Educational Administration: Theory and Practice
2024, 30(11), 418 -430
ISSN: 2148-2403
https://kuey.net/

Research Article

Edge Computing Strategies For Secure Data Processing In Multi-Cloud Environments

Sai Sandeep Ogety*

*Independent Researcher, Raleigh, United States, ogetysaisandeep04@gmail.com

Citation: Sai Sandeep Ogety (2024), Edge Computing Strategies For Secure Data Processing In Multi-Cloud Environments, *Educational Administration: Theory and Practice*, 30(11) 418 – 430, Doi: 10.53555/kuey.v30i11.8560

ARTICLE INFO

ABSTRACT

To operate applications with low latency requirements, edge computing infrastructures are frequently used. Users can use nodes that are close to their physical locations to dispatch arithmetic and information to the Cloud faster. Because users move around a lot and there aren't many resources at the Edge, managing these infrastructures presents fresh, challenging problems. Partitioning. This research introduces the Allocation, Placement, and Scaling system for efficient, automated, and scalable management of large-scale Edge topologies. Edge computing serves as a serverless platform for the Edge. Applications composed of small, stateless functions that adhere to latency restrictions can be uploaded by service providers. It takes care of them by executing the apps as containers at runtime, moving the containers across the Edge topology to accommodate the geographical spread of the workload, and allocating resources efficiently. In order to enhance the security, the Trusted key-based Secure Communication (TK-SCom) Model proposes as it combined cryptography along with a data classification-based learning approach and it performs the encryption and decryption process for a larger volume of data. The data classification is performed between the sender and receiver as the classification contains classifier i.e. modified boost classifier to check whether the data is normal or abnormal based on DDoS attack. Based on the data classification, the data authentication is performed to ensure security in the proposed TK-Scom Model. Then the proposed TK-Scom Model security is ensured based on operational cost and computational time.

Keywords: Serverless, Edge Computing, Latency, Service Provider, Resource.

I. Introduction

When it comes to scalability, multi-cloud is essentially limitless. This allows application providers to rent out virtual resources. Although multi-cloud has drastically revolutionised the processes of software development, distribution, and execution, two key issues persist. To begin, consumers are always being asked to call upon remote services that may be placed in extremely remote locations (potentially, even in another continent). This might be a problem for augmented reality, self-driving cars, and other Internet of Things (IoT) applications that need low latency for real-time operation. Second, users' privacy is jeopardised since their information is sent to a single location [1]. The goal of edge computing is to offload processing from remote servers to devices closer to end users, such 5G antennae. Network latency is decreased and server load is lessened because to the benefits of locality and decentralisation. This improves the network's interactivity and safeguards users' personal information. Edge infrastructure consists of a network of scattered computer nodes (or topology). Both direct and routed connections between nodes are possible [2]. If no path exists, some of the nodes may be inaccessible to the network as a whole.

A key feature of modern edge infrastructures is their capacity to host and operate several apps in parallel. Augmented reality software for engaging with the environment, an app for helping conventional and autonomous cars avoid traffic, and an app for monitoring and regulating the energy use of numerous smart buildings can all function on the specialized edge topology of a smart metropolitan area in Fig. 1. Managing these globally dispersed infrastructures presents a number of challenges [3]. One must take into account not only the workloads and the performance of the programmes, but also the location of the users. In order to decrease communication latency, it may be perfect to copy and install a single application on the nodes that are

physically nearest to clients. However, this isn't always possible due to the Edge's limited resources, which stand in stark contrast to the cloud's almost infinite ones.
Managing the speeds at which components of Edge infrastructures operate is also crucial. The efficiency of runtime management might be diminished by the time it takes to make choices, which is common in centralised or heavily weighted approaches [4]. When edge nodes in densely populated areas fulfil the demands of mobile users, the resulting workloads on the system are expected to be very variable. How quickly things get done at work is heavily dependent on how long it takes to put a strategy into action. Service providers are increasingly adopting container-based deployments as a result of the maturation of virtualization technologies in recent years. Containers are a lightweight virtualization solution that works on the OS level. The software works with other containers to get access to the underlying operating system while remaining isolated in a sandbox. Due to the fact that they do not need to operate a full-fledged OS, containers can be scaled and changed more quickly than virtual machines [5-8].
Serverless computing is a recent cloud-based execution approach that allows service providers to structure programmes as lightweight, easily-managed stateless functions. An application slice in charge of a single functionality is called a function. Serverless computing hides the underlying infrastructure from users so that it is only accessible to cloud service providers, who are responsible for capacity planning, resource allocation, and function deployment in containers [9].

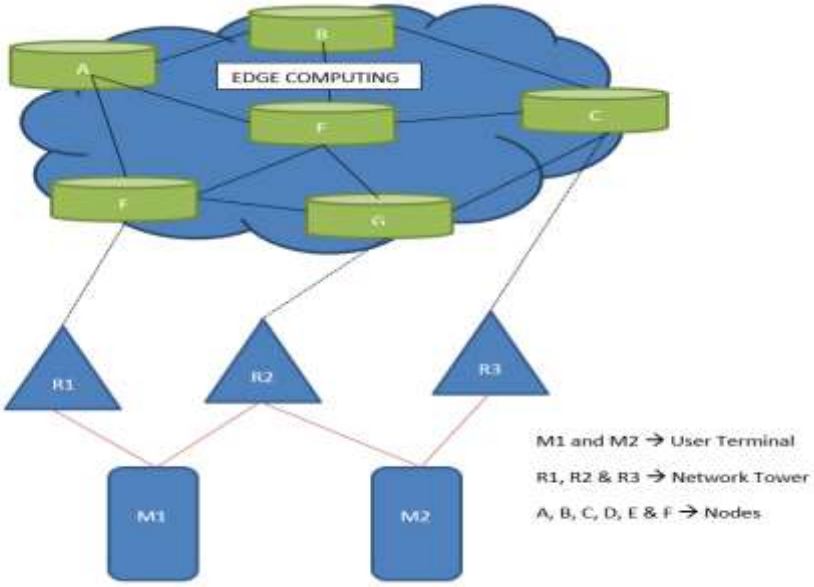

Fig.1. Serverless Edge Computing

The benefits of MEC are lost if the server hosting them becomes overburdened, yet edge infrastructure has limits on how many different applications and/or services it can host because of its dispersed and resource-finite nature [10]. Therefore, it is not possible to simply embrace virtualization and containerization technologies and implement the present cloud architecture at the edge. Serverless Architectures [11] and [12], also known as Functions-as-a-Service (FaaS), provide new, game-changing possibilities by shifting application execution environment responsibilities to the underlying infrastructure provider in the form of stateless functions. As a result, the service provider manages the containers in which the services are performed [13], doing away with the need for the customer to make resource reservations or to worry about scaling or load balancing [14]. The value of the edge nodes should rise as a result, since more features may be deployed with the same infrastructure and resources to meet the applications' need for low latency.
The following are the primary inquiries driving this research:

- Can serverless computing accommodate edge computing? And if so, how? While fine-grained scaling (i.e., at a function level) may be readily adaptable to the massively varied requirements and execution conditions at the edge [15], the scale-to-zero feature, in which unused containers are deallocated from the platform, is ideal for energy-aware IoT use cases with intermittent applications.
- Like functions in Serverless, many IoT systems depend on events that are triggered by sensing or acting, and they frequently detect or act only sporadically while sleeping the most of the time to save power, much like the promise of ephemeral functions in Serverless [16].

At first inspection, Serverless appears to be a near-perfect execution approach.
Serverless was originally designed for cloud settings, which do not have the same constraints as edge compute devices, therefore this provides a hurdle.

The Paper is organized as follows, Serverless Computing is employed in various technologies as discussed in Section 2. Functionalities of serverless computing is discussed in section 3. Existing works related to serverless computing as discussed in section 4. Challenges and approaches as discussed in section 5. Finally,

II. Employing Serverless On Various Technologies

Following the introduction of on-demand cloud services, a true pay-as-you-go model and simple scalability were missing. Unless they wanted to use cloud providers' generic auto-scalers, they had to build their own auto-scaling solutions from the ground up. In addition, the system was under extreme stress from the combination of dated practises such heavyweight virtualization and monolithic applications [17]. Meanwhile, so-called microservices were developing into service-oriented architectures; these are tiny, loosely coupled systems that are easier to deploy, maintain, and monitor. With this improvement, it became simpler to break down programmes into their component "functions," which are the fundamental building blocks of modern computing. This development ultimately led to the FaaS we know and use today. In contrast to traditional, monolithic application deployments, the controller may now optimise intermediate sections of applications using this white box method. However, techniques that make use of function composition can accommodate the reality of dependencies and interactions between functions.
Service complexity can be increased by combining functions [18]. Thus, event-driven programming was adopted by microservices to facilitate more granular development and communication between services. Due to the event-driven nature and dispersed nature of these systems, automation and CI/CD (continuous development and continuous deployment) become indispensable. Last but not least, Linux's cgroup mechanism for separating processes and resources laid the groundwork for the creation of containerization as we know it today, even if its full potential was not realised for many years. Containers, popularised by Docker, allowed for the development of microservices. However, until Serverless Computing came along, there was no answer that could combine all of these innovations while also enabling pure pay-per-use and seamless scaling. Serverless is a higher level of abstraction than Infrastructure-as-a-Service (IaaS), approaching Platform-as-a-Service (PaaS), because it gives users a platform on which to run code without needing to know anything about the underlying operating system (OS), not even in a virtualized environment like a virtual machine (VM) or container.

III. Functionalities on Edge Computing

A review of the literature suggests that using serverless computing has a variety of drawbacks and problems. Following is a list of challenges, both functional and non-functional, associated with serverless computing:

A. Value and price structure

Providers of serverless computing should minimise costs by making efficient use of resources throughout both the execution and idle phases of operation. The pricing method is a further issue with serverless computing as compared to other multi-cloud solutions [19]. For instance, the cost of using a dedicated server decreases when the function is CPU-bound rather than I/O-bound. Table 9 displays scholarly literature on the topic of serverless multi-cloud pricing and costs.

B. Frozen start

Since no services or functionalities are required, serverless computing may be scaled to infinity. Scaling all the way down to zero causes a condition known as cold start. Cold starts need additional time when resuming execution of serverless functions following an inactive period. Numerous research have looked at the cold start problem because of the significance of finding solutions to it.

C. Resource constraints

Resources are needed in serverless computing to make sure the platform can handle growing loads. This covers bandwidth, memory, CPU use, and execution time

D. Security

In serverless, cloud environments, security is the biggest challenge. With so many users using the same server, it might be difficult to keep each one's data safe when they all need to execute their own processes [20]. Thus, effective isolation is required. Another difficulty with process-sensitive data security is the issue of trust. Security features for serverless apps depend on the proper operation of many different parts of the system. Multiple researches on serverless security

E. Scalability

In serverless computing, scalability and functional flexibility must be ensured. For example, when a serverless application is experiencing high demand, the serverless cloud provider should be able to scale up to meet all of the demands.

F. Long-running
Serverless computing completes its tasks in a limited and brief amount of time, whereas certain procedures may require a lengthy execution period. These functions cannot sustain extended execution runs because they are stateless; once they are suspended, they cannot be revived.

G. Debugging and programming
There aren't enough debugging tools available right now. Additionally, monitoring tools are necessary since developers must keep an eye on the application to see how its features are operating. In order for developers to conduct refactoring operations like merging or separating functions and rolling back operations to a prior version, more sophisticated integrated development environments (IDEs) are required. The developer also needs to get logs with complete stack traces from calls to serverless functions. A reliable technique must be used to inform the developer of the specifics of an issue when it happens. There is presently no serverless computing equivalent of a stack trace. Numerous works that discuss the difficulties and problems of programming and debugging [21].

H. Supplier lock-in
As a result of the FaaS paradigm's separation of the code and data, the functions are heavily dependent on the ecosystem of the cloud service provider to store, acquire, and move data [282]. Customers are forced to depend on the serverless supplier for goods and services as a result of this problem, and switching vendors in the future is difficult and expensive. Customers must therefore wait for further services from the serverless supplier

I. Performance
Among the many performance issues and limitations that serverless computing confronts are those relating to scheduling and service call overhead. For instance, when a serverless function is activated in response to an event, scheduling decides which container or virtual machine (VM) will execute the function. Depending on a number of criteria (including, but not limited to, the quantity of available resources, the location of the input data and code, load balancing, etc.), this resource may have a significant impact on performance. Articles discussing the efficiency of serverless computing are compiled [22].

J. Mistake tolerance
It describes a system that functions and performs its functions even when some of its components fail. It mainly happens when some containers malfunction. A simple retry technique is utilised to get around this problem

K. Composition of functions
Users can deploy small stateless functions to the cloud from serverless cloud vendors to complete a specific job. However, in order to do some complex tasks, numerous functions must collaborate with one another [23]. In order to employ function composition in serverless multi-cloud effectively and efficiently, more study is required

L. Resource exchange
Serverless functions leverage resource sharing to lower multi-cloud costs. Resources might be hard to distribute across functions and other serverless components. Therefore, it is important to study proven methods for achieving this goal.

M. Testing
A serverless application is made up of numerous little tasks. The functionality of the application is achieved by these features working together. In order to ensure that the application functions effectively, integration testing for these functions is essential

N. System for naming and addressing
In serverless multi-cloud, end-users may install systems. All of these features fall short of enabling network-based monitoring of traffic. This feature is not available in existing serverless multi-cloud frameworks [24]. This is accomplished via the use of third-party tools, such as Amazon S3, for coordinating efforts and exchanging information. Thus, with serverless multi-cloud, it is difficult to establish the IP address of a function by other functions and services.

O. Dated systems
The term "legacy systems" describes outdated methods, tools, pieces of hardware, and software that are still in use. These systems ought to be accessible using serverless multi-cloud. Additionally, it might be necessary to move these systems to the cloud. Serverless cloud services need more development effort on the migration process and the extraction of functions from legacy systems before they can be deployed.

P. In charge of hybrid cloud

An application may be shared across many clouds with the help of a hybrid cloud (hybrid cloud) [25]. When different parts of an application's functionality are hosted by different serverless cloud providers, it may be difficult to keep track of everything and make sure it all works together as intended.

Q. Lack of support for quality of service (QoS)

The QoS of serverless functions cannot be controlled by users using the serverless platforms and frameworks that are now available The key factors affecting the quality of service in serverless computing are cold starts, queuing, and orchestration

R. Building complexity

A serverless application could include multiple functions, and the complexity of the architecture rises as the number of functions increases [25]. A complicated architecture results from managing various application-related functions and services

S. Tracking interactions

Typically, stateful queries are utilised by real-world applications. It indicates that implemented systems record the current status of user interactions and save them for later usage on the server. However, it is not immediately clear how stateless serverless functions would handle and manage the states of each user as represented in Fig. 2.

T. Concurrency control

When a function is invoked, it may handle any number of requests thanks to concurrency. For instance, a serverless function will handle a request that has been made and process it. A hybrid cloud allows for an app's deployment over many cloud platforms (hybrid cloud) [26]. It might be difficult to manage the interdependencies between different serverless cloud functions if, for instance, some of an application's capabilities are hosted by one serverless cloud provider while others are hosted by another.

U. Support for several device types

Some specialized hardware, such as graphics processing units (GPUs) and field programmable gate arrays, may not be supported by existing serverless platforms (FPGAs). Supporting heterogeneous hardware poses a difficult problem for vendors [27].

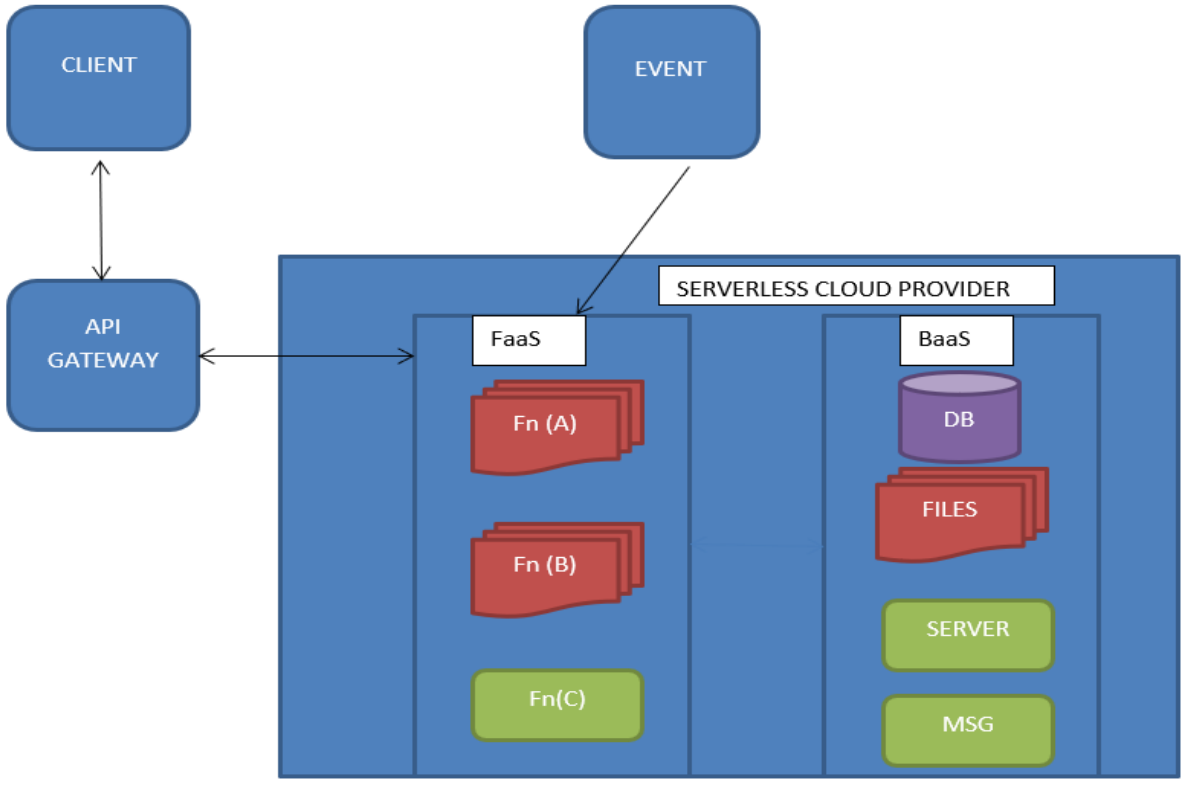

Fig. 2. Serverless Cloud Provider

IV. Existing Works Related to Serverless Architecture

For Serverless architectures, the service provider is in charge of everything, including monitoring servers and managing virtual machines and images throughout their lifetimes. Microservices, Function-as-a-Service, Event-driven Programming, Containerization, the Pure Pay-per-Use Model, and Frictionless Scaling are all examples of technologies that came together to form Serverless. [28] Proposed a novel construct to formalize a serverless application process and use analytical models to predict its average end-to-end response time and cost. As a result, we present a heuristic algorithm called Probability Refined Critical Path Greedy algorithm (PRCP) with four greedy strategies to answer two fundamental optimization concerns pertaining to performance and cost. In this paper, we address two research concerns about the modeling and optimization of serverless applications' cost and performance. We presented a formal definition of the serverless approach, taking into account a number of serverless cloud features. Then, we solved the problem of unpredictability in performance and cost for serverless applications by proposing the performance and cost models. We tested the validity of the proposed models by evaluating five serverless applications hosted on Amazon AWS.

We have solved two optimization problems: best performance under budget constraint and best cost under performance constraint. In response, we put out a probability-refined critical path algorithm with four greedy methods as a heuristic algorithm. Once more, we conducted experimental evaluations of the suggested methods on AWS to confirm their validity. Many applications are adopting serverless computing, yet many are unable to reap the benefits due to the difficulty of cold start. A cold start is significantly slower than invoking a function whose container has already been deployed, as has been amply demonstrated. The difficult element is to decrease cold start latency without using additional resources.

[29] minimize the cold start latency while avoiding the use of additional resources. We propose using a traditional statistical model called SARIMA to forecast future inbound requests. After receiving the model's predictions, we feed them into the PBA, which adjusts the quantity of pods in accordance with the model's projections. This makes it easier to provision the exact quantity of resources that the application needs at any given time. The programme runs without any issues as a result of neither over-provisioning nor under-provisioning of resources. In order to validate our technique, we compare our PBA to HPA, Kubernetes' default pod autoscaler. When comparing the two, HPA suffered from cold start, whereas PBA was able to gracefully mitigate the effect of cold start. Additionally, PBA used 18% fewer resources than HPA. There are still numerous facets of cold start in serverless computing that must be better understood and refined. One approach to solving this issue is through future projections, but further research is needed to fully grasp the root of the issue and determine whether there are any creative solutions.

[30] in-browser real-time facial recognition-based attendance tracking system was built and put into action utilising tensorflow.js, numerous parallel face-api.js models, and Google Cloud for dynamic real-time storage. The solution uses serverless edge computing to update attendance records and Google Sheets API to store them on the go, lowering overall latency. The attendance system uses a system web camera to track the individual, evaluates their emotions and movements to moderate and confirm their physical presence, verifies student presence if the accuracy is above 65%, validates them with the records, and then marks the student's attendance.

[31] Cross-monitoring architecture that can act rapidly when neighboring Edges are experiencing problems. This architecture imports measurements directly from a neighboring Edge to the Edge, enabling faster response. The architecture will be implemented using KEDA and Prometheus, according to the plan. Next, define and test the various metrics that can be utilized for cross-monitoring. When creating serverless platforms aimed at edge devices, unicernel technology is a potential substitute for containers. Our UniFaaS platform is more effective. In terms of processing speed, memory use, and boot-time, OpenWhisk [32]. We implemented this platform with our Smart Citizen Kit as a use case, showcasing its usefulness as a tool for creating IoT applications.

Three Serverless Data Pipeline (SDP) strategies have been proposed: DFT, OSS, and MQTT-based SDP using Apache NiFi, MinIO, and MQTT services, respectively. Three different fog computing applications, Aeneas, Pocketsphinx, and a video processing application, were the subjects of our implementation of these methods [33]. We examined their performance using measures like processing time (computation time, disc access time, and network access time) and resource consumption (CPU, Memory, Network, and Disk utilization) and thoroughly analyzed the results by producing an appropriateness index for each of them. Additionally, using dynamic and stochastic workloads in the future will make it easier to assess the proposed SDP techniques' dependability, robustness, throughput, etc. [34] Solution for edge computing without devices was suggested to enhance service for mobile users. We have put up a seamless virtual function migration plan to maintain services notwithstanding user mobility.

[35] created a migration interface to facilitate function transfer while needing little input from function developers and attaining user transparency. We've turned the migration into a binary linear programming issue and presented an online technique to solve it. We made a broadcasting match as a typical virtual function that uses a lot of packets and is sensitive to latency. We then put it on real edge devices. Telemetry from 5G transportation networks promotes edge & cloud deployment of latency-sensitive serverless sub functions. Serverless functions provide a lightning-fast invocation time of less than 450ms once they were deployed [36]. Significant potential for processing speed and cost reduction through effective configuration adjustment and

identified its real-world difficulties. Motivated by our measurement research findings, we created CharmSeeker [37], a solution that employs a properly built SBO algorithm to automatically tune setups for serverless video processing pipelines. AWS Lambda is further implemented in a prototype for evaluation.
Significant potential for processing speed and cost reduction through proper configuration adjustment was recognized, as well as its real-world challenges. Measurement research, we developed CharmSeeker, a solution that uses a well-designed SBO algorithm to automatically tune settings for serverless video processing pipelines [38]. In order to evaluate a real-world IoT serverless application on both edge-core installations, AWS Lambda is further deployed in a prototype. The proposed component can be utilised to increase the processing power of hardware-restricted edges nodes so that they can compete with more reliable edges nodes and the core cluster. OT/IT integration issue using a theoretical edge/fog node as a Convergence layer with storage and processing capabilities [39].
Due to the rapid development of serverless computing and end devices serverless computing is in high demand. To evaluate the performance of serverless computing on edge devices, we provide EdgeFaaSBench, an innovative set of benchmarks for edge devices using serverless [40] computing. EdgeFaaSBench is built on a modern container engine/orchestration and a widely used open-source serverless framework. Examine whether a reinforcement-based strategy to resource-based auto-scaling in OpenFaaS [41], the most popular open-source serverless platform, is applicable. When working with edge computing on limited-resource devices or machines, serverless methods are very practical.
[42] Suggested the serverless edge computing strategy for putting machine learning applications into practice. The overall methodology has demonstrated serverless computing's appropriateness for workloads involving machine learning carried out in dispersed edge environments. We also thoroughly assess the performance of a well-known case study image classification using the MNIST dataset during training and serving. The serverless approach can guarantee optimal response and service life, minimize the machine learning application's end-to-end delay, and support distributed machine learning, according to our results. Design and execution of a device-driven, on-demand strategy [43] for deploying serverless computing capabilities to the edge.
Future RT-FaaS systems may constitute the evolution of current cloud strategies. Second, three design methods for providing real-time communications in such a system were offered, and related work that could be prospective implementations of them was summarized [44]. Finally, we suggested a real-time scheduling approach, partitioned-EDF with our partitioning algorithm that can perform RT functions. Assuming heterogeneous, multi-node multiprocessor scheduling, we show that our algorithm beats the relevant prior approaches in terms of the number if real solutions, provided CPUs, and runtime complexity. An innovative, decentralized FaaS-based architecture is intended to automatically balance the traffic load among edge nodes that are a part of federated Edge Computing ecosystems [45].
[46] Examining several methods to reduce latency in serverless applications. Instance reuse and instance pre-warming are the two basic methods for addressing the cold-start issue, and we examine how each of these methods affects both the application response time and resource usage. These results reveal how instance reuse mechanisms can greatly reduce the frequency of cold-starts and hence improve response time; however, selecting an appropriate keep-alive period matched to each specific application or input data profile is critical to get the best reaction time without incurring additional resource costs.
Standard architecture that will serve as a guide for creating Serverless IoT systems that are simple to connect and communicate with. The use of the serverless computing paradigm for Internet of Things systems is driven by the ability of the system engineers to include additional decentralization. As a result, a large amount of work will be done at the Edge and Fog layers. Only the most demanding functions will be outsourced to the cloud, resulting in more efficient and cost-effective systems. The fully-centralized strategy's optimal resource provisioning and allocation [47], which takes request queuing and scheduling into account. This method provisions the best possible set of function instances on a periodic basis to maximize CPU utilization [48], and it allocates jobs to the best possible computing places to maximize the proportion of jobs completed by their due dates. We proposed a realistic centralized strategy that only requires periodic coordination among computation spots for provisioning and leaves admission choices to computation spots due to the coordination overheads of such an approach.
Astra, an optimization framework, is being used to traverse the cost-performance tradeoff for serverless analytics operations. When formulating optimization issues for user-specified objectives [49], Astra focuses on modelling of project completion time performance and financial cost. Astra determines the optimal solutions of resource configuration and Lambda function orchestration based on graph theory [50], in order to either minimise work completion time with a budget constraint or minimise monetary cost with a performance criterion. Processing speed, memory use, and boot-time are all areas where UniFaaS platform performs better than OpenWhisk. With our Smart Citizen Kit as a use case, we deployed this platform.
When compared to OpenWhisk, the UniFaaS platform outperforms it in terms of processing speed, memory usage, and boot-time [51]. With our Smart Citizen Kit as a use case, we set up this platform. Developing mobile applications for effective execution in serverless edge computing environments presented us with a number of challenges. Here, we discuss the issues we faced and the solutions we came up with to address them as discussed in section 5.

V. Challenges and Approaches Related to Severless Computing

There are numerous issues we faced in designing mobile applications for efficient execution in serverless edge computing settings, and summarize our approaches tackling the numerous difficulties.

A. Scaling of application micro services:

Multiple instances of application micro services must be generated in order to handle processing of data from this set of cameras when a developer asks DataXe to deploy a certain application pipeline on a set of sensors, such as cameras. Poor estimates can result in over-provisioning or under-provisioning of microservices, reducing the accuracy of insights derived from analytics processing. In contrast to managing state across several cameras, auto-scaling stateless micro services are simpler than auto-scaling stateful micro services. As a result, DataXe creates a domain specific micro service replica for every reported stream generated by the AU (one-to-one mapping), while there is no each representation of registered streams to stateless micro services. DataXe dynamically scales up or scales down the number of instances of stateless microservices to effectively process the camera streams based on the processing pace of the input data streams.

B. Enhancing application-level effectiveness on a large scale:

An application consists of several microservices, and an efficient application execution necessitates both efficient communication and processing. Efficiency in communication and processing both refer to the speed at which data is transferred within and between microservices. The simultaneous handling and management of both is a challenging endeavour. By ensuring that the majority of time is spent on meaningful tasks rather than waiting for I/O, DataXe helps to monitor and improve application-level performance. The microservices are never idle or busy waiting because to the high performance communication techniques deployed. Such careful data management within and across microservices enables excellent microservice performance, which leads to increased application-level performance.

C. Scalable increasing system efficiency.

It makes sense to assume that enhancing application-level performance would also improve system-level performance. However, application-level performance and system-level performance are distinct concepts. Take into account two distinct applications that are performing a task that is shared by the two applications. Application-level performance enhancement would concentrate on getting the processing and communication correct in order to complete this typical task. But on a systemic level, we are exerting twice as much effort to complete the same activity. Such duplication obviously wastes processing power, but it can be disastrous for apps that use video analytics. For example, video analytics apps feature various AI engines that make considerable use of GPUs to perform deep learning-based AI models. These models need a substantial amount of memory resources in a GPU. Since GPU memory is limited, it may be impossible to load two copies of an AI model (such object identification) if it is used by more than one analytics pipeline processing more than one video stream. One of the analytics pipelines will malfunction in such situations.

Thus, it is crucial for DataXe to identify common AI models across several application pipelines and load the model on the GPU just once. DataXe must manage the GPU-processing of the AI model across multiple distinct application pipelines. Therefore, managing system-level performance is more difficult than managing application-level performance. DataXe carefully monitors individual applications, including numerous microservices that are a part of the application, in order to enhance system-level performance. DataXe automatically establishes a common resource of microservice instances for programs that can benefit from reuse and sharing, all while maintaining the processing pace required for each video input stream. By creating a restricted number of microservice instances and spreading them among applications, the overall utilisation of system resources is maximised, hence increasing system-level efficiency. As an example, when microservices use common deep learning models, only a single model is loaded into GPU memory and shared across applications. This keeps applications from running out of resources or crashing, which improves system-level performance. The security and privacy of each application pipeline are appropriately taken into account while enabling such sharing.

1. Proposed Trusted key based Secure Communication (TK-SCom) Model:

Based on Fig. 3, feature selection and data classification are performed in predicting the DDoS attack to classify the data as normal / abnormal. In the data classification model, several feature subsets are segregated with the association of extracting the feature. Then cross validation model is applied to improve the data accuracy. Finally the classifier of modified XG Boost algorithm is performed as it improve the computational speed and data reliability. Then the cloud based knowledge discovery is deployed to store the large volume of data and then explore the data used to perform the feature extraction.

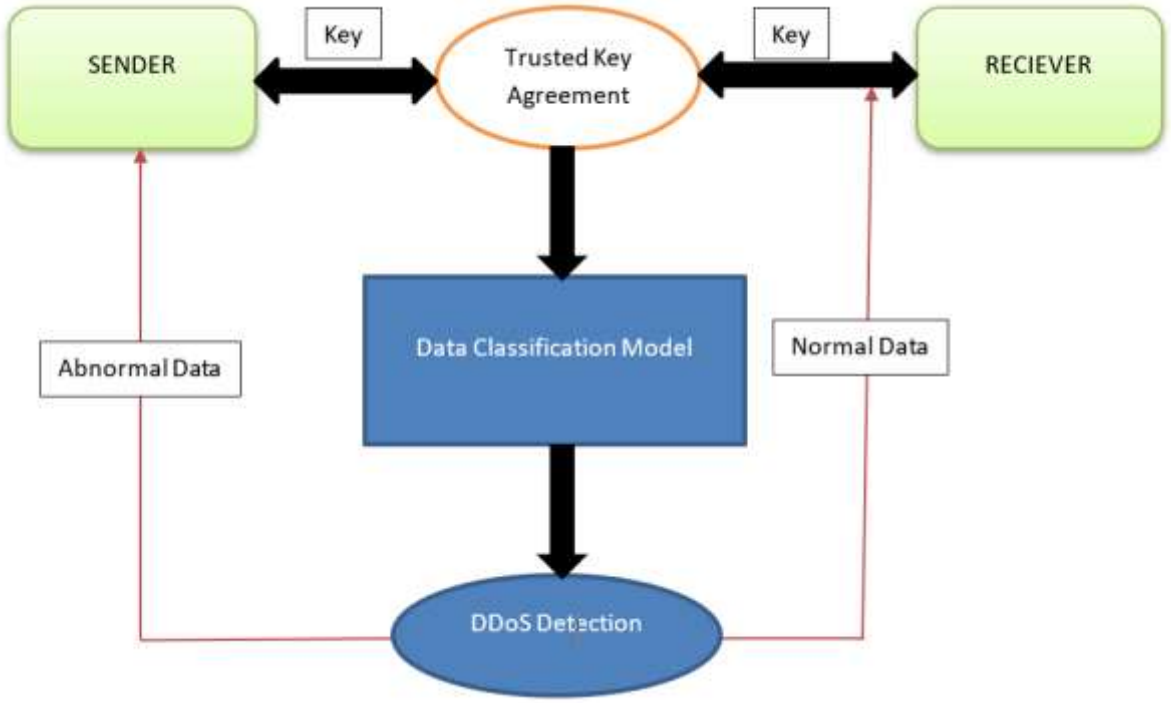

Fig. 3. Trusted key based Secure Communication (TK-SCom) Model

2. Trusted key based Secure Communication (TK-SCom) Model:

In the proposed Trusted key based Secure Communication (TK-SCom) Model, which act as the anomaly detection model as it associated with results generated form the learning classifier and perform prediction by applying best feature subsets. For the classifier, modified XG Boost algorithm is deployed as it helps to predict the DdoS attack to categorized normal or abnormal data. Then the above setup is deployed between the sender and receiver and the sender initiate the connection with some data entity, which is send to the receiver i.e. server. Between the client and server, the data classification model is deployed associated with feature extraction to predict the DdoS attack whether the data is malicious or not as represented in Fig. 3.

1. To verify the key-based agreement between the two point 'A' and 'B' where A → Sender and B→ Receiver.
2. Initially, 'A' request for the connection establishment with authentication along with One Time Pad (OTP) and Nonce, so that 'A' can initiate the establishment with session information.
3. Then after initiating the connection request along with data, then trust is ensured as it added into the data classification as it helps to predict the threat detection to categorize the data whether it is normal or abnormal.
4. The mask ID is generated for 'A' and 'B' along with the digital signature as the sender performs encryption along with the private key and decrypted by the receiver.
5. Receiver also uses same procedure to establish the connection with the sender as it uses asymmetric type of algorithm.
6. By using the integration of digital signature, verification is done based on the public and private key of 'A' and 'B'.
7. Using the digital signature as it performs encryption and decryption, the verification time will be reduced as the digital signature is applied with batch verification and multiply dot operation.

Algorithm: 1 Trusted key-based Secure Communication (TK-Scom) Model

Input: Sender and Receiver 'A' and 'B' along with private and public key
Output: Connection established with data exchange

```
Initialize 'A' and 'B' with private and public key along with mask ID.
Establish the authentication connection.
'A' sends the login establishment with certain key 'K'.
Classifier {Feature 1,2, …n} = Identification of threat.
If (Threat = Normal Data)
{
        'B' receives data along with digital signature 'n'
            Digital Signature {Encryption; Decryption}
            Hash Function and information batch
            Secure verification
```

11. Connection establishment completes for initiate the information exchange between ‘A’and ‘B’.
12. Else
13. Terminate the connection establishment
14. }

VI. PERFORMANCE ANALYSIS

In the analysis, the data classification and feature extraction are performed based on the input datasets. The modifier classifier is used to detect whether the data is abnormal or normal based on a DDoS attack. In order to analyze the performance of the learning classifier, the detection rate is considered.
In Fig. 4, the operational time is determined for the proposed TK Scom Model as it performs reduced in the operational time in terms of milliseconds. The proposed TK Scom Model analyze the decrease in the value as the test data size varies from 0 to 60. The proposed model outperforms the existing model such as Identity Auth Model, ECC Auth Model, and Cloud MW Model.

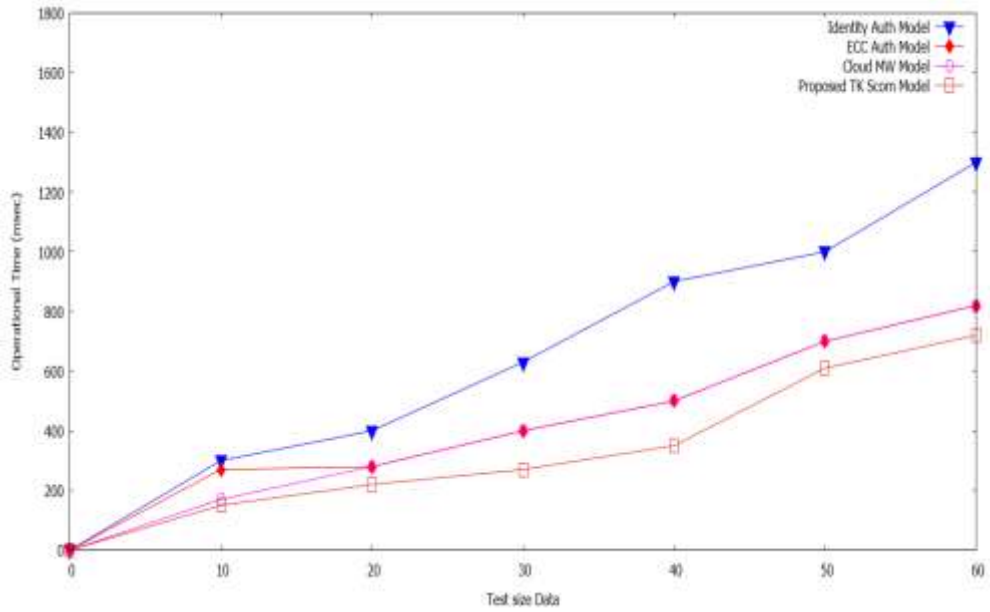

Fig. 4. Test Size Data Vs. Operational Time

In Fig. 5, the Computational cost is determined for the proposed TK Scom Model as it performs reduced in the operational time in terms of KiloBytes. The proposed TK Scom Model analyzes the decrease in the value as the test data size varies from 0 to 60. The proposed model outperforms the existing model such as Identity Auth Model, ECC Auth Model, and Cloud MW Model.

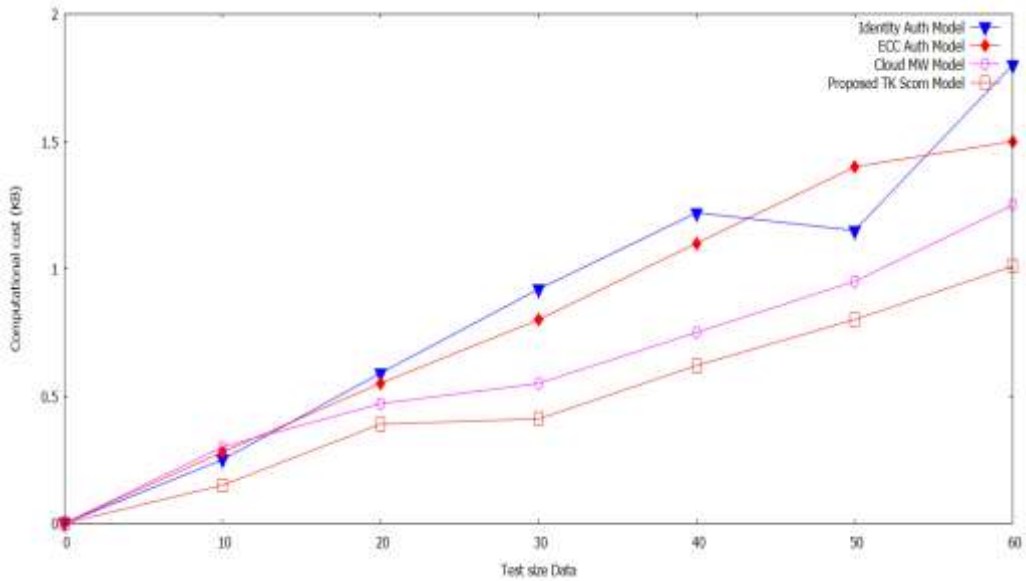

Fig. 5. Test Size Data Vs. Computational Cost

VII. Conclusion

The cloud plays a significant role as there is a vast amount of information is used as it provides certain services for various applications. Securing the cloud is a challenging task as it provides different services in the sector of the cloud. The services like feature extraction and data classification are performed as it contributes to the modified learning classifier it combines the objective function. It helps to optimize the process to obtain improved classification accuracy and check whether the data is normal or abnormal based on DDoS attacks. The modified classifier performance is analyzed based on the detection rate and classification accuracy. To establish secure communication based on cryptography of encryption/decryption and between the sender and receiver, a data classification process is performed. Based on the classification of data and DDoS verification, the data is sent to the receiver to ensure secure communication based on data confidentiality and authentication. The proposed model is analyzed in terms of operational cost and computational cost is determined.

References:

1. G. Coviello, K. Rao, M. Sankaradas and S. T. Chakradhar, "DataX: A system for Data eXchange and transformation of streams", The 14th International Symposium on Intelligent Distributed Computing (IDC 2021), 2021
2. S. Ranger, "What is the IoT? Everything you need to know about the Internet of Things right now", Feb. 2020
3. van Eyk, L. Toader, S. Talluri, L. Versluis, A. Uţă and A. Iosup, "Serverless is more: From paas to present multi-cloud", IEEE Internet Computing, vol. 22, no. 5, pp. 8-17, 2018.
4. Cristina L. Abad, Edwin F. Boza, and Erwin Van Eyk. 2018. Package-aware scheduling of FaaS functions. In Companion of the 2018 ACM/SPEC International Conference on Performance Engineering. 101–106.
5. Gojko Adzic and Robert Chatley. 2017. Serverless computing: Economic and architectural impact. In Proceedings of the 11th Joint Meeting on Foundations of Software Engineering. ACM, 884–889.
6. Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and DianaMaria Popa. 2020. Firecracker: Lightweight virtualization for serverless applications. In 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI'20). 419–434.
7. Nabeel Akhtar, Ali Raza, Vatche Ishakian, and Ibrahim Matta. 2020. COSE: Configuring serverless functions using statistical learning. In IEEE Conference on Computer Communications (INFOCOM'20). IEEE, 129–138.
8. Eyhab Al-Masri, Ibrahim Diabate, Richa Jain, Ming Hoi Lam Lam, and Swetha Reddy Nathala. 2018. A serverless IoT architecture for smart waste management systems. In IEEE International Conference on Industrial Internet (ICII'18). IEEE, 179–180.
9. May Al-Roomi, Shaikha Al-Ebrahim, Sabika Buqrais, and Imtiaz Ahmad. 2013. Multi-cloud pricing models: A survey. International Journal of Grid and Distributed Computing 6, 5 (2013), 93–106.
10. Kalev Alpernas, Cormac Flanagan, Sadjad Fouladi, Leonid Ryzhyk, Mooly Sagiv, Thomas Schmitz, and Keith Winstein. 2018. Secure serverless computing using dynamic information flow control. arXiv preprint arXiv:1802.08984 (2018).
11. Sundar Anand, Annie Johnson, Priyanka Mathikshara, and R. Karthik. 2019. Low power real time GPS tracking enabled withRTOSandserverlessarchitecture. In4th IEEEInternational Conference on Computer andCommunication Systems (ICCCS'19). IEEE, 618–623.
12. Sundar Anand, Annie Johnson, Priyanka Mathikshara, and R. Karthik. 2019. Real-time GPS tracking using serverless architecture and ARM processor. In 11th International Conference on Communication Systems & Networks (COMSNETS'19). IEEE, 541–543.
13. Lixiang Ao, Liz Izhikevich, Geoffrey M. Voelker, and George Porter. 2018. Sprocket: A serverless video processing framework. In Proceedings of the ACM Symposium on Multi-cloud. ACM, 263–274.
14. Dulcardo Arteaga, Jorge Cabrera, Jing Xu, Swaminathan Sundararaman, and Ming Zhao. 2016. CloudCache: Ondemand flash cache management for multi-cloud. In 14th USENIX Conference on File and Storage Technologies (FAST'16). 355–369.
15. Gabriel Aumala, Edwin Boza, Luis Ortiz-Avilés, Gustavo Totoy, and Cristina Abad. 2019. Beyond load balancing: Package-aware scheduling for serverless platforms. In 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID'19). IEEE, 282–291.
16. Arda Aytekin and Mikael Johansson. 2019. Harnessing the power of serverless runtimes for large-scale optimization. arXiv preprint arXiv:1901.03161 (2019).
17. Ting Bai, Jian-Yun Nie, Wayne Xin Zhao, Yutao Zhu, Pan Du, and Ji-Rong Wen. 2018. An attribute-aware neural attentive model for next basket recommendation. In The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval. 1201–1204.
18. Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, et al. 2017. Serverless computing: Current trends and open problems. In Research Advances in Multi-cloud. Springer, 1–20.

19. Priscilla Benedetti, Mauro Femminella, Gianluca Reali, and Kris Steenhaut. 2021. Experimental analysis of the application of serverless computing to IoT platforms. Sensors 21, 3 (2021), 928.
20. DavidBermbach,SetarehMaghsudi,JonathanHasenburg,andTobiasPfandzelter.2020.Towardsauction-basedfunction placement in serverless fog platforms. In IEEE International Conference on Fog Computing (ICFC'20). IEEE, 25–31.
21. AnkitBhardwaj, ChinmayKulkarni, andRyanStutsman.2020.Adaptiveplacementforin-memorystoragefunctions. In USENIX Annual Technical Conference (ATC'20). 127–141.
22. Anirban Bhattacharjee, Yogesh Barve, Shweta Khare, Shunxing Bao, Aniruddha Gokhale, and Thomas Damiano. 2019. Stratum: A serverless framework for the lifecycle management of machine learning-based data analytics tasks. In USENIX Conference on Operational Machine Learning (OpML'19). 59–61.
23. Nilton Bila, Paolo Dettori, Ali Kanso, Yuji Watanabe, and Alaa Youssef. 2017. Leveraging the serverless architecture for securing Linux containers. In 37th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'17). IEEE, 401–404.
24. Yoni Birman, Shaked Hindi, Gilad Katz, and Asaf Shabtai. 2020. Cost-effective malware detection as a service over serverless cloud using deep reinforcement learning. In 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID'20). IEEE, 420–429.
25. Maria C. Borges, Sebastian Werner, and Ahmet Kilic. 2021. FaaSter troubleshooting–evaluating distributed tracing approaches for serverless applications. arXiv preprint arXiv:2110.03471 (2021).
26. Stefan Brenner and Rüdiger Kapitza. 2019. Trust more, serverless. In 12th ACM International Conference on Systems and Storage. ACM, 33–43.
27. Sebastian Burckhardt, Chris Gillum, David Justo, Konstantinos Kallas, Connor McMahon, and Christopher S. Meiklejohn. 2021. Serverless workflows with durable functions and netherite. arXiv preprint arXiv:2103.00033 (2021).
28. Changyuan Lin; Hamzeh Khazaei, Modeling and Optimization of Performance and Cost of Serverless Applications, IEEE Transactions on Parallel and Distributed Systems, 32, 3, 615-632, 2020.
29. Akash Puliyadi Jegannathan, Rounak Saha and Sourav Kanti Addya, A Time Series Forecasting Approach to Minimize Cold Start Time in Cloud-Serverless Platform, Networking and Internet Architecture, DOI: 10.48550/arXiv.2206.15176, 2022.
30. Deepak Yadav; Sarthak Maniar; Krish Sukhani and Kailas Devadkar, In-Browser Attendance System using Face Recognition and Serverless Edge Computing, 12th International Conference on Computing Communication and Networking Technologies (ICCCNT), 2021.
31. Jaeeun Cho and Younghan Kim, A Design of Serverless Computing Service for Edge Clouds, International Conference on Information and Communication Technology Convergence (ICTC), 2021.
32. Chetankumar Mistry; Bogdan Stelea; Vijay Kumar and Thomas Pasquier, Demonstrating the Practicality of Unikernels to Build a Serverless Platform at the Edge, IEEE International Conference on Multi-cloud Technology and Science (CloudCom), 2020.
33. Shivananda R. Poojara, Chinmaya Kumar Dehury, Pelle Jakovits and Satish Narayana Srirama, Serverless data pipeline approaches for IoT data in fog and multi-cloud, Future Generation Computer Systems, 130, 91-105, 2022.
34. Yaodong Huang; Zelin Lin; Tingting Yao; Xiaojun Shang; Laizhong Cui and Joshua Zhexue Huang, Mobility-aware Seamless Virtual Function Migration in Deviceless Edge Computing Environments, 42nd International Conference on Distributed Computing Systems (ICDCS), 2022.
35. Miao Zhang; Yifei Zhu; Jiangchuan Liu; Feng Wang; Fangxin Wang, CharmSeeker: Automated Pipeline Configuration for Serverless Video Processing. IEEE/ACM Transactions on Networking, 30, 6, 2730-2743, 2022.
36. István Pelle, Francesco Paolucci, Balázs Sonkoly, and Filippo Cugini, OFC Telemetry-Driven Optical 5G Serverless Architecture for Latency-Sensitive Edge Computing, Optical Fiber Communication Conference (OFC), 2020.
37. Trang Quang; Yang Peng, Device-driven On-demand Deployment of Serverless Computing Functions, IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)
38. Jaime Dantas; Hamzeh KhazaeiHamzeh Khazaei; Marin Litoiu, GreenLAC: Resource-Aware Dynamic Load Balancer for Serverless Edge Computing Platforms, Conference: CASCON '22 Proceedings of the 31st Annual International Conference on Computer Science and Software EngineeringAt: Toronto, Ontario, Canada, 2022.
39. Armir Bujari; Antonio Corradi; Luca Foschini; Lorenzo Patera; Andrea Sabbioni, Enhancing the Performance of Industry 4.0 Scenarios via Serverless Processing at the Edge, IEEE International Conference on Communications, 2021.
40. Kaustubh Rajendra Rajput; Chinmay Dilip Kulkarni; Byungjin Cho; Wei Wang; In Kee Kim, EdgeFaaSBench: Benchmarking Edge Devices Using Serverless Computing, IEEE International Conference on Edge Computing and Communications (EDGE), 2022.
41. Ta Phuong Bac; Minh Ngoc Tran; YoungHan Kim, Serverless Computing Approach for Deploying Machine Learning Applications in Edge Layer, International Conference on Information Networking (ICOIN), 2022.

42. Priscilla Benedetti; M. Femminella; G. Reali; Kris Steenhaut, Reinforcement Learning Applicability for Resource-Based Auto-scaling in Serverless Edge Applications, IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), 2022.
43. Anirban Das; Shigeru Imai; Stacy Patterson and Mike P. Wittie, Performance Optimization for Edge-Cloud Serverless Platforms via Dynamic Task Placement, IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), 2020.
44. Mark Szalay; Peter Matray; Laszlo Toka, Real-Time FaaS: Towards a Latency Bounded Serverless Cloud, IEEE Transactions on Multi-cloud, 2022.
45. Michele Ciavotta; Davide Motterlini; Marco Savi; Alessandro Tundo, DFaaS: Decentralized Function-as-a-Service for Federated Edge Computing, 10th International Conference on Cloud Networking (CloudNet), 2021.
46. Rafael Moreno-Vozmediano; Eduardo Huedo; Rubén S. Montero; Ignacio M. Llorente, Latency And Resource Consumption Analysis For Serverless Edge Analytic, Journal of Multi-cloud, 2022.
47. Neel Pradip Shah, Design of a Reference Architecture for Serverless IoT Systems, IEEE International Conference on Omni-Layer Intelligent Systems (COINS), 2021.
48. Onur Ascigil; Argyrios G. Tasiopoulos; Truong Khoa Phan; Vasilis Sourlas; Ioannis Psaras; George Pavlou, Resource Provisioning and Allocation in Function-as-a-Service Edge-Clouds, IEEE Transactions on Services Computing, 15, 4, 2410-2424, 2021.
49. Jananie Jarachanthan; Li Chen; Fei Xu; Bo Li, Astra: Autonomous Serverless Analytics with Cost-Efficiency and QoS-Awareness, IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2021.
50. Chetankumar Mistry; Bogdan Stelea; Vijay Kumar; Thomas Pasquier, Demonstrating the Practicality of Unikernels to Build a Serverless Platform at the Edge, IEEE International Conference on Multi-cloud Technology and Science (CloudCom), 2020.
51. Jaeeun Cho; Younghan Kim, A Design of Serverless Computing Service for Edge Clouds, International Conference on Information and Communication Technology Convergence (ICTC), 2021.