applied sciences

MDPI

Article

Optimizing Data Processing: A Comparative Study of Big Data Platforms in Edge, Fog, and Cloud Layers

Thanda Shwe * and Masayoshi Aritsugi *

Faculty of Advanced Science and Technology, Kumamoto University, Kumamoto 860-8555, Japan
* Correspondence: thandashwe@cs.kumamoto-u.ac.jp (T.S.); aritsugi@cs.kumamoto-u.ac.jp (M.A.)

Abstract: Intelligent applications in several areas increasingly rely on big data solutions to improve their efficiency, but the processing and management of big data incur high costs. Although cloud-computing-based big data management and processing offer a promising solution to provide scalable and abundant resources, the current cloud-based big data management platforms do not properly address the high latency, privacy, and bandwidth consumption challenges that arise when sending large volumes of user data to the cloud. Computing in the edge and fog layers is quickly emerging as an extension of cloud computing used to reduce latency and bandwidth consumption, resulting in some of the processing tasks being performed in edge/fog-layer devices. Although these devices are resource-constrained, recent increases in resource capacity provide the potential for collaborative big data processing. We investigated the deployment of data processing platforms based on three different computing paradigms, namely batch processing, stream processing, and function processing, by aggregating the processing power from a diverse set of nodes in the local area. Herein, we demonstrate the efficacy and viability of edge-/fog-layer big data processing across a variety of real-world applications and in comparison to the cloud-native approach in terms of performance.

Keywords: big data; computing platforms; IoT; serverless

Citation: Shwe, T.; Aritsugi, M. Optimizing Data Processing: A Comparative Study of Big Data Platforms in Edge, Fog, and Cloud Layers. *Appl. Sci.* **2024**, *14*, 452. https://doi.org/10.3390/app14010452

Academic Editor: Alexander N. Pisarchik

Received: 20 November 2023
Revised: 26 December 2023
Accepted: 2 January 2024
Published: 4 January 2024

1. Introduction

In the last few years, a new class of applications that use unprecedented amounts of data generated from mobile devices and Internet of Things (IoT) sensors has been widely deployed in many areas, bringing better quality of life to humans through the automation of daily tasks and enabling time and energy savings, as well as monitoring, efficient communication, better decision making, etc. [1–3]. This type of application, which is present in various domains, such as healthcare monitoring, smart cities, industry, and transportation [4], not only involves performing intensive computation on large amounts of sensor data but also occasionally requires the output to be processed in real time to provide faster interactions and better user experiences. To provide the necessary computing resources, such as virtual machines and storage resources, including object-based storage and databases for these types of applications, cloud solutions are considered, as they are capable of storing and processing data in the cloud and sending the results back to the IoT devices [5,6]. Most existing systems use big data management platforms, which deal with large amounts of big data processing in the cloud computing platform or in self-hosted, dedicated clusters or data centers [7,8]. However, some IoT applications, such as smart cities and machinery automation, are real-time applications, and the adoption of cloud computing for such applications can result in high latency because sensor data need to be transferred to a cloud data center in a remote location. Thus, cloud solutions may no longer be appropriate for these types of applications due to the limitations caused by network bandwidth consumption and latency constraints [9].

Another noteworthy issue with cloud-based processing is that all users' data tend to be sent to the cloud. Specifically, there is potential for security and privacy issues and

an increase in the communication payload and costs. The storage of sensitive data in the cloud raises concerns about data vulnerability. In a centralized and shared environment, there is a risk of unauthorized access or data breaches. Regarding data privacy, the geographic locations of cloud servers may impact data privacy. Navigating international data protection laws and ensuring data residency compliance becomes a complex task in terms of protecting sensitive information. In addition, transferring large volumes of data to and from the cloud can incur significant costs. Organizations must carefully manage data transfer to reduce their expenses. These issues are particularly relevant in the context of data processing applications, such as healthcare and sensitive data handling, where the importance of security and low latency necessitates more focused consideration [10–14]. Concerns about sensitive information being exposed to remote cloud data centers highlight the limitations of cloud-based processing.

To overcome this challenge, fog and edge computing, which move the processing and storage tasks to locations that are closer to the data source, have been introduced as complements to cloud computing to provide storage and computing capabilities [9,15,16]. In these types of computing, any local device, however small, is considered capable of some processing tasks. Edge/fog computing helps users to perform data processing at lower latency and in the most appropriate way. At present, edge- and fog-layer devices are only used for some preprocessing tasks, such as filtering, feature processing, and compression. Computationally intensive tasks such as big data analytics and machine learning training are sent out to the cloud, as they cannot be efficiently run on low-performance local edge and fog devices [17,18]. Data processing in a layer architecture allows real-time and lightweight tasks processed on edge devices that need high storage capacities and processing capabilities to be moved to the cloud. However, important drawbacks related to the high bandwidth consumption involved in sending large quantities of data between local devices and cloud providers emerge. If the data can be stored and processed locally, closer to the IoT devices, the data transmitted through the network would be significantly reduced.

We can observe a recent breakthrough in edge- and fog-layer computing devices as the processing abilities of these devices have become increasingly efficient and capable [16]. In addition, most of the data or applications pass through the edge/fog-layer devices as a gateway before connecting to the cloud. Edge/fog computing is potentially a good solution for data processing problems, thanks to its recently developed resource-scaling capabilities. This provides the possibility of deploying edge- and fog-layer big data management platforms using local edge/fog devices as computing and storage nodes. Such a big data management platform can not only help to reduce latency, bandwidth consumption, and cloud budgets but can also be applied in some environments without an Internet connection.

IoT applications that span various domains, ranging from smart cities to industrial environments and healthcare systems, continuously produce massive volumes of data, including sensor readings, images, videos, and other types of information. Data processing platforms handle large volumes of diverse data, making them well-suited for the processing of the varied data generated by IoT sources. IoT deployments often involve distributed environments with numerous devices spread across different locations. The distributed computing models of big data processing allow them to efficiently process data across distributed clusters, accommodating the distributed nature of IoT deployments. As many IoT applications demand real-time or near-real-time processing to enable swift responses and decision making, centralized data management approaches encounter challenges in the efficient handling of information. Thus, by strategically integrating computational capabilities closer to the data source, within the edge and fog layers, it is possible to facilitate local data processing, reducing the necessity of transmitting all data to centralized cloud infrastructure. This not only addresses latency concerns but also optimizes bandwidth usage, making it particularly beneficial in scenarios where network resources are limited. Furthermore, the decentralized nature of edge/fog computing contributes to enhanced security and privacy by allowing sensitive data to be processed closer to their origin. In

summary, the cooperative interconnection between IoT applications, data management, and edge/fog computing represents a strategic approach to meet the evolving demands of efficient, real-time, and secure data processing within the dynamic landscape of the Internet of Things.

In addition, the requirements of big data processing, such as the training of machine learning models [19–21], stream processing [22], and event processing [23,24], raise scalability issues and require significant computing and storage resources [25,26]. Fortunately, big data management and processing platforms such as Apache Spark [27], Apache Flink [28], and Apache OpenWhisk [29] address these concerns by facilitating distribution and collaboration. In such systems, big data processing tasks are performed using the collaborative power of nodes in clusters. With recent increases in the resource capacity of edge/fog devices and the capability of big data processing platforms, it has become increasingly important to explore the deployment of big data processing platforms on resource-constrained edge/fog devices. The opportunity to combine recent advances in edge/fog devices' computing capabilities and big data processing platforms motivates the work described in this paper. To address data processing across all computing environments, while previous works have focused on specific data processing paradigms (batch, stream, or function processing) [30–36] and the capabilities of resource-constrained devices [37–39], our research addresses the need for a comprehensive examination of the performance of these platforms across three paradigms: batch processing, stream processing, and function processing. This comprehensive approach, coupled with the inclusion of various applications, such as image classification, object detection, and image resizing, adds a layer of comprehensiveness to the evaluation. This work helps to elucidate the efficiency gains and potential benefits that the collaborative deployment of local computing resources across different computing layers can offer in the field of big data processing. The specific research gap that our work addresses is the need for a comprehensive understanding of big data processing in diverse computing environments, considering different paradigms and applications.

In this paper, we ask the following question: Can edge- or fog-layer devices host big data management platforms for data science applications, such as image classification, object detection, and event-based image processing? To answer this question, we present an extensive comparison between edge-/fog-layer big data management platforms and other cloud-based solutions regarding their performance. We evaluate three big data management and processing platforms—Apache Spark, Apache Flink, and Apache OpenWhisk—based on three different computing paradigms, namely batch processing, stream processing, and function processing, as demonstrated in Figure 1. Real-world applications such as image classification, object detection, and image processing are used to evaluate the big data platforms in edge, fog, and cloud layers. To the best of our knowledge, this is the first study that not only examines the performance of different big data processing platforms for different types of applications in resource-constrained environments but also comprehensively explores their deployment in computing layers, namely edge, fog, and cloud computing layers, using real-world applications.

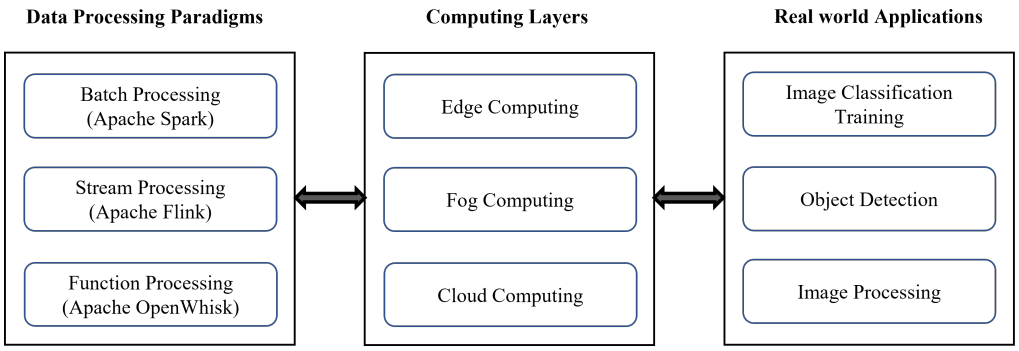

Figure 1. Overall organization of data processing platform deployment.

This study primarily focuses on conducting a thorough comparative and analytical analysis of big data platform deployment across edge, fog, and cloud computing layers rather than introducing new methods or technologies. We emphasize the evaluation of edge/fog computing big data processing platforms and aim to offer practical recommendations derived from experimental findings. While this work may not present new methods, its significance lies in providing valuable insights into the comparative efficiency and applicability of big data processing platforms across different computing layers. The comprehensive nature of this study, covering three computing paradigms across various layers with real-world applications and detailed experimental findings, contributes practical guidance for decision making in big data platform deployment.

In summary, we make the following contributions in this paper.

- We conduct an extensive performance comparison of the edge-/fog-based processing of big data management frameworks against other cloud-based solutions. Our analysis covers the deployment of three different big data platforms based on three data processing applications.
- We present interesting findings that suggest that an edge- and fog-based big data management platform is a viable and promising option for applications where local processing power of local data is desired.
- We further explore the impact of the deployment of edge- and fog-based big data management frameworks and present recommendations regarding how to better use big data management frameworks for data science applications. In addition, we discuss the research challenges and opportunities in building a more practical edge-/fog-based big data management system.

The remainder of this paper is structured as follows. Section 2 summarizes some related studies. Section 3 explains the details of our methodology for the evaluation of the performance of big data management platforms. Section 4 describes the experiments that we carried out and the results of the experiments. Section 5 presents the discussion and recommendations based on the implications and results of our experiments. Finally, our conclusions and suggestions for future work are included in Section 6.

2. Related Work

The related work is split into three categories. First, we review works that investigate the capability and performance of resource-constrained single-board computers. Then, we focus on studies that propose design improvements for big data processing platforms that cater to resource-constrained edge devices. Afterward, we consider works that perform comprehensive and comparative benchmark studies of big data processing platforms.

2.1. Capability of Resource-Constrained Devices

Due to the increasing capabilities of low-cost, single-board computers and their benefits in terms of cost, low power consumption, small size, and reasonable performance, several works [37–39] have studied the capabilities and performance of low-cost edge-layer devices for deep learning and machine learning interference. The works reported in [37,38] focused on the inference of deep learning models in various low-cost devices, such as the Raspberry Pi 4, Google Coral Dev Board, and Nvidia Jetson Nano, and evaluated their performance in terms of the inference time and power consumption. Their main focus was to implement the inference parts of deep learning models in edge devices in order to achieve real-time processing. The authors of [39] conducted a comprehensive survey of design methodologies for AI edge development, emphasizing the importance of single-layer specialization and cross-layer codesign, which includes hardware and software components for edge training, inference, caching, and offloading. Several insights were highlighted with respect to the quality of AI solutions in the edge computing layer. While the deployment of low-cost edge devices is still a relatively new paradigm for advanced applications, it has found widespread interest, particularly in function processing. Some approaches [40–44] have been developed for edge-layer serverless platforms. Specifically,

all of these works attempted to develop edge-layer serverless platforms that can support edge AI applications. Moreover, increasing attention has been paid to the capabilities and viability of single-board computers; several works [45–47] have explored the characteristics and possibilities of deploying different big data applications in resource-constrained environments. In contrast to our work, these works did not investigate the deployment of big data processing platforms using the collaborative power of local nodes in edge, fog, and cloud layers.

2.2. Improvement of Big Data Processing Platforms for Edge Devices

Some previous studies have evaluated data processing platforms under different application scenarios and proposed design changes to off-the-shelf software platforms to cater to the requirements of applications in edge/fog computing layers. In [30], the authors proposed a serverless edge platform based on the OpenWhisk serverless platform to address the challenges of real-time and data-intensive applications in fog/edge computing. The proposed platform comprised additional components for latency-sensitive computation offloading, stateful partitions, and real-time edge coordination. The platform was evaluated in terms of its resource utilization footprint, latency overhead, throughput, and scalability under different application scenarios. The results show that the serverless architecture reduced the burden of infrastructure management, allowed greater functionality to be deployed on fog nodes with limited resources, and fulfilled the requirements of different application scenarios and the heterogeneous deployment of fog nodes. To optimize stream processing in the edge/fog environment, the authors of [31] proposed Amnis, a stream processing framework that considers computational and network resources at the edge. It extended the Storm framework in terms of stream processing operator allocation and placement for stream queries. Compared to the default operator scheduler in Apache Storm, it performed better in terms of end-to-end latency and overall throughput. These works targeted only individual data processing paradigms, such as batch processing, stream processing, or function processing. Our work differs by investigating big data processing platforms for all three of these computing paradigms, namely batch processing, stream processing, and function processing, based on various applications across the computing layers.

2.3. Comparative Studies of Big Data Processing Platforms

Some previous works have included benchmark studies of the performance of stream processing systems, such as [32], which evaluated the performance of three stream processing platforms, namely Apache Storm, Apache Spark, and Apache Flink, in terms of throughput and latency. In [34], these aspects were also evaluated for Apache Spark and Apache Flink. With respect to batch processing platforms, the work reported in [33,34] included a comprehensive study of two widely used big data analytics tools, namely Apache Spark [27] and Hadoop MapReduce [48], on a common data mining task, i.e., classification. With respect to function processing platforms, the work reported in [35,36] provided a benchmarking framework for the characterization of serverless platforms in both commercial cloud and open-source platforms. The work that is most related to ours is [49], which evaluated computing resources across the computing continuum using three applications: video encoding, machine learning, and in-memory analytics. It also provided recommendations based on the evaluation results regarding where to perform the tasks across the computing continuum. The authors utilized a real test bed named the Carinthian Computing Continuum (C 3) to extend cloud data centers with low-cost devices located close to the edge of the network. They recommended offloading the applications to edge and fog resources to reduce the network traffic and CO_2 emissions, with an acceptable performance penalty, while the cloud was used for lower execution times. Another work [50] mainly evaluated big data processing possibilities on a Raspberry Pi-based Apache Spark and Hadoop Cluster and examined the impact on storage performance of employing three different external storage solutions. Then, the cluster performance was compared with

that of a single desktop PC using microbenchmarks such as Wordcount, TeraGen/TeraSort, TestDFSIO, and Pi computation.

However, neither of these works included a comprehensive investigation of the performance of existing big data processing platforms for the three computing paradigms, namely batch processing, stream processing, and function processing, based on various applications (e.g., image classification, object detection, image resizing). Therefore, exploring the operating performance of batch, stream, and function processing in the current edge, fog, and cloud layers is of great significance for research and industrial applications in the field of big data processing. We are particularly interested in the potential improvement in performance obtained by deploying big data platforms.

3. Target Data Processing and Computing

In this section, we first discuss the three data processing paradigms applied in most processing scenarios, such as IoT and big data processing. Then, we briefly describe computing in the edge/fog layer, which can bring benefits in terms of reducing the latency and bandwidth consumption when sending a large amount of data to the cloud.

3.1. Three Data Processing Paradigms

In the area of data processing, three basic data processing paradigms, namely batch processing, stream processing, and function processing, have become prominent, each offering distinct methodologies to handle information efficiently [51]. The first paradigm, batch processing, involves the systematic analysis and processing of data in predetermined, fixed-size sets. It is well-suited for tasks that can tolerate a degree of latency and operate on collected data. In contrast, stream processing is designed for real-time analysis, enabling the continuous processing of data as they flow, making it particularly suitable for applications requiring immediate insights or actions. The third paradigm, function processing, represents a paradigm shift toward event-driven computing, where discrete functions or microservices are triggered by specific events, offering a highly responsive and modular approach to data processing. These three paradigms offer a broad range of tools that can be used to handle a variety of data processing needs, from large-scale, periodic analyses to real-time, event-driven applications.

3.1.1. Batch Processing

Batch processing involves the intensive computation of high-volume, repetitive data processing tasks. Nowadays, data are generated from several sources, and there is a huge demand for the storage, processing, and querying of big data. The use cases for batch processing include machine learning, deep learning, big data analytics, and high-throughput data processing. Batch processing, which is designed for the handling of large volumes of data in situations where real-time processing is unnecessary, takes on a strategic role when deployed on collaborative local edge/fog devices. This deployment approach, represented by applications such as machine learning training, minimizes the need for extensive data transmission to centralized cloud servers. By harnessing the collective power of local devices, this strategy enhances data security and privacy while optimizing resource utilization. The MapReduce model and its implementations, such as Apache Hadoop [48] and Apache Spark [27], are the de facto solutions for big data batch processing. This approach to distributed computing provides a robust and scalable framework for the handling of large datasets by dividing complex processing tasks into manageable subtasks across a cluster of nodes. Apache Hadoop, an open-source software framework, pioneered the practical implementation of the MapReduce model, offering a distributed file system (Hadoop Distributed File System) and MapReduce programming. Similarly, Apache Spark emerged as a powerful alternative, introducing in-memory processing capabilities that significantly accelerate data processing workflows. These solutions offer platforms for the development, storage, and deployment of big data applications from large-scale analytics to machine learning applications. Commercial cloud providers such as Amazon EMR,

Google Cloud Dataproc, and Azure HDInsight offer services for batch processing. These de facto big data solutions can also be hosted on private clusters or a private cloud. Big data batch processing platforms are organized with the master–workers distributed computing architecture. It consists of one or more master nodes that are responsible for distributing tasks to the workers and managing the workers. The workers perform the actual data processing tasks assigned to them. Generally, the data are partitioned and distributed among the nodes, resulting in the parallel processing of data by the nodes.

Among the many big data platforms available for batch processing, we selected Apache Spark, which is an open-source, distributed computing system that has gained widespread popularity for the efficient processing of large-scale datasets. Its technological underpinnings are grounded in a resilient distributed dataset (RDD) abstraction, which allows data to be distributed across a cluster of machines while ensuring fault tolerance and parallel processing [27,52]. In the context of edge and fog computing, where resource constraints and network variability pose significant challenges, Apache Spark offers several features to overcome these obstacles. One notable feature is its ability to optimize data locality, minimizing data transfer across the network [53]. Furthermore, Spark's flexibility in terms of deployment in various types of infrastructure, including edge devices and fog nodes, makes it adaptable to diverse computing environments [46].

3.1.2. Stream Processing

Stream processing is becoming increasingly common as businesses nowadays need to process large amounts of data in real time. It helps to develop responsive applications and accelerate decision making. The most common and popular use cases for data stream processing include real-time data analytics, fraud detection, cyber security, stock market monitoring, healthcare monitoring systems, and sensor data processing. Big data stream processing platforms such as Apache Spark [27], Storm [54], Flink [28], and Kafka [55] are available for stream processing, and each of them is designed to efficiently handle the continuous flow of data from diverse sources. Apache Spark is not only proficient in batch processing but also possesses robust stream processing capabilities, making it a comprehensive choice for hybrid scenarios. Apache Storm specializes in real-time data processing, providing fault-tolerant and scalable solutions for streaming applications. Apache Flink stands out with its sophisticated event-time processing and state management features, which are ideal for scenarios requiring the precise handling of event sequences. Kafka, while primarily recognized as a distributed event streaming platform, complements these frameworks by serving as a distributed messaging system, facilitating the seamless integration and transport of streaming data. In addition, all the commercial and public cloud providers provide services for stream processing application development, such as Amazon Kinesis, Azure Stream Analytics, and Google Cloud Dataflow. Streaming systems process data as soon as they arrive and produce the processing results shortly afterward. In other words, streaming processing provides data management tasks to consume, process, and produce data securely and reliably. Stream processing starts by consuming data from an integrated data source, performing single or multiple stream processing actions on them, then publishing the results back to the data sink.

Among stream processing platforms, we selected Apache Flink to deploy in edge, fog, and cloud layers. Apache Flink is a powerful, open-source stream processing framework designed to efficiently process large-scale data with low latency and high throughput. Its technological underpinnings revolve around a distributed data stream processing model, providing fault tolerance and stateful processing capabilities. In the context of edge and fog computing, where resource constraints and network challenges are significant, Apache Flink offers support for event-time processing, which enables accurate handling of events across distributed and potentially unreliable environments [28]. This is important for applications at the edge, where time-sensitive data processing is essential. Furthermore, in the case of scenarios where intermittent connectivity or device failures may occur, Flink's

state management ensures resilience and consistency, overcoming challenges related to network variability and device reliability [56].

3.1.3. Function Processing

The serverless computing model has emerged as a subset of the cloud computing model in which the cloud service provider takes responsibility for managing underlying servers and the scaling of applications, while the developers can build and run functions on demand. The most common use cases for serverless computing include event-driven IoT applications, data analytics, and machine learning inferences. The main cloud service providers have services for function processing, such as Amazon Lambda, Google Cloud Functions, IBM Cloud Functions, and Azure Functions. Moreover, some open-source serverless platforms, such as Apache OpenWhisk [29], OpenFaaS [57], and KNative [58], are also available for deployment in private environments. Apache OpenWhisk, with its event-driven programming model, allows for the execution of functions in response to events, offering a scalable and modular approach to serverless computing. OpenFaaS, known for its simplicity and ease of deployment, extends the capabilities of serverless platforms to private environments, enabling users to efficiently build, deploy, and scale functions. KNative extends Kubernetes to provide a set of building blocks for serverless applications. In serverless computing, developers write their business logic code as a set of functions. Each function performs a specific task when it is called by a function invocation event, such as image uploading, an HTTP request, an incoming message, or database state changes. These functions are deployed in serverless computing platforms along with their triggers and function invocation methods. When a function is invoked, it is executed on the running server. The execution process is abstracted from the user/developer and only returns the invocation results to the user.

We chose Apache OpenWhisk for function processing, which is an open-source serverless computing platform designed to execute functions in response to events, offering a flexible and scalable approach to distributed computing. In the context of edge and fog computing, where resource constraints are common, OpenWhisk's serverless architecture allows for efficient and dynamic resource utilization. This is particularly beneficial in scenarios where devices at the edge have limited computational capabilities. Furthermore, OpenWhisk's ability to break down applications into small, independent functions aligns with the requirements of edge and fog computing [29]. This modular approach allows for the deployment of lightweight functions to edge devices, ensuring that computational tasks are distributed optimally. The event-driven nature of OpenWhisk makes it well-suited for real-time processing, which is crucial in edge and fog computing applications [59]. By responding to events in near-real-time, OpenWhisk can efficiently handle tasks that require immediate responsiveness.

3.2. Computing in the Edge/Fog Layer

Computing in the edge/fog layer has been introduced to overcome the limitations imposed by centralized cloud computing by performing processing as close to the data source as possible, resulting in a reduction in the round-trip latency when sending data to the cloud and returning the results to IoT devices. It plays an important role in addressing the challenges of real-time and near-real-time data processing and analysis in Internet of Things (IoT) applications. An overview of the layered architecture of edge–fog–cloud computing is shown in Figure 2.

Edge/fog devices can be considered local or gateway devices that are close to the data source; in some cases, they can be devices at one or two hops along the network to the cloud. Edge devices such as IoT sensors, gateways, and routers execute computations locally, making rapid decisions and providing immediate responses. Applications that need more substantial local processing, data aggregation, and advanced analytics are ideally suited for fog computing. It facilitates the interpretation of data in real time, enabling quicker insights and decisions. While edge computing is best for quick, localized decision

making, as it is located closer to the data source, fog computing bridges the gap between the edge and the centralized cloud by enabling more sophisticated processing, aggregation, and analysis. Generally, fog computing has greater processing power and storage capacity than edge computing.

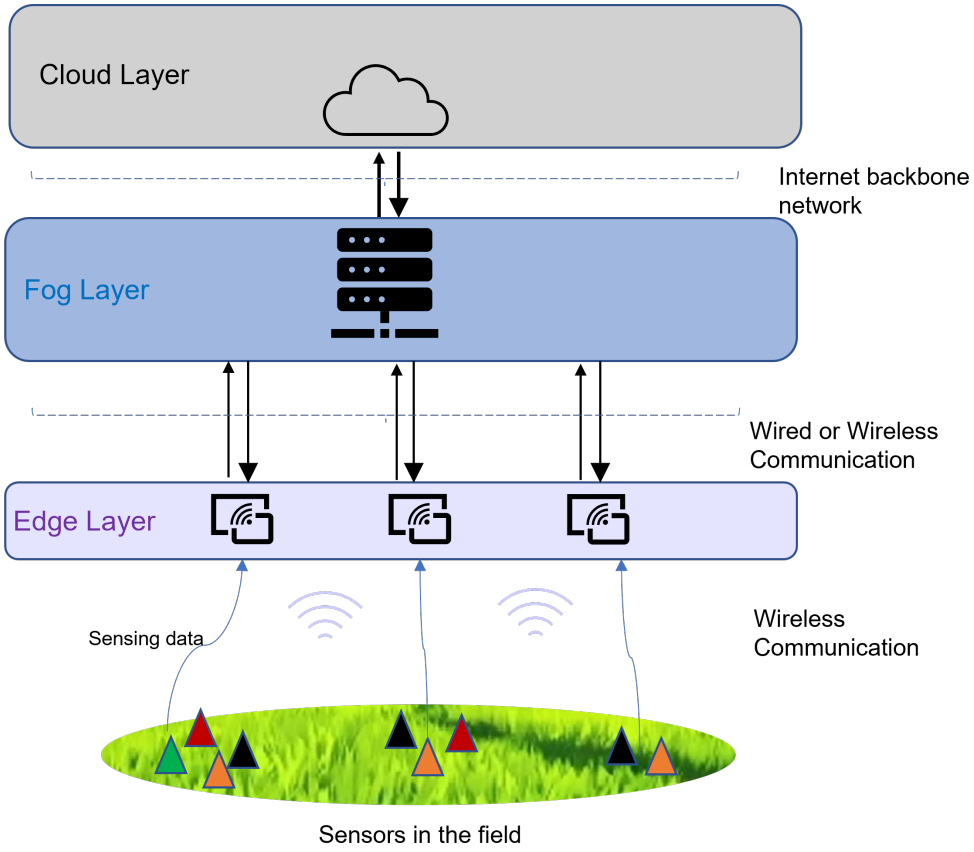

Figure 2. Overview of edge/fog computing layers.

Initially, fog/edge computing is used as a complement to cloud computing. Small computing tasks and preprocessing tasks that are lightweight are usually executed in this layer before being delegated to the cloud layer. However, the recent increase in the resource capability of low-cost devices offers the potential to use resources as much as possible and provides capabilities for local devices. This can lead to significant achievements in terms of reducing the latency, bandwidth consumption, and the network cost involved in sending large amounts of data to the cloud. As data travel from their point of origin (e.g., sensors) to applications, they pass through many computing devices, each of which is a potential target of computation. By keeping sensitive data localized and lowering the likelihood of data breaches, edge/fog computing improves the security and privacy. In order to ensure that devices can function independently when not connected to the Internet, it also provides offline functionalities.

Big data platforms play a crucial role in edge and fog computing, offering scalable and efficient solutions to process and analyze vast amounts of data closer to their source. In summary, big data platforms optimized for edge and fog computing environments offer proximity, reduced latency, efficient resource utilization, scalability, flexibility, and support for real-time decision making.

4. Experiments and Results

4.1. Experimental Methodology

We use real-world applications to understand and compare the performance of big data processing platforms in each computing layer. Specifically, we use image classification

as batch processing, object detection as stream processing, and image processing as function processing. Although our experimental design is centered around image-related applications within three fundamental data processing paradigms, it is applicable to a broad spectrum of use cases. The image-related applications serve as representative examples to showcase the platforms' performance across diverse scenarios. The findings and recommendations derived from our study are generalizable to a variety of applications within the selected data processing paradigms. In the following subsection, we present (a) detailed description of the experimental environment, including hardware configurations, software versions, and any relevant settings; (b) the tools, frameworks, and techniques utilized during the research; and (c) the metrics used to evaluate the performance of real-world applications deployed on big data processing platforms.

4.1.1. Image Classification

Over the years, machine learning and deep learning have been increasingly used for classification and detection problems that include training and inference tasks. Due to the intensive resource usage of the training task, it is generally performed on high-performance computers and GPU servers. However, training on edge devices is gaining importance attracting attention in various applications, such as edge AI, IoT, and autonomous systems, as it enables real-time processing, reduced latency, and enhanced privacy by keeping data and computation local. The training of machine learning models at the edge involves dealing with resource constraints and customized algorithms that cater to edge devices [39]. In this work, we deploy image classification training on Apache Spark because machine learning and deep learning training are some of the scenarios associated with batch processing. Apache Spark ships with machine learning libraries but not for complicated deep learning training. There are some frameworks available to provide deep learning on Apache Spark, such as BigDL [60] and TensorFlow on Spark [61]. However, the use of distributed deep learning libraries such as BigDL, Horovod, and TensorFlow on Spark is still not compatible with edge devices due to their limited computational resources, including processing power, memory, and storage. These libraries are designed for the training of very large and complex deep learning models. More edge-friendly solutions and optimizations are still being explored [39]. Thus, we utilize a multilayer perceptron for the training of images. Although a multilayer perceptron is of low computational complexity, it is beneficial for quick experimentation, model prototyping, and scenarios involving limited computational resources. The selection of the MNIST dataset for image classification in our experiments is justified by its widespread use and established reputation as a benchmark dataset in the field of machine learning [62–64]. The MNIST dataset consists of a large collection of hand-written digits (0–9) in a consistent format, making it suitable for training and evaluating image classification models. Its simplicity and well-defined nature facilitate experimentation in Apache Spark's big data platform in edge, fog, and cloud computing layers. The MLP model consists of four layers and is trained for 1000 iterations. As this model showed nearly 95% accuracy in all the experiments on edge-, fog-, and cloud-layer computation, we focused on execution time as the key performance metric for the training. In our research, the primary focus is the evaluation of the efficiency and computational performance of big data platforms in edge, fog, and cloud computing layers for the training machine learning models as a representative example of the batch processing paradigm. The choice of execution time as a key performance metric is rooted in its direct reflection of the computational efficiency of the batch-processing-based training process [65].

4.1.2. Object Detection

Object detection and surveillance have become common applications associated with the widespread deployment of surveillance cameras. Autonomous video surveillance is the process of analyzing video sequences using object detection, segmentation, and classification for various applications. As object detection from multiple surveillance cameras is one of the use cases of stream processing, we deploy an object detection application on the

Apache Flink stream processing platform. The application is adopted from [66], and it is a traffic detection application that employs the YOLO V2 model for the detection of images from a traffic flow. In addition to the YOLO V2 model, we also evaluate the Tiny YOLO model to better understand both of the models' performance on resource-constrained devices. YOLO V2 is used for the detection of larger numbers of object categories, while Tiny YOLO is designed for limited categories of objects with faster detection. We investigated the inference workflow and performance of YOLO-based object detection for edge, fog, and cloud layers. A traffic video that consisting of 5792 sequence frames was used as the test dataset for object detection. We used the prediction time per frame and detection rate performance metrics to evaluate the Flink streaming application's performance in each computing paradigm. Prediction time per frame is an important metric when dealing with real-time streaming applications like video analytics or object detection, where low latency is paramount [67]. Prediction time per frame provides insights into the responsiveness of the Flink platform's stream processing application, ensuring the timely processing of incoming data. The detection rate is a metric used to assessing the accuracy and effectiveness of the Flink streaming application in identifying relevant patterns or objects within the streaming data [68].

4.1.3. Image Processing

Image processing applications are some of the most widely used serverless applications. The serverless image processing function is triggered when an image is stored or updated in the database or object storage. The image processing application used in this work is adopted from [69] and modified to be executed in an open-source serverless platform. As the deployment of the AWS image recognition collection has limitations prohibiting its execution in the edge and fog layers, we adapted the application to include simple image processing functions, namely metadata extraction, metadata transformation, metadata storage, and image resizing. Thus, our adapted application only consists of four image processing functions. The functions are chained together to process a request. In the cloud layer, we used step functions to create a workflow that connects all four functions. For the storage of metadata in the database, in the cloud layer, we used DynamoDB, and in the edge and fog layers, we used CouchDB, which is included in the OpenWhisk platform. We submitted 50 function invocations with a Poisson arrival rate of 0.5. In the cloud layer, a Python script that can invoke the step functions is hosted in an ec2 instance in the same region. We used the latency performance metric to measure the performance of the serverless platform in each layer. Latency directly measures the time taken for a serverless platform to respond to a request. In scenarios where user experience and responsiveness are critical, such as web applications or interactive services, latency is a fundamental metric. Latency is a widely accepted and standardized metric for the benchmarking and comparison of the performance of serverless platforms [70,71].

4.1.4. Experimental Testbed Configuration

Table 1 shows the hardware/software configurations of our experiments. To emulate local node collaboration, for the edge and fog layers, we set up a compute cluster of compute units in each layer using LAN. The cloud layer was an EMR cluster. On the cluster in each computing layer, we deployed the big data processing platforms to facilitate collaborative processing and distributed computing tasks. Our primary focus in this paper is to analyze and compare the performance of existing big data processing platforms across edge, fog, and cloud computing layers. We carefully selected widely used, existing, off-the-shelf platforms for the following three data processing paradigms: Apache Spark for batch processing, Apache Flink for stream processing, and Apache OpenWhisk for function processing. For batch processing, the ability of Spark to distribute computations across a cluster of nodes enhances its suitability for decentralized computing scenarios, facilitating parallel processing and optimizing performance, aligning with the distinctive characteristics of edge/fog computing layers. Flink's distributed stream processing capa-

bilities contribute to scalability, allowing it to effectively harness the collaborative power of nodes within the edge and fog clusters. The lightweight nature of Apache Flink aligns with the resource constraints often present in edge devices, making it an optimal choice for deployment and resource-efficient stream processing. The selection of Apache OpenWhisk for function processing in edge and fog computing layers is underpinned by its serverless architecture and suitability for the execution of small, event-driven functions in resource-constrained environments. It is one of the most widely used and popular open-source serverless platforms. These platforms are deployed across edge, fog, and cloud computing layers, and our experiments are designed to reflect real-world scenarios. There were some differences in the platforms' software versions, as we chose the latest available versions in the fog and cloud environments at the time of the experiment and manuscript preparation. As these were minor variations, we assumed that there was no impact on the comparative performance of the platforms.

Table 1. Experimental environment.

Data Processing Paradigm	Edge	Fog	Cloud
Batch Processing	Raspberry Pi (four-node cluster; CPU: four cores/node; Mem: 8 GB/node; Spark 3.3.1)	Ubuntu VM (four-node cluster; CPU: four cores/node; Mem: 8 GB/node; Spark 3.3.1)	Amazon EMR (four-node cluster; c5a.xlarge type; CPU: four cores/node; Mem: 7.5 GB/node; Spark 3.3.2)
Stream Processing	Raspberry Pi (four-node cluster; CPU: four cores/node; Mem: 8 GB/node; Flink 1.16.0)	Ubuntu VM (four-node cluster; CPU: four cores/node; Mem: 8 GB/node; Flink 1.16.0)	Amazon EMR (four-node cluster; c3.2xlarge type; CPU: 8 cores/node; Mem: 15 GB/node; Flink 1.17.0)
Function Processing	Raspberry Pi 4 (standalone; CPU: four cores; Mem: 8 GB; Lean OpenWhisk [72])	Ubuntu VM (standalone; CPU: four cores; Mem: 8 GB; OpenWhisk 1.0.0)	AWS Lambda

To simulate a distributed computing environment that reflects real-world conditions, a four-node cluster provides a balance between resource availability and complexity, allowing us to observe the scalability and performance of both Apache Spark and Apache Flink in a multinode setup and enabling us to assess the parallel processing capabilities of these big data processing frameworks, providing valuable insights into their behavior in a distributed computing environment representative of practical deployment scenarios. For the function processing platform, the standalone setup in our experiments is driven by the nature of function processing, where a single function is the primary unit of computation, and parallelization is not applicable.

4.2. Results

In this section, we compare the performance of the big data processing platforms using real-world applications by deploying them in each layer of the computing paradigm.

4.2.1. Image Classification as Batch Processing

In this experiment, the processing time in the MNIST training classification problem using the multilayer perceptron was measured. The experimental results are shown in Figure 3. The reported results are the average of ten application runs, and as there was no other application workload in the system, we did not find significant variability between runs. The experimental results show that the speed increased when we set the the number of cluster nodes from one to four, and throughout our experiments, we did not note any changes in accuracy when deploying Spark in different computing layers or any changes in the number of nodes when parallelizing and spreading the training tasks across the nodes. The fog layer is superior to the edge layer in terms of performance, primarily due to the greater resource limitations associated with edge devices. As expected,

edge computing, constrained by the capabilities of individual devices, faces challenges in efficiently handling resource-intensive tasks. The cloud layer demonstrates a steady improvement as the number of nodes increases. This scalability advantage of the cloud allows it to harness the collaborative power of multiple nodes in the cluster, resulting in enhanced overall performance. In an EMR (elastic MapReduce) cluster within the cloud layer, the computational resources are insufficient to handle the image classification task with only one node. The error indicating "not enough resources to process the task" suggests that the specific image classification task may have higher resource requirements than the available capacity of a single node within the EMR cluster. A two-fold speedup in the edge layer was observed when we performed processing in two nodes rather than one node. By distributing the image classification tasks across the nodes, we leveraged the parallel processing capabilities, allowing for the simultaneous execution of computations. This parallelization significantly reduces the overall processing time compared to a single-node configuration. The use of more nodes allows for more efficient resource scaling. In cases of increased demand or larger datasets, the system can dynamically allocate resources across nodes, providing scalability and further enhancing the overall speed of image classification. However, the parallelization of an application involves several factors, such as the communication overhead, the size of the data, and the uneven distribution of data across the nodes, as the data have to be distributed and shared between nodes over the network. Thus, Apache Spark does not show a consistent speedup due to several factors inherent in distributed computing and the characteristics of the Spark framework. The limited speedup observed in the fog layer, in contrast to the edge layer, can be attributed to the virtual nature of the compute instances and the network infrastructure connecting the virtual VM, characterized by a virtual private network.

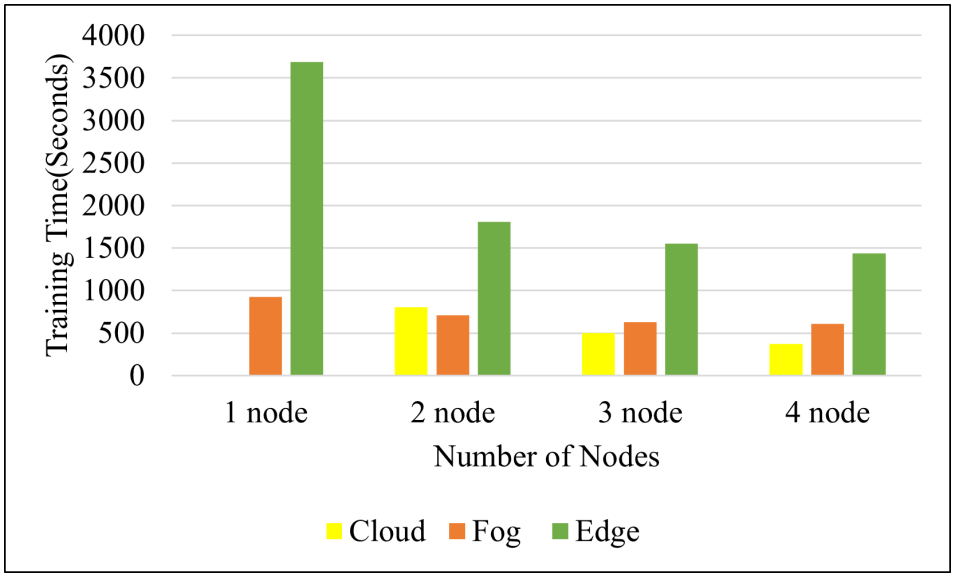

Figure 3. Impact of the number of nodes on performance.

Spark's configuration parameters, including the executor memory and executor cores, are key parameters in terms of optimizing the performance of applications. The executor memory is the amount of memory allocated to each Spark executor node, and the executor cores represent the number of CPU cores allocated to each executor node, which influences the parallelism and concurrency of the tasks. The executor memory and cores can be adjusted based on the nature of the workload, ensuring that the Spark application performs optimally under varying conditions. As Spark's performance in the cloud and in clusters of computers has been extensively studied [73,74], we focused on the most resource-constrained device and performed detailed experiments on the edge-layer Spark cluster. In this experiment, we set the number of executor nodes to four and the memory size to

6 GB. We measured the execution (training) time by varying the number of cores in each worker node from one to four. As shown in Figure 4, increasing the number of cores in each executor significantly increased the performance. Compared with our obtained results shown in Figure 3 and reported in this section, we can observe that training on one node with four cores took 3690 s, and training with four nodes with one core took 2467 s to complete. Training on one node had a longer execution time than with one core each and four nodes under the same executor memory size, which highlights the promising results obtained from the collaborative power of local computing devices when training a machine learning task in a distributed and collaborative way.

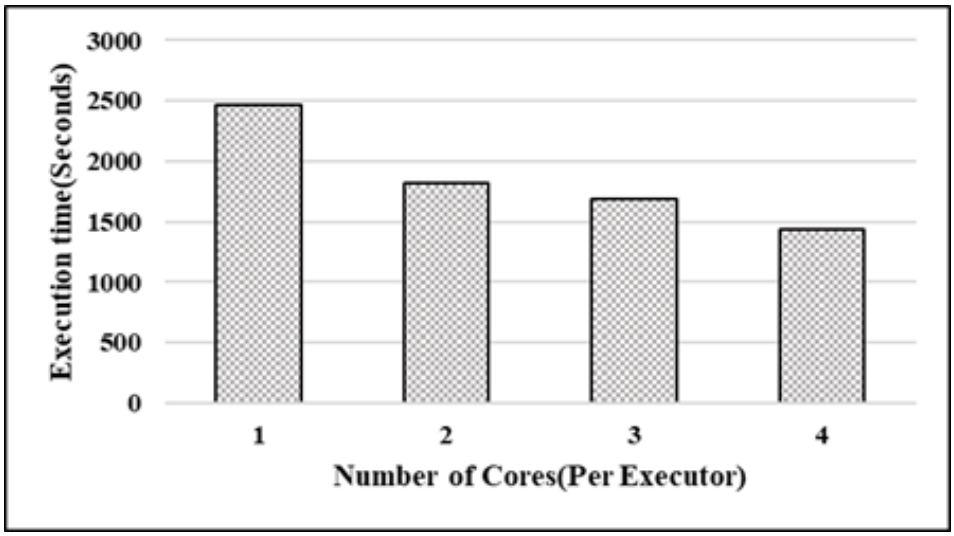

Figure 4. Impact of executor cores on performance.

To investigate the impact of the executor memory size on the performance, we measured the training time by varying the memory size of the Spark executor from 1 GB to 6 GB. We set the number of worker nodes to four, and the number of executor cores per node was four. The execution was repeated five times, and in the graph, it is plotted as the average value. As shown in Figure 5, increasing the size of the memory in each executor node did not speed up the execution. In addition, we found that execution with 2G memory took longer than with 1G, and execution with 5G memory took longer than with 4G. Although the image classification application was trained for 1000 iterations using the MLP model, this pattern was not caused by Spark image classification tasks. We observed the memory utilization in five-second intervals using the SAR (system activity report) command in Linux and confirmed that the actual memory utilization in all Spark worker nodes consistently remained below 1 GB throughout the entire execution period. Please note that the parallel training of MNIST classification tasks on Spark worker nodes does not fully consume all allocated memory, and the excessive allocation of memory beyond the necessary amount does not bring about benefits in terms of performance. Thus, we can conclude that increasing the Spark executor memory size cannot guarantee a speedup in a distributed environment. However, the current analysis, we did not specifically incorporate memory-intensive data analytic tasks. The observed patterns in Figure 5 primarily arose from the execution of the image classification training application, and as mentioned above, the actual memory utilization during this execution remained below 1 GB. If we execute memory-intensive data analytic tasks, changing the memory size impacts the system performance. This is particularly important in resource-constrained edge environments, where resources are limited and need to be carefully managed.

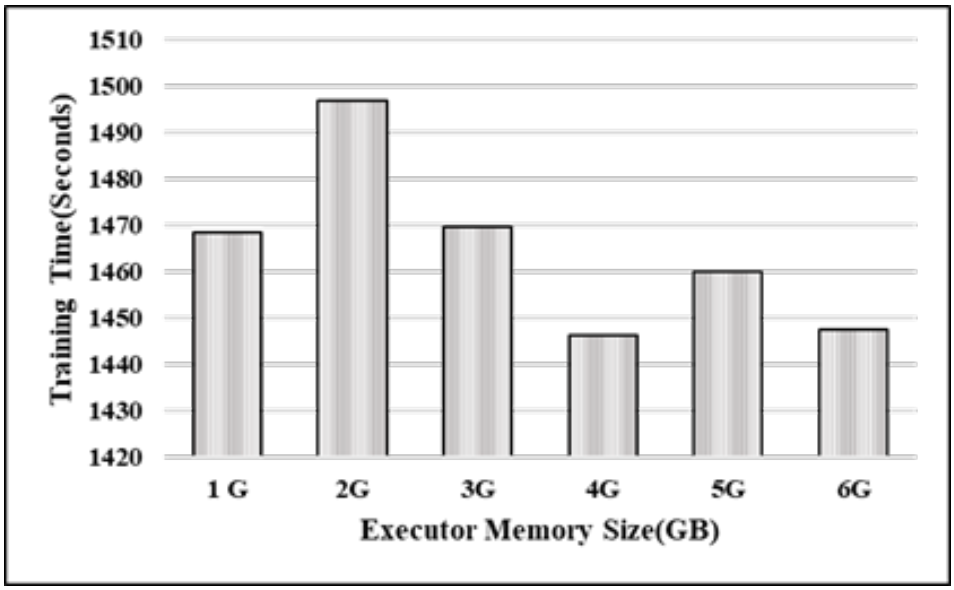

Figure 5. Impact of executor memory size on performance.

4.2.2. Object Detection as Stream Processing

Figure 6 shows a comparison of the prediction times per frame between edge, fog, and cloud layers for the object detection stream processing application. We deployed the object detection application using both the YOLO V2 and Tiny YOLO models on Apache Flink, and the detailed application parameters are described in Section 4.1.2. We repeated the execution 10 times for each computing paradigm to obtain more accurate average. Although we intended to set the same CPU and memory size for all computing layers, the cloud layer failed to allocate resources for the hosting of object detection models with the same CPU and memory size as the fog layer. Thus, we chose the c3.2xlarge instance type as a larger instance type in the region. It can be easily concluded that the fog layer outperformed the cloud layer and edge layer. For the YOLO V2 model, the average prediction time per frame was 0.765 s in the cloud layer, 0.367 s in the fog layer, and 2.138 s in the edge layer. For the Tiny YOLO model, the average prediction time was 0.23 s, 0.11 s, and 0.76 s in the cloud, fog, and edge layers, respectively. The observed lower performance in the edge layer, as anticipated, can be attributed to the resource limitations of edge Raspberry devices. Thus, the edge layer may struggle to meet the time-sensitive demands of certain applications. However, if the primary focus is on efficient resource utilization and time is not a critical factor, deploying on-edge devices remains a feasible option. This tradeoff emphasizes the need to carefully consider the specific requirements of an application and choose the computing layer accordingly. In contrast, both the fog and cloud layers, with more robust resources, showcase superior performance. However, for both models, i.e., YOLO V2 and Tiny YOLO, we found that the average prediction time of the cloud layer was approximately two times that of the fog layer, although we deployed the Flink cloud layer with a large number of CPU cores and a large amount of memory. This needs to be explored further in the future, but possible reasons are the actual specifications of the CPU, the CPU architecture, and the instance type. In the fog layer, the specifications of the CPU were as follows: 12th Gen Intel(R) Core(TM) i7-12700 2.10 GHz. We noticed that although we deployed four worker nodes as a cluster, only one worker node was occupied with jobs because of the single-input video stream. To exploit the full advantages of streaming platforms for object detection cases, multiple video streams that can be distributed and parallelized to nodes in the cluster need to be ingested through the use of a message-queuing platform such as Kafka.

Furthermore, to examine the real-time detection rate in the computing layers, we set the frame transition to 200 milliseconds and measured the number of frames that could be successfully detected and predicted from a total of 5792 frames. As shown in Table 2,

the fog-layer detection rate was higher than that of the cloud layer, and the edge-layer detection rate was only 4.49% . In the edge layer, the detection rate was significantly lower compared to the fog layer and cloud layer. We observe that in all three computing layers, all frames could not be detected during real-time video playback. In this case, a message-queuing system such as Apache Kafka can be deployed for the temporary storage and processing of video frames. In addition, the speed requirements for object detection from surveillance cameras vary depending on the specific use case and scenario. In real-time applications such as autonomous vehicles, low latency and high processing speeds are critical in making instantaneous decisions. There are also application scenarios that do not require instant decisions, such as environmental monitoring. Without such information, it is unfeasible to conclude whether a given edge or fog deployment is suitable for the use case in absolute terms.

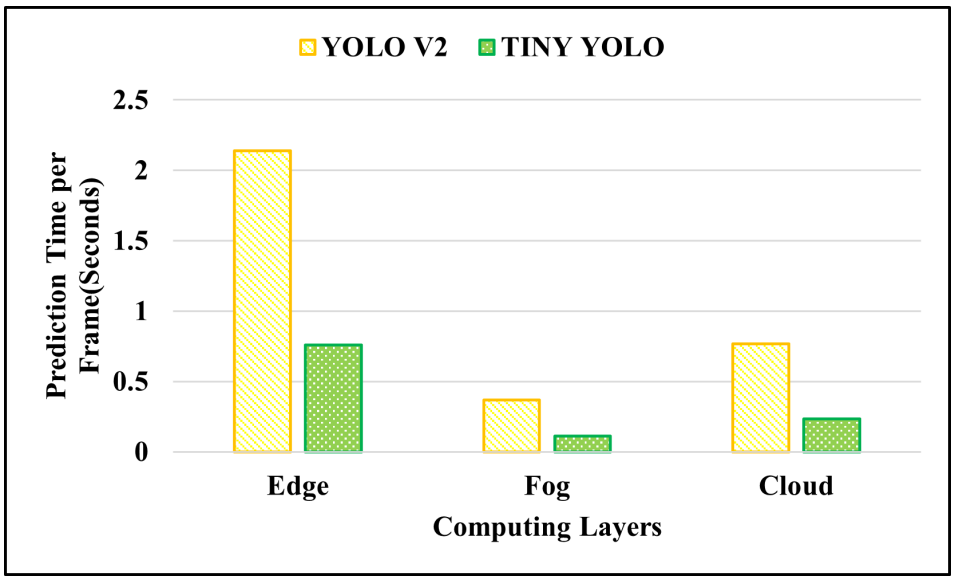

Figure 6. Average prediction time per frame.

Table 2. Detection rate (percentage).

Computing Layer	Detection Rate
Edge	4.49
Fog	35.50
Cloud	16.72

In summary, the consistent deployment of the same object detection model and the application of identical data preprocessing steps ensure a standardized approach to object detection across all layers. Moreover, network latency is not a relevant factor, as the workloads are input from the same machine. Thus, the primary factor contributing to the observed differences lies in the varying computational resources available in each layer. Differences in hardware capabilities can directly impact the prediction time and detection rate. The computational resources, processing power, memory, and storage capacity play a pivotal role in determining the efficiency and speed of executing resource-intensive tasks associated with object detection. The cloud layer does not showcase accelerated performance despite using a more powerful VM type. Differences in the underlying infrastructure, hardware specifications, or hypervisor configurations between the cloud VM and fog layer VM contribute to the performance outcomes. The edge layer is capable of accommodating a stream processing platform and its applications while exhibiting variations due to more constrained computational environments.

4.2.3. Image Processing as Function Processing

We evaluated the impact of the deployment of the function processing platform in the edge, fog, and cloud computing layers on performance by executing the application workload described in Section 4.1.3. Figure 7 reports the function invocations for latency with a different arrival rate of the lambda value in the Poisson distribution. There are three main observations. First, fog-layer function processing deployment is more efficient than deployment with the other two platforms, i.e., the edge and cloud layers. In our experimental analysis, with varying arrival rates of Poisson distribution (0.5, 0.05, and 0.005), our observations indicated that deploying function processing in the fog layer resulted in improved efficiency compared to deployments in both the edge and cloud layers across all tested arrival rates. The limitations in terms of resources within the edge layer were anticipated, contributing to its expected lower efficiency. Regarding the cloud layer, despite setting the same memory size (128 MB) for each function in the application across all computing layers, as serverless platforms allocate CPU power proportionally to the amount of memory provisioned, variations in performance may stem from the compute-unit specifications, such as actual specifications of the CPU, the CPU architecture, and the instance type, which differ between the cloud and fog layers. Similar patterns were also observed in the stream processing experiments, further emphasizing the impact of actual underlying hardware performance. The efficiency gain is particularly obvious, as evidenced by the significantly lower latency in the fog layer when compared to both the edge and cloud layers. This finding positions the fog layer as an efficient platform for function processing to improve responsiveness, making it well-suited for applications where timely interactions are required for user satisfaction. It also allows for improved accommodation of varying workloads and adaptation to changing application requirements. Secondly, in the cloud layer, the maximum invocation latency for the image processing sequence of functions is around 9–10 s. This maximum latency was observed in the first few invocations, and a latency of 6 s (which is double the mean value) was also found between the distant arrival times. This was because of the cold-start latency; however, we observed that it was constantly handled with varied arrival rates and never exceeded 9 to 10 s during several executions of our image processing application. A cold start is the time needed for the initialization of the container and execution environment in the serverless platform. Cold-start latency is experienced when a function is invoked for the first time or after a significant period of inactivity. For instance, if the function is idle for a specific amount of time, the system decontainerizes the container that is assigned for the function, and all the resources are deallocated for the scaling down of resources. Potential strategies may involve prewarming techniques [75], resource allocation optimization [76], or the utilization of serverless-specific features designed to address cold-start challenges [77]. However, in the edge layer, it largely depends on the arrival rate of the function invocation requests. This is because the distant arrival of function invocation requests leads to high latency due to long cold-start times and resource limitations in the edge layer. Thirdly, stable latency is more significant with regular and uniform function invocations. Function invocation patterns have a strong influence on the overall latency distribution of application execution. If they are too frequent, the processing and handling of many requests at the same time results in high latency. If the function invocation request arrival is distant, the cold start dominates the average latency, and it is not negligible, as function execution takes approximately two times longer than the average execution time.

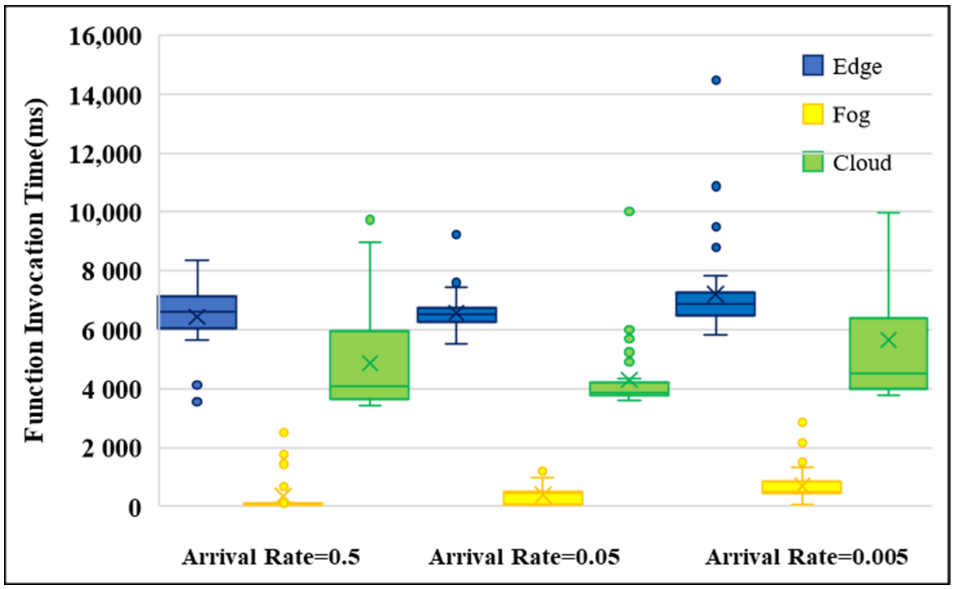

Figure 7. Function invocation latency.

To examine the overhead of serverless platform deployment in the edge and fog layers, we measured the CPU and memory utilization without the execution of function workloads. We measured this before and after the deployment of the serverless platforms. Our observations are described in Table 3. We found that in the edge layer, Lean OpenWhisk [72] does not consume as much memory as the fog layer does. It only consumes approximately 3% of the total memory in the Raspberry Pi after the deployment of all the containers and the controller, the Nginx server, CouchDB, and the Nodejs runtime environment. This is because the serverless edge layer combines a controller and revoker in one container and only loads the Nodejs runtime environment.

Table 3. Serverless platform deployments.

Description	Edge	Fog
CPU	<1%	<1%
Memory	3.13%	23%

5. Discussion and Recommendations

In this section, we discuss the limitations of this research and provide guidance for future research and the deployment of big data processing platforms in edge, fog, and cloud layers.

5.1. Discussion

In this section, we discuss the limitations and scope of this work.

- Training of large and complex models: For batch processing, we can only leverage an MLP model, which encompasses the Spark machine learning library, because there are compatibility issues with edge-layer devices when using a distributed deep learning library on Spark, such as BigDL, Horovod, or TensorflowonSpark, while we can easily leverage them in the fog and cloud layers to exploit the collaborative power of nodes in the cluster for the training of large and complex models.
- Non-parallelizable tasks: Our target context and environment involve the utilization of locally available resources in a collaborative way to solve big data problems. Thus, the tasks needed to be parallelized and distributed among the cluster nodes. However, the tasks were not parallelizable and needed a single computing power source to run; these types of tasks are not suitable for the target environment.

- Cluster setup overhead: We argue that the setup of a big data processing cluster can be achieved with little effort, as long as it is placed within the local network, allowing for collaborative utilization of locally available devices.

5.2. Recommendations

In this section, we present our recommendations based on our results, as well as the implications of our experiments. Generally, the deployment of big data platforms in fog computing layers leads to better performance and is appropriate in most cases.

According to our results, as reported in in Section 4.2.1, we specify the following recommendations for the **batch processing paradigm** :

- Deploying Spark in the edge computing layer can be a reasonable option if (a) the dataset size for training is small, (b) a long training time is tolerable for the business objective, and (c) only machine learning algorithms included in the Spark MLlib library are used. Our experiments reported in Section 4.2.1 were conducted under such a scenario and confirmed that training can be performed in the edge layer, achieving reasonable performance.
- When training large and complex models, it is necessary to use a distributed deep learning library on Spark, which is still not available or compatible with edge resources. Fog-layer Spark deployment is recommended for model training, where we can utilize the collaborative power of nodes in the cluster and a distributed deep learning library for speedup, as the cloud EMR cluster incurs high costs when sending large amounts of data to the cloud.
- Cloud EMR clusters are more suitable for the processing of extremely large datasets and when the throughput is important.

Based on our experiments, as presented in Figure 6 and Table 2 in Section 4.2.2, we list the following recommendations for the **stream processing paradigm**:

- Fog-layer-based stream processing achieves the best performance in terms of both the prediction time and detection rate. Although we can appropriately deploy the Flink streaming platform in the edge layer, the prediction time and detection rate are significantly low due to the size of the object detection model, which is hosted in resource-constrained devices.
- The advantage is pronounced only when multiple video streams are distributed to several nodes in the cluster and when real-time data analytics tasks that do not involve large computations in each individual task are distributed and parallelized among the nodes in the cluster.

In reference to the obtained results reported in Section 4.2.3, we present the following recommendations for the **function processing paradigm**:

- The fog computing layer is recommended for function processing, in which we observe the lowest latency for function invocations compared to the edge and cloud layers.
- Function processing is best-suited for applications with uniform and regular arrival patterns. Specifically, for the applications, the time between function invocations is not too long (i.e., function inactivity time), which can lead to cold-start latency, and the time between function invocations is not too short, which can lead to increased latency when serving multiple synchronous requests at the same time.

6. Conclusions

In this work, we conducted a comprehensive comparison of the deployment of big data processing platforms for three computing paradigms, namely batch processing, stream processing, and function processing in resource-constrained environments, namely edge and fog computing layers, versus cloud-based deployments. For big data processing platforms, we used three off-the-shelf big data platforms, namely Apache Spark for batch processing, Apache Flink for stream processing, and Apache OpenWhisk for function processing. The evaluation results demonstrate that deploying big data processing frameworks

in resource-constrained environments by utilizing the collaborative power of local low-cost devices is a viable and promising option for the three investigated processing paradigms. Finally, we discussed our recommendations in regard to building more practical big data frameworks for resource-constrained edge and fog computing environments and provided insights into the performance of applications in each computing layer.

As part of our future work, we would like to investigate the training of large and complicated deep learning models in edge and fog computing layers by leveraging the local nodes' computational capabilities to collectively contribute to the training processes of resource-intensive deep learning models. Furthermore, the computing continuum across edge, fog, and cloud computing layers could be studied, with a specific focus on observing the implications of big data processing tasks across this continuum. Understanding how big data processing tasks traverse and interact within this multilayered computing architecture is important in terms of optimizing the performance, minimizing the latency, and enhancing the overall efficiency of data processing workflows. This research direction involves the dynamics of data movement, task distribution, and resource utilization as large amounts of data move from edge devices to fog nodes and, ultimately, to cloud infrastructure. Another research direction involves proposing and implementing design changes on each data processing platform, tailored to resource-constrained edge and fog computing environments. In the context of big data solutions, optimizing these platforms for edge and fog environments by carefully considering the constraints of these environments, including their restricted computational capabilities and storage limitations, is crucial due to the unique challenges posed by limited resources and the need for efficient data processing. In addition, we plan to extend the scope of our experiments by incorporating diverse application scenarios. This expansion will ensure a more comprehensive evaluation of big data processing platforms, allowing us to obtain insights into their performance across various use cases and real-world applications.

Author Contributions: Conceptualization, M.A. and T.S.; methodology, M.A. and T.S.; software, T.S.; deployment, T.S.; evaluation and analysis, T.S.; writing—original draft preparation, T.S.; writing—review and editing, M.A.; supervision, M.A.; project administration, T.S.; funding acquisition, M.A. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by JSPS KAKENHI (grant number 22KF0314).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

EMR	Elastic MapReduce
DL	Deep learning
IoT	Internet of Things
GPU	Graphical processing unit
AI	Artificial intelligence
MLP	Multilayer perceptron
YOLO	You Only Look Once
AWS	Amazon Web Services
LAN	Local area network
VM	Virtual machine
CPU	Central processing unit
Mem	Memory
MLLib	Machine learning library

References

1. Rani, S.; Chauhan, M.; Kataria, A.; Khang, A. IoT Equipped Intelligent Distributed Framework for Smart Healthcare Systems. In *Towards the Integration of IoT, Cloud and Big Data: Services, Applications and Standards*; Rishiwal, V., Kumar, P., Tomar, A., Malarvizhi Kumar, P., Eds.; Springer Nature Singapore: Singapore, 2023; pp. 97–114. [CrossRef]
2. Sarker, I.H.; Khan, A.I.; Abushark, Y.B.; Alsolami, F. Internet of Things (IoT) Security Intelligence: A Comprehensive Overview, Machine Learning Solutions and Research Directions. *Mob. Netw. Appl.* **2023**, *28*, 296–312. [CrossRef]
3. Mukati, N.; Namdev, N.; Dilip, R.; Hemalatha, N.; Dhiman, V.; Sahu, B. Healthcare Assistance to COVID-19 Patient using Internet of Things (IoT) Enabled Technologies. *Mater. Today Proc.* **2023**, *80*, 3777–3781. . [CrossRef] [PubMed]
4. The Top 10 IoT Segments in 2018—Based on 1,600 Real IoT Projects. Available online: https://iot-analytics.com/top-10-iot-segments-2018-real-iot-projects/ (accessed on 20 July 2023).
5. Alhaidari, F.; Rahman, A.; Zagrouba, R. Cloud of Things: Architecture, applications and challenges. *J. Ambient. Intell. Humaniz. Comput.* **2023**, *14*, 5957–5975. [CrossRef]
6. Turukmane, A.V.; Pradeepa, M.; Reddy, K.S.S.; Suganthi, R.; Riyazuddin, Y.; Tallapragada, V.S. Smart farming using cloud-based Iot data analytics. *Meas. Sens.* **2023**, *27*, 100806. [CrossRef]
7. Alam, T. Cloud-Based IoT Applications and Their Roles in Smart Cities. *Smart Cities* **2021**, *4*, 1196–1219. [CrossRef]
8. Rajabion, L.; Shaltooki, A.A.; Taghikhah, M.; Ghasemi, A.; Badfar, A. Healthcare big data processing mechanisms: The role of cloud computing. *Int. J. Inf. Manag.* **2019**, *49*, 271–289. [CrossRef]
9. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog Computing and Its Role in the Internet of Things. In Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC '12, New York, NY, USA, 17 August 2012; pp. 13–16. [CrossRef]
10. Johri, P.; Balu, V.; Jayaprakash, B.; Jain, A.; Thacker, C.; Kumari, A. Quality of service-based machine learning in fog computing networks for e-healthcare services with data storage system. *Soft Comput.* **2023**. [CrossRef]
11. Azizi, S.; Farzin, P.; Shojafar, M.; Rana, O. A scalable and flexible platform for service placement in multi-fog and multi-cloud environments. *J. Supercomput.* **2023**. [CrossRef]
12. Karatas, M.; Eriskin, L.; Deveci, M.; Pamucar, D.; Garg, H. Big Data for Healthcare Industry 4.0: Applications, challenges and future perspectives. *Expert Syst. Appl.* **2022**, *200*, 116912. [CrossRef]
13. Agapito, G.; Cannataro, M. An Overview on the Challenges and Limitations Using Cloud Computing in Healthcare Corporations. *Big Data Cogn. Comput.* **2023**, *7*, 68. [CrossRef]
14. Quy, V.K.; Hau, N.V.; Anh, D.V.; Ngoc, L.A. Smart healthcare IoT applications based on fog computing: Architecture, applications and challenges. *Complex Intell. Syst.* **2022**, *8*, 3805–3815. [CrossRef] [PubMed]
15. Yi, S.; Hao, Z.; Qin, Z.; Li, Q. Fog Computing: Platform and Applications. In Proceedings of the 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), Washington, DC, USA, 12–13 November 2015; pp. 73–78. [CrossRef]
16. Muniswamaiah, M.; Agerwala, T.; Tappert, C.C. Fog Computing and the Internet of Things (IoT): A Review. In Proceedings of the 2021 8th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2021 7th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), Washington, DC, USA, 26–28 June 2021; pp. 10–12. [CrossRef]
17. Saini, K.; Kalra, S.; Sood, S.K. An Integrated Framework for Smart Earthquake Prediction: IoT, Fog, and Cloud Computing. *J. Grid Comput.* **2022**, *20*, 17. [CrossRef]
18. Verma, P.; Tiwari, R.; Hong, W.C.; Upadhyay, S.; Yeh, Y.H. FETCH: A Deep Learning-Based Fog Computing and IoT Integrated Environment for Healthcare Monitoring and Diagnosis. *IEEE Access* **2022**, *10*, 12548–12563. [CrossRef]
19. Alazzam, H.; AbuAlghanam, O.; Sharieh, A. Best path in mountain environment based on parallel A* algorithm and Apache Spark. *J. Supercomput.* **2022**, *78*, 5075–5094. [CrossRef]
20. Bagui, S.; Walauskis, M.; DeRush, R.; Praviset, H.; Boucugnani, S. Spark Configurations to Optimize Decision Tree Classification on UNSW-NB15. *Big Data Cogn. Comput.* **2022**, *6*, 38. [CrossRef]
21. Chebbi, I.; Mellouli, N.; Farah, I.R.; Lamolle, M. Big Remote Sensing Image Classification Based on Deep Learning Extraction Features and Distributed Spark Frameworks. *Big Data Cogn. Comput.* **2021**, *5*, 21. [CrossRef]
22. Ed-daoudy, A.; Maalmi, K.; El Ouaazizi, A. A scalable and real-time system for disease prediction using big data processing. *Multimed. Tools Appl.* **2023**, *82*, 30405–30434. [CrossRef]
23. Kumari, A.; Behera, R.K.; Sahoo, B.; Misra, S. Role of Serverless Computing in Healthcare Systems: Case Studies. In *Proceedings of the Computational Science and Its Applications—ICCSA 2022 Workshops*; Gervasi, O., Murgante, B., Misra, S., Rocha, A.M.A.C., Garau, C., Eds.; Springer International Publishing: Cham, Switzerland, 2022; pp. 123–134.
24. Bac, T.P.; Tran, M.N.; Kim, Y. Serverless Computing Approach for Deploying Machine Learning Applications in Edge Layer. In Proceedings of the 2022 International Conference on Information Networking (ICOIN), Jeju-si, Republic of Korea, 12–15 January 2022; pp. 396–401. [CrossRef]
25. Kong, L.; Tan, J.; Huang, J.; Chen, G.; Wang, S.; Jin, X.; Zeng, P.; Khan, M.; Das, S.K. Edge-Computing-Driven Internet of Things: A Survey. *ACM Comput. Surv.* **2022**, *55*, 1–44. [CrossRef]
26. Singh, R.; Gill, S.S. Edge AI: A survey. *Internet Things Cyber-Phys. Syst.* **2023**, *3*, 71–92. [CrossRef]
27. Apache Spark. Available online: https://spark.apache.org/ (accessed on 8 August 2023).
28. Apache Flink. Available online: https://flink.apache.org/ (accessed on 8 August 2023).
29. Apache OpenWhisk. Available online: https://openwhisk.apache.org/ (accessed on 8 August 2023).

30. Baresi, L.; Filgueira Mendonça, D. Towards a Serverless Platform for Edge Computing. In Proceedings of the 2019 IEEE International Conference on Fog Computing (ICFC), Prague, Czech Republic, 24–26 June 2019; pp. 1–10. [CrossRef]
31. Xu, J.; Palanisamy, B.; Wang, Q.; Ludwig, H.; Gopisetty, S. Amnis: Optimized stream processing for edge computing. *J. Parallel Distrib. Comput.* **2022**, *160*, 49–64. [CrossRef]
32. Karimov, J.; Rabl, T.; Katsifodimos, A.; Samarev, R.; Heiskanen, H.; Markl, V. Benchmarking Distributed Stream Data Processing Systems. In Proceedings of the 2018 IEEE 34th International Conference on Data Engineering (ICDE), Paris, France, 16–19 April 2018; pp. 1507–1518. [CrossRef]
33. Tekdogan, T.; Cakmak, A. Benchmarking Apache Spark and Hadoop MapReduce on Big Data Classification. In Proceedings of the 2021 5th International Conference on Cloud and Big Data Computing, ICCBDC '21, New York, NY, USA, 17–18 March 2021; pp. 15–20. [CrossRef]
34. Veiga, J.; Expósito, R.R.; Pardo, X.C.; Taboada, G.L.; Tourifio, J. Performance evaluation of big data frameworks for large-scale data analytics. In Proceedings of the 2016 IEEE International Conference on Big Data (Big Data), Washington, DC, USA, 5–8 December 2016; pp. 424–431. [CrossRef]
35. Grambow, M.; Pfandzelter, T.; Burchard, L.; Schubert, C.; Zhao, M.; Bermbach, D. BeFaaS: An Application-Centric Benchmarking Framework for FaaS Platforms. In Proceedings of the 2021 IEEE International Conference on Cloud Engineering (IC2E), San Francisco, CA, USA, 4–8 October 2021; pp. 1–8. [CrossRef]
36. Yu, T.; Liu, Q.; Du, D.; Xia, Y.; Zang, B.; Lu, Z.; Yang, P.; Qin, C.; Chen, H. Characterizing Serverless Platforms with Serverlessbench. In Proceedings of the 11th ACM Symposium on Cloud Computing, SoCC '20, New York, NY, USA, 19–21 October 2020; pp. 30–44. [CrossRef]
37. Baller, S.P.; Jindal, A.; Chadha, M.; Gerndt, M. DeepEdgeBench: Benchmarking Deep Neural Networks on Edge Devices. In Proceedings of the 2021 IEEE International Conference on Cloud Engineering (IC2E), San Francisco, CA, USA, 4–8 October 2021; pp. 20–30. [CrossRef]
38. Feng, H.; Mu, G.; Zhong, S.; Zhang, P.; Yuan, T. Benchmark Analysis of YOLO Performance on Edge Intelligence Devices. *Cryptography* **2022**, *6*, 16. [CrossRef]
39. Hao, C.; Dotzel, J.; Xiong, J.; Benini, L.; Zhang, Z.; Chen, D. Enabling Design Methodologies and Future Trends for Edge AI: Specialization and Codesign. *IEEE Des. Test* **2021**, *38*, 7–26. [CrossRef]
40. Rausch, T.; Hummer, W.; Muthusamy, V.; Rashed, A.; Dustdar, S. Towards a Serverless Platform for Edge AI. In Proceedings of the 2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19), Renton, WA, USA, 9 July 2019.
41. Pfandzelter, T.; Bermbach, D. tinyFaaS: A Lightweight FaaS Platform for Edge Environments. In Proceedings of the 2020 IEEE International Conference on Fog Computing (ICFC), Sydney, NSW, Australia, 21–24 April 2020; pp. 17–24. [CrossRef]
42. Smith, C.P.; Jindal, A.; Chadha, M.; Gerndt, M.; Benedict, S. FaDO: FaaS Functions and Data Orchestrator for Multiple Serverless Edge-Cloud Clusters. In Proceedings of the 2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC), Messina, Italy, 16–19 May 2022; pp. 17–25. [CrossRef]
43. Großmann, M.; Ioannidis, C.; Le, D.T. Applicability of Serverless Computing in Fog Computing Environments for IoT Scenarios. In Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion, UCC '19 Companion, Auckland, New Zealand, 2–5 December 2019; pp. 29–34. [CrossRef]
44. Tzenetopoulos, A.; Apostolakis, E.; Tzomaka, A.; Papakostopoulos, C.; Stavrakakis, K.; Katsaragakis, M.; Oroutzoglou, I.; Masouros, D.; Xydis, S.; Soudris, D. FaaS and Curious: Performance Implications of Serverless Functions on Edge Computing Platforms. In *Proceedings of the High Performance Computing: ISC High Performance Digital 2021 International Workshops, Frankfurt am Main, Germany, June 24–July 2, 2021, Revised Selected Papers 36*; Jagode, H., Anzt, H., Ltaief, H., Luszczek, P., Eds.; Springer International Publishing: Cham, Switzerland, 2021; pp. 428–438. [CrossRef]
45. Ab Wahab, M.N.; Nazir, A.; Zhen Ren, A.T.; Mohd Noor, M.H.; Akbar, M.F.; Mohamed, A.S.A. Efficientnet-Lite and Hybrid CNN-KNN Implementation for Facial Expression Recognition on Raspberry Pi. *IEEE Access* **2021**, *9*, 134065–134080. [CrossRef]
46. James, N.; Ong, L.Y.; Leow, M.C. Exploring Distributed Deep Learning Inference Using Raspberry Pi Spark Cluster. *Future Internet* **2022**, *14*, 220. [CrossRef]
47. Curtin, B.H.; Matthews, S.J. Deep Learning for Inexpensive Image Classification of Wildlife on the Raspberry Pi. In Proceedings of the 2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), New York, NY, USA, 10–12 October 2019; pp. 82–87. [CrossRef]
48. Apache Hadoop. Available online: https://hadoop.apache.org/ (accessed on 8 August 2023).
49. Kimovski, D.; Mathá, R.; Hammer, J.; Mehran, N.; Hellwagner, H.; Prodan, R. Cloud, Fog, or Edge: Where to Compute? *IEEE Internet Comput.* **2021**, *25*, 30–36. [CrossRef]
50. Lee, E.; Oh, H.; Park, D. Big Data Processing on Single Board Computer Clusters: Exploring Challenges and Possibilities. *IEEE Access* **2021**, *9*, 142551–142565. [CrossRef]
51. Pfandzelter, T.; Bermbach, D. IoT Data Processing in the Fog: Functions, Streams, or Batch Processing? In Proceedings of the 2019 IEEE International Conference on Fog Computing (ICFC), Prague, Czech Republic, 24–26 June 2019; pp. 201–206. [CrossRef]
52. Salloum, S.; Dautov, R.; Chen, X.; Peng, P.X.; Huang, J.Z. Big data analytics on Apache Spark. *Int. J. Data Sci. Anal.* **2016**, *1*, 145–164. [CrossRef]

53. Markovic, A.; Kolovos, D.; Soares Indrusiak, L. Distributed Data Locality-Aware Job Allocation. In Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, SC-W '23, New York, NY, USA, 12–17 November 2023; pp. 2089–2096. [CrossRef]
54. Apache Storm. Available online: https://storm.apache.org/ (accessed on 8 August 2023).
55. Apache Kafka. Available online: https://kafka.apache.org/ (accessed on 8 August 2023).
56. Carbone, P.; Ewen, S.; Fóra, G.; Haridi, S.; Richter, S.; Tzoumas, K. State Management in Apache Flink®: Consistent Stateful Distributed Stream Processing. *Proc. VLDB Endow.* **2017**, *10*, 1718–1729. [CrossRef]
57. OpenFaaS. Available online: https://www.openfaas.com/ (accessed on 8 August 2023).
58. Knative. Available online: https://knative.dev/docs/ (accessed on 8 August 2023).
59. Javed, H.; Toosi, A.N.; Aslanpour, M.S. Serverless Platforms on the Edge: A Performance Analysis. In *New Frontiers in Cloud Computing and Internet of Things*; Buyya, R., Garg, L., Fortino, G., Misra, S., Eds.; Springer International Publishing: Cham, Switzerland, 2022; pp. 165–184. [CrossRef]
60. BigDL. Available online: https://github.com/intel-analytics/BigDL (accessed on 8 November 2023).
61. Tensor Flow on Spark. Available online: https://github.com/yahoo/TensorFlowOnSpark (accessed on 8 November 2023).
62. Grover, D.; Toghi, B. MNIST Dataset Classification Utilizing k-NN Classifier with Modified Sliding-Window Metric. In *Advances in Computer Vision: Proceedings of the 2019 Computer Vision Conference (CVC)*; Arai, K., Kapoor, S., Eds.; Springer International Publishing: Cham, Switzerland, 2020; Volume 21, pp. 583–591.
63. Cheng, K.; Tahir, R.; Eric, L.K.; Li, M. An analysis of generative adversarial networks and variants for image synthesis on MNIST dataset. *Multimed. Tools Appl.* **2020**, *79*, 13725–13752. [CrossRef]
64. Japa, L.; Serqueira, M.; Mendonça, I.; Aritsugi, M.; Bezerra, E.; González, P.H. A Population-Based Hybrid Approach for Hyperparameter Optimization of Neural Networks. *IEEE Access* **2023**, *11*, 50752–50768. [CrossRef]
65. Assefi, M.; Behravesh, E.; Liu, G.; Tafti, A.P. Big data machine learning using apache spark MLlib. In Proceedings of the 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 11–14 December 2017; pp. 3492–3498. [CrossRef]
66. Object Detection on Apache Flink. Available online: https://github.com/mk-hasan/Flink-Kuberenets (accessed on 24 August 2023).
67. Lin, J.; Liu, D.; Li, H.; Wu, F. M-LVC: Multiple Frames Prediction for Learned Video Compression. In Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 13–19 June 2020; pp. 3543–3551. [CrossRef]
68. Ha, T.W.; Kang, J.M.; Kim, M.H. Real-Time Deep Learning-Based Anomaly Detection Approach for Multivariate Data Streams with Apache Flink. In *Proceedings of the ICWE 2021 Workshops*; Bakaev, M., Ko, I.Y., Mrissa, M., Pautasso, C., Srivastava, A., Eds.; Springer International Publishing: Cham, Switzerland, 2022; pp. 39–49.
69. AWS Samples. Available online: https://github.com/aws-samples/lambda-refarch-imagerecognition/ (accessed on 3 August 2023).
70. Ali, A.; Pinciroli, R.; Yan, F.; Smirni, E. Optimizing Inference Serving on Serverless Platforms. *Proc. VLDB Endow.* **2022**, *15*, 2071–2084. [CrossRef]
71. Shwe, T.; Aritsugi, M. Towards an edge-fog-cloud serverless continuum for IoT data processing pipeline. In Proceedings of the 2024 IEEE International Conference on Big Data and Smart Computing (BigComp), Bangkok, Thailand, 18–21 February 2024.
72. Lean OpenWhisk. Available online: https://github.com/kpavel/incubator-openwhisk/tree/lean (accessed on 1 August 2023).
73. Ahmed, N.; Barczak, A.; Susnjak, T.; Rashid, M. A Comprehensive Performance Analysis of Apache Hadoop and Apache Spark for Large Scale Data Sets Using HiBench. *J. Big Data* **2020**, *7*, 110. [CrossRef]
74. Mostafaeipour, A.; Jahangard, A.; Ahmadi, M.; Arockia Dhanraj, J. Investigating the performance of Hadoop and Spark platforms on machine learning algorithms. *J. Supercomput.* **2021**, *77*, 1273–1300. [CrossRef]
75. Roy, R.B.; Patel, T.; Tiwari, D. IceBreaker: Warming Serverless Functions Better with Heterogeneity. In Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '22, Lausanne, Switzerland, 28 February–4 March 2022; pp. 753–767. [CrossRef]
76. Yang, Y.; Zhao, L.; Li, Y.; Zhang, H.; Li, J.; Zhao, M.; Chen, X.; Li, K. INFless: A Native Serverless System for Low-Latency, High-Throughput Inference. In Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '22, Lausanne, Switzerland, 28 February–4 March 2022; pp. 768–781. [CrossRef]
77. Liu, X.; Wen, J.; Chen, Z.; Li, D.; Chen, J.; Liu, Y.; Wang, H.; Jin, X. FaaSLight: General Application-Level Cold-Start Latency Optimization for Function-as-a-Service in Serverless Computing. *ACM Trans. Softw. Eng. Methodol.* **2023**, *32*, 1–29. [CrossRef]