" hardware that will remain suitable for the near future. We present ComB, an application-oriented benchmarking suite for edge that assists early adopters in evaluating the suitability of an edge deployment. ComB is flexible, extensible, and incorporates a microservice-based video analytics pipeline as default workload to measure underlying hardware's compute and networking capabilities accurately. Our evaluation on a heterogeneous testbed shows that ComB enables both providers and developers to understand better the runtime capabilities of different hardware configurations for supporting operations of applications designed for the edge.

CCS CONCEPTS

• **Computer systems organization** → **Distributed architectures**; • **Networks** → **In-network processing**; • **Software and its engineering** → **Software architectures**.

KEYWORDS

edge computing, benchmarking, next-generation applications

ACM Reference Format:
Simon Bäurle and Nitinder Mohan. 2022. ComB: A Flexible, Application-Oriented Benchmark for Edge Computing. In *5th International Workshop on Edge Systems, Analytics and Networking (EdgeSys '22), April 5–8, 2022, RENNES, France.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3517206.3526269

1 INTRODUCTION

Over the past decade, edge computing has emerged as a compelling solution to enable many next-generation networked applications. The primary selling point of edge is its ability to satiate latency-sensitive compute requirements of applications by moving workloads from centralized datacenter silos to infrastructures deployed closer to the users [7, 8, 10]. However, edge computing is most advantageous for bandwidth-hungry applications, such as live video analytics, augmented/virtual reality, etc., due to the possibility to pre-process and reduce the data sizes closer to the collectors [9, 29]. As a result, these applications have also been proclaimed as "killer" use-cases by the edge research community [6].

Several options for deploying edge servers have been proposed in recent years. While cloud providers are expanding their infrastructure to the edge by making use of spare CDN server capacity via technologies like Azure IoT Edge [28] or AWS Greengrass [4], ETSI and industrial standardization bodies aim to install compute hardware (like miniature datacenters) within their premises [3]. On the other hand, democratic adopters of the edge envision a world in which any entity with spare hardware can contribute to the infrastructure at large – thereby establishing a ubiquitous, crowdsourced compute fabric [15, 33]. While we believe that the edge computing concepts will continue to mature and a shared understanding of the infrastructure will persist, different assumptions regarding the capability of "edge infrastructures" are limiting the technological growth of applications aimed at the edge [29]. The uncertainty is further exacerbated by the increasing heterogeneity of edge devices, caused by the recent availability of specialized resources (e.g., Intel NCS2, Raspberry PI 4, or other accelerated devices) that take radical design choices to achieve smaller form factors. This results in several complications for both application providers and edge infrastructure operators. Not only would the application providers have to assess if their services would be optimally supported on the variety of edge hardware, but the early providers of edge computing also have to make expensive hardware investments in the hope that the majority of edge-oriented applications are compatible with their infrastructure.

To this end, we present ComB, which is a comprehensive benchmarking suite designed explicitly for edge infrastructures and applications. ComB enables edge operators to understand, assess, and adjust their infrastructure to become more suitable for different edge applications (§3). Unlike existing benchmarks, ComB relies on realistic microservice-based edge-oriented applications instead of artificial workloads to provide a realistic assessment of a hardware's capabilities for supporting edge computing tasks. In the current implementation, we use a modular distributed live video analytics pipeline as our application workload – which detects and tracks objects across video frames captured from live video cameras (§4). We also allow developers to easily replace the workload in ComB to the application of their choice with minimal programming overhead – enabling them to explore counter-intuitive deployment options.

EdgeSys '22, April 5–8, 2022, RENNES, France
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9253-2/22/04…$15.00
https://doi.org/10.1145/3517206.3526269

ComB includes its own scheduling and service placement scheme for mapping workload tasks over a distributed infrastructure, but is expandable to support widely-used orchestration platforms (e.g., K3s [1]). As we show in §5, ComB can accurately highlight the runtime capabilities of heterogeneous hardware with different processor configurations and provide valuable metrics to users regarding the realistic operation of edge infrastructure.

2 BACKGROUND AND RELATED WORK

Benchmarking edge devices is not a new endeavor, and several works in the past have proposed toolchains for different flavors of edge computing.

Hardware benchmarking. Kruger and Hanke [20] propose a two-phase benchmark that measures the performance of constrained off-the-shelf hardware (Raspberry Pi, BeagleBone Black, etc.). In the first phase, Lmbench [27] is used to compute operation speeds and latencies to determine the overall machine performance. The second phase deploys a gateway for the constrained application protocol (CoAP) on the devices and measures the response latency for repeated requests. While the benchmark provides a good overview of the CPU and network performance, the included workloads are quite simplistic and do not highlight the potential for realistic edge applications. RIoTBench [31] focuses on distributed stream processing system (DSPS) that provides a dataflow model for scalable, low-latency streaming-based applications. The suite includes 27 distinct micro-benchmarks that address different types of streaming tasks (e.g., data parsing, filtering or pattern detection, visual analytics, I/O operations). IoTBench [22] provides benchmark tasks imitating edge processing of IoT applications. The authors provide workloads for computer vision (e.g. video summarization, image recognition, etc.), speech recognition and signal processing.

Neural network based benchmarking. Due to the increasing popularity and use of neural networks and machine learning in applications, many researchers have designed benchmarking tools specifically to assess a device's capability to support such workloads. Reuther et. al. [30], and Dinelli et. al. [13] benchmark, study and compare the performance of different pluggable hardware accelerators (e.g., Google Edge TPU, Intel Movidius Compute Stick) with general CPU execution. Their research primarily focuses on the performance of different accelerators and not on application-based generic edge benchmarking. OpenRTiST [16] implements an end-to-end benchmark that utilizes neural style transfer (NST) to transform a live video stream in a reference painting. The benchmark takes the camera input from a mobile phone and processes the image on the cloud and the cloudlet. The benchmark provides end-to-end metrics for image processing and network communication between the user device and the resource executing the neural network. Scission [23] proposes a context-aware distribution of deep neural networks across multiple devices such that the individual layers are scattered across the edge-cloud continuum. The network layers are split at specified partitioning points – determined by the architecture of the neural net (linear or branching). While the approach provides insights into the distributed execution of convolutional neural networks, the benchmark disregards the networking performance as the metric is only simulated and never explicitly measured.

Edge infrastructure benchmarking. EdgeBench [11] focuses on benchmarking the performance of the commercial edge-like offerings, specifically AWS IoT Greengrass [4] and Azure IoT Edge [28]. Since both platforms are based on serverless computing paradigm, the workloads in the toolchain are composed of service functions and includes a speech-to-text decoder, an image recognition model, and a scalar value generator. DeFog [26], on the other hand, is designed to support different possible infrastructure deployment options in the edge-cloud continuum. The benchmark includes a diverse mix of edge-oriented workloads, such as ML-based object classification, speech-to-text conversion, geo-location-based gaming, etc. The assets for these tasks are hosted in the cloud and transferred to the service destination upon scheduling. Yang et al. [32] implement an extensible benchmarking suite for functional edge workflows, also named EdgeBench. The benchmark uses OpenFaaS on top of Kubernetes to execute either a video analytics pipeline or an IoT hub as predefined workloads. The suite comes clostest to ComB and allows the integration of custom workloads, however, their extensibility is limited to FaaS workloads which does not encompass stateful complex edge applications (e.g. using ML/AI).

ComB complements the previous work by providing a comprehensive and flexible benchmark suite for edge computing to date. We design ComB to support distributed pipelined tasks that measure the edge infrastructure's processing and networking capabilities. The in-built orchestration scheme of ComB enables developers to tweak and extend the workload design to benchmark their infrastructure as per the intended application operation. Our default video analytics workload is a good representation of neural network-based tasks common in edge applications. ComB is fully customizable and extensible and is therefore compatible with future innovations in edge hardware and applications. Additionally, we make ComB open-source[1] to allow developers and researchers to use and customize the benchmark for their specific edge infrastructure and use cases.

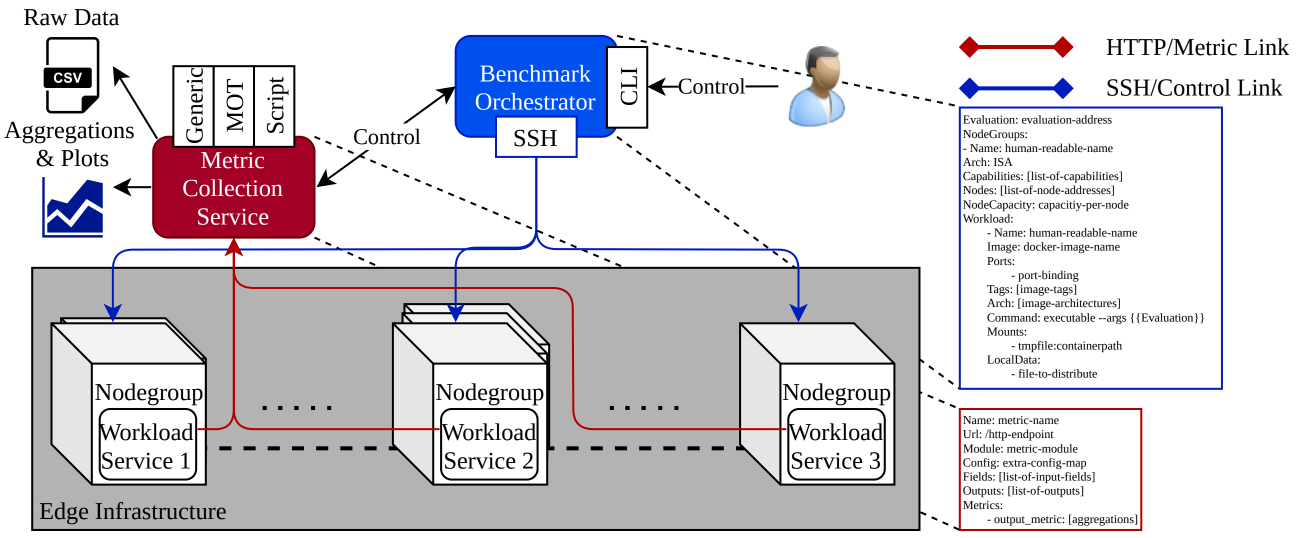
3 COMB OVERVIEW

Majority of the state-of-the-art benchmarks discussed in §2 do not accurately incorporate the complexities at the edge since they either rely on generic stress-based workloads or specialized applications that only assess a resource's capabilities for specific ML/neural network tasks. Contrarily, we believe that much like the growth of cloud computing, the intended applications, technologies, and hardware for the edge will continue to evolve. We envision a more generic and configurable benchmark suite that not only allows developers to assess their infrastructure's capabilities via provided workloads but also supports customized applications. In this regard, the benchmark suite can be viewed more like an orchestrator with several different applications as workload options. To accomplish this vision, we identify the following goals.

(1) The rapid development in hardware and software for the edge mandates a flexible, intuitive configuration structure that allows easy integration of novel applications. Adopting new metrics and processing procedures should only require minimal code changes in the benchmark suite.

(2) By definition, edge environments are heterogeneous and incorporate many specialized devices with varying network (or other)

[1] https://github.com/sbaeurle/ComB



```
Evaluation: evaluation-address
NodeGroups:
- Name: human-readable-name
Arch: ISA
Capabilities: [list-of-capabilities]
Nodes: [list-of-node-addresses]
NodeCapacity: capacitiy-per-node
Workload:
        - Name: human-readable-name
        Image: docker-image-name
        Ports:
                - port-binding
        Tags: [image-tags]
        Arch: [image-architectures]
        Command: executable --args {{Evaluation}}
        Mounts:
                - tmpfile:containerpath
        LocalData:
                - file-to-distribute
```

```
Name: metric-name
Url: /http-endpoint
Module: metric-module
Config: extra-config-map
Fields: [list-of-input-fields]
Outputs: [list-of-outputs]
Metrics:
        - output_metric: [aggregations]
```

Figure 1: Overview of ComB benchmark suite. The benchmark orchestrator (blue) finds the mapping and places the microservices of the underlying workload (see §4) to each device in the edge infrastructure. The metric collection service (red) collects the performance results of each microservice from the infrastructure, aggregates it and plots it in human-readable format.

constraints. Distributing application services on a fixed subset of these devices may produce counter-intuitive results, where initially unreasonable scenarios may appear more performant (e.g., high networking costs to the cloud might outweigh the penalty of a slightly less powerful edge accelerator). As such, the benchmarking suite must evaluate all reasonable configurations and service placements – guiding the developers towards such anomalies.

(3) The control plane of the benchmark must have a minimal footprint such that the inherent workload can assess the hardware's capability at its full capacity. Costly operations (such as I/O or postprocessing) should be decoupled from the collection routine. Local data persistence should be avoided at best since many edge devices typically rely on slower storage, such as SD-cards or eMMC storage. Furthermore, the suite must also support remote resource benchmarking over networking restrictions (e.g., firewalls or NATs), which are common in edge deployments.

3.1 System Architecture

ComB is designed as a split system, consisting of the *benchmark orchestrator* and the *metric collection service*. Figure 1 shows the cross-section of ComB components and their interaction with the edge infrastructure. Edge operators can register their infrastructure and schedule benchmark runs through a configuration file (outlined blue) in the *benchmark orchestrator*. The configuration includes details of edge resources grouped based on their hardware capabilities (a.k.a. node groups). For example, multiple Raspberry Pi 4 with the exact RAM and storage capacity in the infrastructure form a single node group since their performance is likely to be similar. Each node group is defined with hardware capabilities (e.g., CPU or GPU), underlying processor architecture and access to special peripherals (e.g., cellular dongle, accelerators, etc.). The configuration file also includes runtime information for the workload services, e.g., execution environment, processor architecture, port bindings, data volumes, etc. The operator can also specify a list of container image tags that explicitly correspond to the node group's hardware capability. This way, ComB can utilize specialized build artifacts specifically compiled for different hardware types, e.g. x86, ARM, CUDA-support, etc., without probing.

Using the supplied configuration settings, the *benchmark orchestrator* performs multiple iterations and combinations of workload placement on the infrastructure (detailed in §3.2). In the current version of ComB, the workloads (discussed in §4) are scheduled as docker containers via the built-in SSH executor. However, ComB allows future integration with widely-used orchestration platforms (e.g., Kubernetes or KubeEdge) and novel virtualization techniques (e.g., AWS Firecracker [5], Unikernels, etc.). For each benchmark run, the *metric system* asynchronously collects the results of the workload from the involved resources (see §3.3).

3.2 Benchmark Orchestration

ComB interprets the described configuration as a bipartite graph $G = (V_W \cup V_N, E)$, where workload services (V_W) and node groups (V_N) are connected with an edge iff the service requirements (tags T, processor architecture PA) match the node group (capabilities C, architecture A) $[T \cap C \neq \emptyset \wedge A \in PA]$. Each maximal matching (M) in this graph corresponds to a valid distribution of workload services across the node groups, with the set of all maximal matchings ($M_a = \{M_1, \ldots, M_n\}$) containing all possible schedules. Initially, we experimented with multiple algorithmic approaches to find optimal matchings [18, 21]. However, we found a brute-force matching scheme to be the best fitting performance-wise during the implementation, which we explain as follows. Since most resources and workloads will have a matching for CPU execution, this likely results in (almost) fully-connected graphs. As a result, the computational complexity of more efficient approaches converges towards $O(n!)$ with $n = |V_W| \wedge |V_N|$, similar to a brute-force approach. For

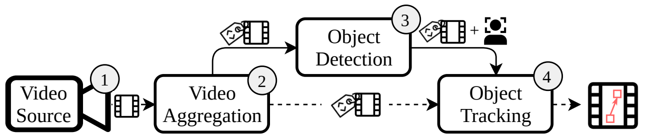


Figure 2: Architecture of the video analytics pipeline used as workload for ComB. Each square represents a microservice which can be independently executed on an edge device in the infrastructure. Arrowed links represent dependencies and data flows within the application pipeline.

reasonable workload sizes, the performance difference between efficient and brute-force matches becomes negligible, favoring the more straightforward implementation.

After all possible matchings are generated, the orchestrator filters them based on maximum service capacity and the number of nodes per node group – such that only sensible workload distributions are scheduled as different runs. The benchmark orchestrator uses a template for each run to generate (docker) scheduling commands over SSH connection towards selected resources. Our implementation allows us to later support other virtualizations and communication protocols through minor template adjustments.

3.3 Metric System

ComB also features a highly customizable metric system that receives and processes metrics from the benchmark workloads after every run. We implement the system as a separate component to allow flexible placement and overcome network restrictions within the benchmark (read edge) environment. Developers can express metrics as shallow JSON objects that get processed using configurable HTTP handlers (see configuration in blue outline in fig. 1). Any costly operations (such as calculating metrics or I/O) are fully decoupled from the HTTP endpoints and handled in separate execution routines to maintain asynchronicity. After each benchmark run finishes, ComB calculates configurable aggregations along with complex metrics, such as Higher Order Tracking Accuracy (HOTA [25]) or Multiple Object Tracking Accuracy (MOTA), and writes them to disk. In the current version, ComB supports three different metric modules, which are easily extensible. The GENERIC module converts received metrics to numerical values and outputs them in a machine-readable format. The SCRIPT module allows developers to provide custom tengo language [19] based scripts for quick and easy integration of custom calculation procedures without significant code modification. The MOT module integrates multiple object tracking code from the TrackEval [24] repository and exports valuable performance assessments from our video analytics workload. The module expects the current frame number, IDs, and bounding boxes of all currently tracked objects and logs them in the MOTChallenge [12] format.

4 COMB WORKLOAD

The quality of a benchmark's workload directly influences the representativeness of its results. We employ a microservice-based Multi-Object Tracking (MOT) pipeline implemented using modern application design principles and technologies. The following reasons influence our design:

(1) The workload should resemble potential edge applications and provide a realistic insight into the performance of the infrastructure. Moreover, the workload must be composed of services that rely on heavy and lightweight computations and must be interconnected to assess the communication between resources.

(2) We expect rapid development of new edge solutions, with potential to disrupt current development principles or technologies. As a result, the workload should be modular to support future technological extensions. Developers should be able to easily improve, replace or extend individual services without affecting other components.

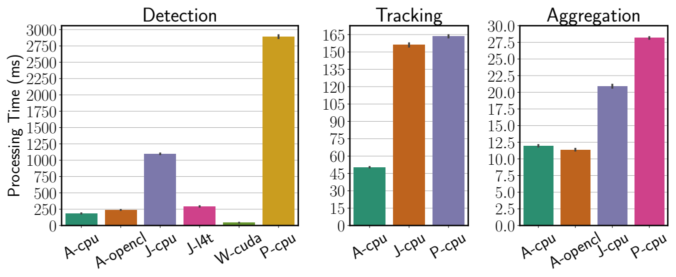
(3) Typical edge environments will consolidate various devices with different configurations, such as Raspberry PIs, BeagleBone Boards, Nvidia Jetson accelerators, Google Edge TPUs, etc. An ideal workload should be able to exploit the entire computational potential of such hardwares by supporting accelerators and instruction set optimizations.

4.1 Architecture

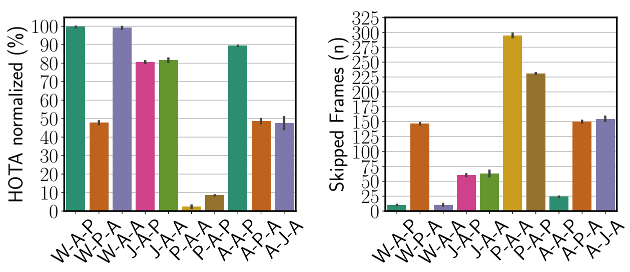
ComB's workload is a video analytics pipeline that consists of four independent microservices, each responsible for a specific task (see Figure 2) The services interconnect with each other to form a "pipe", and the inter-communication is over well-defined gRPC [17] interfaces.

The **video source** ① takes a video input (e.g. video file or webcam) and ingests it as a RTSP/RTP stream into the pipeline. By default, we utilize an h264 encoded video from MOTChallenge [12] dataset but developers can easily plug video feeds from real IP cameras as input. The implementation is based on gstreamer and supports multiple video sources. The **video aggregation service** ② reads video frames from the source and performs pre-processing for the following services in the pipeline (e.g., frame resizing for detection). Frames are then forwarded to either the object detection or object tracking, with the former path only utilized periodically at fixed frame intervals. We draw inspiration from next-generation applications (e.g., autonomous driving), where timely processing is critical for performance [29]. The frames that arrive faster than the underlying hardware capabilities are skipped – negatively affecting tracking performance. This performance indicator is especially useful for constrained edge infrastructures where slow execution times are expected. The implementation utilizes OpenCV [2] and supports both CPU and OpenCL-accelerated execution.

The **objects detection service** ③ utilizes a convolutional neural net (CNN) to detect objects in the received video frame. By default, we utilize a pre-trained YOLOv3 network using OpenCV's dnn module. The module supports common neural network formats and allows for future integration of other neural nets for evolved use cases (e.g. YOLOX for an improved detection). In the current ComB version, the detection service supports CPU, OpenCL-accelerated and CUDA-accelerated execution. However, we plan to integrate more backends (e.g. OpenVINO) in future iterations. The **object tracking service** ④ receives input from the detection service and initializes tracker instances for each newly detected object. The trackers are then updated for each frame received directly from



(a) Service Performance (Hardware-Execution environment)



(b) Pipeline Performance (Detection-Tracking-Aggregation)

Figure 3: Benchmark Output (W: Workstation, A: APU, J: Jetson, P: Raspberry Pi)

the aggregation service. The service integrates a naive IoU-based identification scheme that associates existing trackers with objects from the detection service. We use OpenCV's implementation of different tracking algorithms with KCF as the default method.

To resemble applications deployed in multi-tenant environments that run multiple, different service alongside each other, we incorporate Docker containers [14] as virtualization technology. Containers are the most common form of application packaging in modern applications and are essentially a good fit for constrained devices available at the edge. Additionally, we use a verbose image naming scheme to address different different docker image variants of each microservice with explicit hardware support compiled into the service binaries – allowing us to onload the most suitable container image based on the edge device hardware configuration.

5 EVALUATION

5.1 Experimental Setup

We evaluate ComB on a heterogeneous, edge-like testbed consisting of two Raspberry PI 4s with 4GB of RAM and an ARM Cortex A72 quad-core processor; one Nvidia Jetson AGX Xavier Developer Board with octa-core ARMv8 processor, 32GB of RAM, and a 512 core Volta GPU; one Fujitsu small form factor PC with a six-core Intel 8400T x86 APU and 8GB of RAM; and a workstation PC with a six-core Ryzen 2600x processor, 16GB RAM, and an Nvidia RTX 2070 GPU. We interconnect all devices over both WiFi and ethernet. We perform several iterations of ComB runs over multiple days. All runs were with a fixed 5 fps input framerate and utilized the Darknet based YOLOv3 CNN pre-trained on the COCO dataset. We only benchmark the GPU capabilities of the workstation and use the CPU to execute the benchmark orchestrator and the metric system.

5.2 Edge Accelerator Performance

An essential aspect of ComB is to highlight the impact of different accelerator hardware on service performance. Figure 3a shows the processing time for the aggregation, detection, and tracking service deployed on all hardware-execution environment combinations. The detection service paints a complete picture since our testbed supports all technology combinations. As expected, the workstation GPU performs the fastest with a mean processing time of 42 ms, followed closely by Intel APU using CPU with 184 ms. Interestingly, we find that Jetson using GPU (J-l4t) performs at par with the Intel APU using OpenCL, disproving the assumed notion behind the highly optimized performance of edge accelerators. On the other hand, we confirm that the ARM-based CPUs are unsuitable for CNN tasks, with the Jetson-CPU and Raspberry Pi 4 achieving processing time in seconds scale.

The aggregation and tracking services support fewer accelerators but perform comparably to the detection service. The Intel APU achieves the best performance taking 50 ms (tracking) and 12 ms (aggregation) processing time, with OpenCL acceleration granting a minimal boost of 0.5ms. Similar to the detection, the ARM-based devices perform noticeably slower with 156ms/21ms (Jetson) and 164ms/28ms (Pi) for tracking and aggregation, respectively. Further investigation reveals that both services are primarily single-threaded, which is the default OpenCV behavior over Python and is, therefore, inherited by applications relying on the library.

5.3 Benchmark and Workload Performance

Figure 3b shows the end-to-end performance of the video analytics pipeline (as tracking accuracy) in different hardware mappings. We use HOTA [25] as well as the number of skipped frames as our key performance indicators. Note that higher HOTA values correlate to higher tracking accuracy, while higher skipped frames lead to lower performance. The pipeline achieves the best tracking accuracy for schedules that utilize the workstation's GPU for the detection and the APU for tracking. We find that the aggregation service does not influence the quality-of-service (QoS) as all resources performed sufficiently fast. The slower detection at both the APU and the Jetson increases the number of skipped frames to 24 and 60 respectively, which decreases the end-to-end accuracy – highlighting the sensitivity of ComB to differentiate between configurations. Similarly, we observe that scheduling the detection on the Raspberry Pis causes a drastic drop in accuracy, and executing the tracking on either the Pi or the Jetson significantly increases the number of skipped frames (and, therefore, accuracy). We also experiment with network connections between resources and find that WiFi leads to additional skipped frames and higher variance in pipeline performance compared to wired (not shown due to space restriction). We also find that our runs over multiple days produce very similar results (as evident by the minimal deviation in our results in fig. 3), highlighting the repeatability of ComB. We also evaluate ComB on a large, homogeneous, compute cluster with resource configurations resembling cloudlets. We found the performance results similar to all-CPU mapping in our edge testbed, which is why we leave it from the discussion for brevity.

While we carefully implement ComB to avoid process blocking, we also evaluate the overhead of our benchmark for completeness. We integrate a CPU profiler into our workload to measure the time taken by each function call in each service. We find that the services spend a majority of their time on functions relevant to the pipeline execution for all configurations (not shown due to brevity) – indicating a minimal footprint of ComB.

6 CONCLUSION

In this work, we presented ComB, which is benchmarking suite specifically designed for edge infrastructures and workloads. The innovation lies in the decoupled orchestration and workload design – allowing developers to customize, extend (or entirely swap) the workload to better complement the target application. The default workload of ComB is a distributed video analytics application, which includes multiple microservices with distinct operational characteristics and requirements. We designed ComB to be highly extensible and future-proof as novel virtualization and execution technologies can be incorporated into the system with minimal implementation overhead. Our evaluation showed that ComB is sensitive to detecting inconsistencies in hardware operation and provides valuable runtime metrics beyond the capacity (and configuration) to resource operators. We make ComB public and open-source for community contributions. In future iterations of ComB, we want to integrate better object detection and object tracking procedures in the default workload to significantly improve performance accuracy. Furthermore, we would like to incorporate workloads compatible with lightweight virtualization technologies (e.g., Unikernels) and specialized execution environments, such as TEEs.

REFERENCES

[1] 2022. K3s - Lightweight Kubernetes. https://github.com/k3s-io/k3s

[2] OpenCV 2022. *Opencv*. OpenCV. https://github.com/opencv/opencv

[3] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. 2018. Mobile Edge Computing: A Survey. *IEEE Internet of Things Journal* 5, 1 (2018), 450–465. https://doi.org/10.1109/JIOT.2017.2750180

[4] Amazon Web Services, Inc. 2021. *AWS IoT Greengrass*. https://aws.amazon.com/greengrass/

[5] Amazon Web Services, Inc. 2021. *Firecracker MicroVM*. https://github.com/firecracker-microvm/firecracker

[6] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. 2017. Real-Time Video Analytics: The Killer App for Edge Computing. *Computer* 50, 10 (2017), 58–67. https://doi.org/10.1109/MC.2017.3641638

[7] Lorenzo Corneo, Maximilian Eder, Nitinder Mohan, Aleksandr Zavodovski, Suzan Bayhan, Walter Wong, Per Gunningberg, Jussi Kangasharju, and Jörg Ott. 2021. Surrounded by the Clouds: A Comprehensive Cloud Reachability Study. In *Proceedings of the Web Conference 2021* (Ljubljana, Slovenia) *(WWW '21)*. Association for Computing Machinery, New York, NY, USA, 295–304. https://doi.org/10.1145/3442381.3449854

[8] Lorenzo Corneo, Nitinder Mohan, Aleksandr Zavodovski, Walter Wong, Christian Rohner, Per Gunningberg, and Jussi Kangasharju. 2021. (How Much) Can Edge Computing Change Network Latency?. In *2021 IFIP Networking Conference (IFIP Networking)*. 1–9. https://doi.org/10.23919/IFIPNetworking52078.2021.9472847

[9] Vittorio Cozzolino, Leonardo Tonetto, Nitinder Mohan, Aaron Yi Ding, and Jorg Ott. 2022. Nimbus: Towards Latency-Energy Efficient Task Offloading for AR Services. *IEEE Transactions on Cloud Computing* (2022), 1–1. https://doi.org/10.1109/TCC.2022.3146615

[10] The Khang Dang, Nitinder Mohan, Lorenzo Corneo, Aleksandr Zavodovski, Jörg Ott, and Jussi Kangasharju. 2021. Cloudy with a Chance of Short RTTs: Analyzing Cloud Connectivity in the Internet. In *Proceedings of the 21st ACM Internet Measurement Conference* (Virtual Event) *(IMC '21)*. Association for Computing Machinery, New York, NY, USA, 62–79. https://doi.org/10.1145/3487552.3487854

[11] Anirban Das, Stacy Patterson, and Mike Wittie. 2018. EdgeBench: Benchmarking Edge Computing Platforms. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. 175–180. https://doi.org/10.1109/UCC-Companion.2018.00053

[12] Patrick Dendorfer, Hamid Rezatofighi, Anton Milan, Javen Shi, Daniel Cremers, Ian Reid, Stefan Roth, Konrad Schindler, and Laura Leal-Taixé. 2020. MOT20: A Benchmark for Multi Object Tracking in Crowded Scenes. *arXiv:2003.09003 [cs]* (March 2020). arXiv:2003.09003 [cs]

[13] Gianmarco Dinelli, Gabriele Meoni, Emilio Rapuano, Gionata Benelli, Luca Fanucci, and Martin Margala. 2019. An FPGA-Based Hardware Accelerator for CNNs Using On-Chip Memories Only: Design and Benchmarking with Intel Movidius Neural Compute Stick. *Int. J. Reconfig. Comput.* 2019 (jan 2019), 13 pages. https://doi.org/10.1155/2019/7218758

[14] Docker Inc. [n.d.]. Empowering App Development for Developers | Docker. https://www.docker.com/.

[15] Leo Eichhorn, Tanya Shreedhar, Aleksandr Zavodovski, and Nitinder Mohan. 2021. Distributed Ledgers for Distributed Edge: Are We There Yet? *(IWCI'21)*. Association for Computing Machinery, New York, NY, USA, 26–33. https://doi.org/10.1145/3488663.3493687

[16] Shilpa George, Thomas Eiszler, Roger Iyengar, Haithem Turki, Ziqiang Feng, Junjue Wang, Padmanabhan Pillai, and Mahadev Satyanarayanan. 2020. OpenRTiST: End-to-End Benchmarking for Edge Computing. *IEEE Pervasive Computing* 19, 4 (2020), 10–18. https://doi.org/10.1109/MPRV.2020.3028781

[17] gRPC Authors. 2021. *gRPC*. https://grpc.io/

[18] John E. Hopcroft and Richard M. Karp. 1973. An $n\hat{5/2} $ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM J. Comput.* 2, 4 (1973), 225–231. https://doi.org/10.1137/0202019 arXiv:https://doi.org/10.1137/0202019

[19] Daniel Kang. 2021. *The Tengo Language*. https://github.com/d5/tengo

[20] C. P. Kruger and G. P. Hancke. 2014. Benchmarking Internet of things devices. In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*. 611–616. https://doi.org/10.1109/INDIN.2014.6945583

[21] H. W. Kuhn. 1955. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly* 2, 1-2 (1955), 83–97. https://doi.org/10.1002/nav.3800020109

[22] Chien-I Lee, Meng-Yao Lin, Chia-Lin Yang, and Yen-Kuang Chen. 2019. Iotbench: A Benchmark Suite for Intelligent Internet of Things Edge Devices. In *2019 IEEE International Conference on Image Processing (ICIP)*. 170–174. https://doi.org/10.1109/ICIP.2019.8802949

[23] Luke Lockhart, Paul Harvey, Pierre Imai, Peter Willis, and Blesson Varghese. 2020. Scission: Performance-driven and Context-aware Cloud-Edge Distribution of Deep Neural Networks. (2020), 257–268. https://doi.org/10.1109/UCC48980.2020.00044

[24] Jonathon Luiten and Arne Hoffhues. 2020. TrackEval. https://github.com/JonathonLuiten/TrackEval.

[25] Jonathon Luiten, Aljosa Osep, Patrick Dendorfer, Philip Torr, Andreas Geiger, Laura Leal-Taixé, and Bastian Leibe. 2020. HOTA: A Higher Order Metric for Evaluating Multi-Object Tracking. *International Journal of Computer Vision* (2020), 1–31. https://doi.org/10.1007/s11263-020-01375-2

[26] Jonathan McChesney, Nan Wang, Ashish Tanwer, Eyal de Lara, and Blesson Varghese. 2019. DeFog: Fog Computing Benchmarks *(SEC '19)*. Association for Computing Machinery, New York, NY, USA, 47–58. https://doi.org/10.1145/3318216.3363299

[27] Larry W McVoy, Carl Staelin, et al. 1996. lmbench: Portable Tools for Performance Analysis.. In *USENIX annual technical conference*. San Diego, CA, USA, 279–294.

[28] Microsoft Corporation. 2021. *IoT Edge*. https://azure.microsoft.com/en-us/services/iot-edge/

[29] Nitinder Mohan, Lorenzo Corneo, Aleksandr Zavodovski, Suzan Bayhan, Walter Wong, and Jussi Kangasharju. 2020. Pruning Edge Research with Latency Shears *(HotNets '20)*. Association for Computing Machinery, New York, NY, USA, 182–189. https://doi.org/10.1145/3422604.3425943

[30] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. 2019. Survey and Benchmarking of Machine Learning Accelerators. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–9. https://doi.org/10.1109/HPEC.2019.8916327

[31] Anshu Shukla, Shilpa Chaturvedi, and Yogesh Simmhan. 2017. RIoTBench: An IoT Benchmark for Distributed Stream Processing Systems. *Concurrency and Computation: Practice and Experience* 29, 21 (2017), e4257. https://doi.org/10.1002/cpe.4257

[32] Qirui Yang, Runyu Jin, Nabil Gandhi, Xiongzi Ge, Hoda Aghaei Khouzani, and Ming Zhao. 2020. EdgeBench: A Workflow-based Benchmark for Edge Computing. *arXiv:2010.14027 [cs]* (Oct. 2020). arXiv:2010.14027 [cs]

[33] Aleksandr Zavodovski, Nitinder Mohan, Suzan Bayhan, Walter Wong, and Jussi Kangasharju. 2019. ExEC: Elastic Extensible Edge Cloud *(EdgeSys '19)*. Association for Computing Machinery, New York, NY, USA, 24–29. https://doi.org/10.1145/3301418.3313941