

Light weight serverless computing at fog nodes for internet of things systems

Mohamed Elkholy¹, Marwa A. Marzok²

¹Department of Computer Engineering, Faculty of Engineering, Pharos University in Alexandria, Alexandria, Egypt

²Information Technology Department Faculty of Specific Education, Matroh University, Matroh, Egypt

Article Info

Article history:

Received Oct 12, 2021

Revised Feb 4, 2022

Accepted Feb 15, 2022

Keywords:

Cloud computing

FaaS

IoT

Serverless computing

Software containers

ABSTRACT

Internets of things (IoT) systems collect large size of data from huge numbers of sensors. A wide range of IoT systems relies on cloud resources to process and analyze the collected data. However, passing large amount of data to the cloud affects the overall performance and cannot support real-time requirements. Serverless computing is a promising technique that allows developer to write an application code, in any programming language, and specify an event to start its execution. Thus, IoT system can get a good benefit of serverless environment. The proposed work introduces a framework to allow Serverless computing to take place on the Fog nodes near the data collectors. The proposed framework is implemented as an extension to a Kubernetes cluster that manages a set of Docker containers at the fog layer. A prototype of the proposed solution was implemented using Node.js for coding and YAML files to transfer data. The proposed framework was evaluated against traditional cloud Serverless execution. The experimental results proved the significant enhancement of the framework by decreasing the respond time especially for data intensive IoT applications.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Mohamed Elkholy

Department of Computer Engineering, Faculty of Engineering, Pharos University in Alexandria

Mohamed Elkholy Alexandria, Egypt

Email: Mohamed.elkholy@pua.edu.eg

1. INTRODUCTION

Smart cities are the themes of human future living. They are provided with large number of internets of things (IoT) sensors that collect data associated with human, homes, streets, and environment [1]. These sensors generate a massive amount of valuable data. To get a significant benefit of the collected data a set of calculations and artificial intelligence (AI) algorithms should be applied to it [2]. Several examples demonstrate the benefits of analyzing data generated by IoT sensors. For instance, a smart health care system can determine abnormal medical measurements of a person and provide him with the required medical aid [3]. Smart road monitoring cameras can collaborate with sensors embedded in vehicles to detect abnormal movement of vehicles or abnormal activities done by drivers to prevent accidents [4].

The first problem that faces IoT systems that the computational power needed to analyze the collected data is not available near IoT sensors [5]. Thus, the most likely scenario of IoT applications is passing the sensors' data to the cloud to get benefit of using high computational power [6]. Cloud computing provides high and flexible computational power needed by IoT systems. One of the most suitable cloud services for IoT systems is Serverless computing [7]. Serverless computing provides function as a service (FaaS) that adopt event-based programming paradigm [8]. Using FaaS allow developers split an application into coherent functions and each function defines an event that triggers its execution. Thus, IoT systems can get several benefits of using serverless computing [9]. Developers can implement IoT application codes

without any concern of resource management and passes the code to be executed as serverless functions at the cloud sides.

The idea of utilizing Serverless computing to serve IoT applications is widely discussed and implemented. From the market view Amazon, Azure, and IBM defined their own protocols to connect Serverless computing with IoT devices. Amazon defines Amazon Web Services (AWS IoT) that can control messages to and from IoT devices to use different AWS services [10]. Azure provides developers with the facilities and protocols to invoke Serverless functions using signals generated from IoT devices. IoT sensors send data to the Azure IoT Hub located at the cloud data center. IBM also declared the use of Open Whisk Serverless to be connected to IoT devices. However, there is a research gap associated with the delay time when executing a data-intensive Serverless function at the cloud side.

According to Tärneberg *et al.* [10] used AWS IoT Serverless computing to build a system for vehicle traffic control. Their work tried to reduce latency by using high computational power enabled in cloud data center. However, the communication between the cloud data center and the edge where IoT data are collected added a significant delay based on distance. According to Benedict [11] introduced the generic architecture of IoT-enabled serverless computing environment. The author discussed the possible deployment of Serverless functions at each level between IoT and cloud namely cloud server, fog level, edge level, and IoT devices. The system introduced by the user is not applicable, and a high computational power is lost in managing messages. Persson and Angelsmark [12] introduced Kappa framework that applies Serverless computing model for IoT systems. Their framework focused on creating a frontend interface with IoT devices to create representational state transfer application programming interface (REST API) request to cloud functions. The backend part at the cloud intercepts these REST calls and starts function execution. However, their work didn't discuss the effect of intensive data applications where a large size of data is passed to the cloud. According to Gusev *et al.* [13] proposed the applicability of using serverless functions on IoT devices to increase the scalability of streaming applications. They proposed a scheduling algorithm to find an edge device to deploy a Serverless function on it. However, edge devices have low computational power [14]. Sending huge size of data to be analyzed and calculated at the cloud provider arises several problems associated with performance, security, and bandwidth utilization [15]. Transferring huge size of data through network channels significantly increases response time. Such delay exceeds 2 sec for applications that use massive data size [14], [16]. Thus, IoT real-time applications suffer from several failures due to increasing response time by increasing data size [17]. Another problem of sending huge data size over Internet channels is associated with security and privacy. IoT systems include sensitive data such as medical records of patients [18]. Frequently sending such data to cloud is considered a security gap that an intruder can get benefit of it.

Thus, processing huge data for IoT application near data generators needs high computational power that is not available [15]. On the other hand, passing such huge amount of data to remote provider has several advantages of performance and security. Such problems lead to isolating of cloud facilities such as Serverless computing from being used in IoT application [19], [20].

The main contribution of the proposed work is to move high computational power near IoT sensors exactly at Fog nodes. Hence, we introduce fog serverless framework (FSF) that provides IoT systems with the ability to execute serverless functions at the Fog layer. The proposed framework provides developers with a flexible environment to write application code without concerning limitation in network or computational resource. According to the proposed framework IoT applications could be developed as a set of coherent functions (FaaS). Each individual function starts its execution according to a trigger signal sent from the edge layer according to data collected from IoT devices. Using serverless allows scaling up resources for several requests to the same function [21]. Thus, providing IoT application with the required performance and enhance the response time especially in real-time applications. To build an applicable Serverless system at the Fog layer it should be extended by the Serverless service offered by cloud provider. In case when the resources at the fog layer are overloaded by several requests to start functions, a number of requests can be passed to the cloud provider. Real-time functions are given the higher priority to be executed at the fog layer, followed by data-intensive functions. While time-intensive and resource-intensive functions are likely passed to the cloud.

The proposed framework forms an interface with Kubernetes cluster which manages a set of Docker containers. Docker containers allow execution of program code by encapsulating the code and its language image together in an isolated container [22]. A set of widely used language images are stored locally at the Fog storage to allow quick start of containers when a function trigger is received. Creating and initializing containers are managed by Kubernetes master node, however scheduling the execution of Serverless function is managed by FSF. When a request for executing a Serverless function is triggered from the edge layer, it is intercepted with FSF. Then according to the metadata associated with the function and the status of the Fog cluster, FSF takes the decision about the function execution.

The paper is organized as shown in: Section 2 represents the methodologies used to define, design and implement the proposed solution. Experimental evaluation is demonstrated in section three. While, section four concludes our work and presents the future work.

2. METHOD

The main objective of the proposed work is to create a serverless environment using fog nodes. The concept of serverless computing and IoT system shows that the two technologies are complementary. serverless offers scalable and huge amount of computing resources, while IoT suffers from limitation in the same aspect. Serverless computing adopts event-driven functions in which functions starts execution when a monitored event occurs [22]. Thus to allow applications to run on Serverless environment the application logic is built as function as a service (FaaS) [23]. In FaaS developers write the function code and specify an event that is responsible of triggering the execution of the function. IoT systems are compatible with such concept as IoT applications strongly depends at the signals and messages generated by IoT devices [18], [24]. Thus implementing a light weight Serverless using computer machines at Fog nodes has a significant impact on IoT systems.

2.1. Criteria of choosing fog layer to host serverless framework

Connecting IoT devices to cloud data centers have four main layers presented in Figure 1. The first consists of IoT devices (sensors and actuators) which are limited in computational power and data transfer protocols [25]. Thus, it is not applicable to apply serverless computing on the device layer. The second layer is the edge layer that gets benefit of 5G towers to collect data generated by IoT devices [26]. Edge layer consists of single board computers such as Raspberry Pi and is capable of performing data analyses to off load the transition of huge data size to cloud [27]. Edge layer is able to host Docker containers and can host Serverless computing, however it cannot provide the required scalability due to resource limitation.

The third layer is fog layer which extends the edge by servers with higher computational power and is supported with the use of virtual machines [28]. The proposed work concludes that the fog layer is suitable to deploy Serverless frame because fog environment is suitable to deploy Docker containers that can host Serverless functions. Fog nodes can be aggregated in a cluster to maximize resources utilization. Moreover, the proposed work uses fog virtual machines to get a reliable connection between geographically separated fog nodes to enhance scalability needed for serverless environment. The fourth layer is "cloud data centers" that provide huge scalable computation resources. However, sending large size of collected data to the cloud raises several issues in terms of delay and privacy.

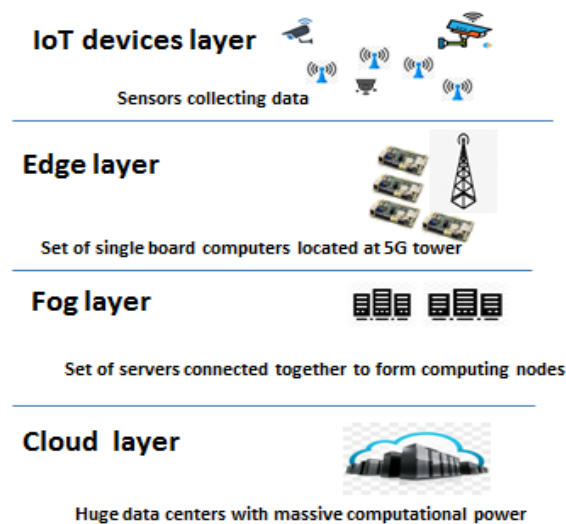


Figure 1. Four layers between IoT system and cloud

2.2. Framework design

This section demonstrates the novelty of the proposed research by introducing the design of fog serverless framework (FSF). FSF allows executing Serverless functions at the fog layer. The framework

aggregate fog devices to form a cluster that runs different services near the data generators. The design is introduced as a general design without any restrictions about implementation approaches to allow different implementation techniques to be applied to the framework. However, our implementation in the next section uses Kubernetes and Docker containers.

IoT devices collect data from the environment and pass it to the edge devices in message queuing telemetry transport (MQTT) protocol. Edge layer receives the continuous huge amount of raw data generated by IoT devices. Edge nodes are reasonable of analyzing and pre-processing raw data and detecting errors. After that edge nodes define the required Serverless function and passes data to the Fog layer in the form of HTTP request Figure 2.

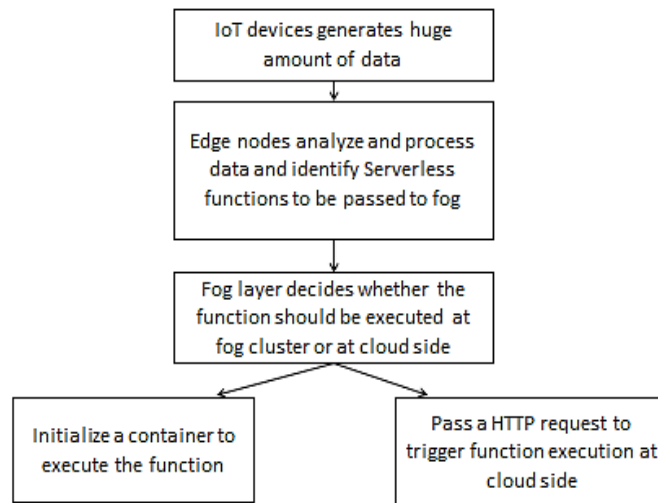


Figure 2. Overview of FSF functionality

Fog nodes are aggregated to form a cluster including one manager node and a set of working nodes. The manager node consists of three major parts, the first is an API gateway to communicate with the external layers, i.e. edge layer and cloud layer. The second part is the analyzer, which is responsible of taking a decision about execution a function at the fog cluster or at the cloud side. The third part is the scheduler that manages the execution of the Serverless function at the fog worker nodes.

The API gateway at the manager node receives the hypertext transfer protocol (HTTP) request including a defined Serverless function to be executed and its associated data and metadata. The proposed work defines a methodology of combining metdat, describing the function attributes, with each Serverless function. The metadata includes three main properties, the size of data associated with the function execution, the required computational resources, and whether the function is a real-time function or not. The function metadata is processed at the analyzer to decide whether the function will be executed at the fog layer or passed to the cloud side. Such decision depends on the size of data, resource requirements, and time constrains associated with the function.

If the function would be executed on the fog cluster, the analyzer passes its trigger to the scheduler. The scheduler initializes a software container to run the Serverless function. A container is initialized by assign a working node to host it and an internet protocol (IP) address to allow its communication. The manager node also provides the container with the required language image and dependencies to run the function code. However, in case that the requests exceed the computational resources available at fog cluster the request is passed to the cloud data center. In such cases the analyzer passes the request to the API gateway to sends a HHTTP request to the cloud provider to trigger the function execution.

Thus IoT applications have three different categories according to the proposed method; data intensive functions, resource intensive function, and real-time functions according to the function metadata. FSF provide a scheduler that gives higher priority to executed real-time functions and data intensive function on the fog cluster. While the resource intensive functions are passed to the cloud to benefit from the high resource's availability at cloud data centers. The architecture design of FSF and steps of function execution is clarified in Figure 3.

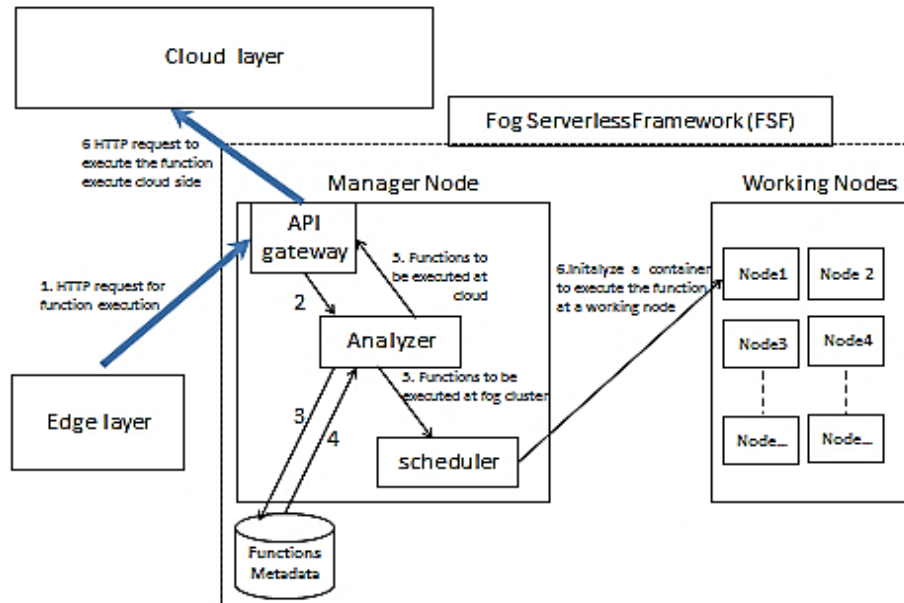


Figure 3. Architecture design of fog serverless framework at a fog cluster

2.3. Implementation of fog serverless framework (FSF)

The proposed design of FSF arranges fog nodes in a cluster with one manager node and a set of working nodes. Kubernetes was used to create and manage a cluster of fog nodes. One node is selected to be the control node that is responsible of receiving external requests and scheduling execution of functions on the worker nodes. Each worker node includes a set of pods that are the smallest unit that allows software container execution. At the top of each node, there is Kubelet service that monitors the status of the node pods and communicates with scheduler.

FSF is integrated to Kubernetes system, and extends its functionality by adding two main parts. The first is the manager node which controls the Kubernetes system and schedule the received HTTP requests for Serverless functions execution from the edge layer. The second part is a set node monitor (NM) that are located at each node and communicate with each node Kubelet. Each NM collects information about the node resources and the containers running in the node pods. NM communicates with the manager node to provide it with the updated information about each worker node. To allow executing Serverless functions at the fog cluster we used OpenFaaS. OpenFaaS allow packing function code with its image and dependencies in a Docker container. Figure 4 demonstrates the implementation of fog serverless framework.

Thus, IoT application logic is implemented as FaaS with a trigger event for each function. FSF also requires a metadata file for each function. The metadata file includes the size of data required for function execution and the required computational resource. It also specifies whether the function is real-time function or not. Functions metadata are listed in YAML file, Listing 1 shows an example for a function with title "road monitoring function". This function is a real-time function that requires 1 GB of data, and needs 512 MB of memory and four central processing units (CPUs) to allow parallel execution. Such information is stored at the FSF node manager to allow the analyzer to take decision about function s execution. As mentioned in the previous section the real-time intensive data applications have the first priority to be executed first. The manager node parses YAML files using node.js to get data associated with each function. Listing 2 presents an example of JavaScript code to parse the YAML file in Listing 1. JavaScript code uses "safeLoad" method to convert data in YAML file to JavaScript literals and objects in the same structure. Listing 3 presents the output after running the code in Listing 2.

Hence, IoT applications are defined as a set of functions with a metadata file for each function. The execution environment (code, language image, and dependencies) are stored as Docker containers. While the metadata associated with each function is stored as YAML file at the manager node. When a function trigger is received by the manager node its metadata file is parsed to get its proper priority. The analyzer uses functions metadata as well as resource information sent from NM to takes its decision.

The decision specifies whether to execute the function at the fog cluster or at the cloud side. Requests for high priority functions (data intensive, real-time) are scheduled to the Kubernetes control plane to assigns a pod to start the function execution inside a defined container. While the resource intensive functions are passed to the cloud side to be executed there.

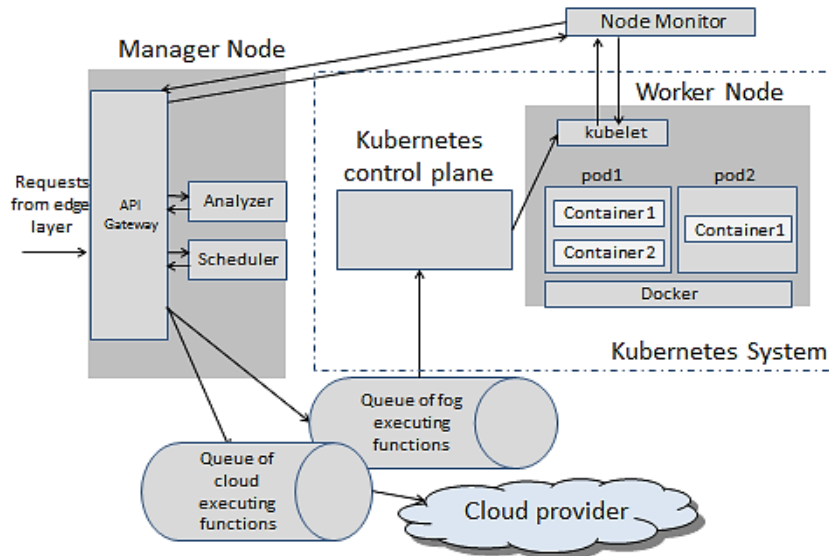


Figure 4. Implementation of fog serverless framework

Listing 1. Example of Function Metadata written in YAML file

```
title: "road monitoring function "
url path: "/reading -yaml-of-road monitoring function"
port: 443
is-https: true
metadata:
  DatasizeinMB: "1000"
  MemoryinMB: "512"
  CPUNumber: "4"
  Real-time: "true"
tags:
- javascript
- node.js
```

Listing 2. Reading metadata for a function using node.js at the edge interface

```
const fs = require('fs');
const yaml = require('js-yaml');

try {
  let fileContents = fs.readFileSync('./road monitoring function.yaml', 'utf8');
  let metadata = yaml.safeLoad(fileContents);

  console.log(metadata);
} catch (e)
{
  console.log(e);
}
```

Listing 3. The function metadata serialized as object in a JavaScript file

```
$ node read.js
{ title: 'Reading YAML File for road monitoring function.js/JavaScript',
  'url path': ' reading -yaml-of-road monitoring function ',
  port: 443,
  'is-https': true,
  meta:
  { 'published-at': 'OCT. 1st, 2021',
    metadata: { DatasizeinMB: "1000", MemoryinMB: "512", CPUNumber: "4"},
    tags: [ 'javascript', 'node.js']
  }
}
```

3. RESULTS AND DISCUSSION

The main contribution of the proposed framework is providing Fog layers with a suitable environment to execute Serverless computations. Thus the framework maintains short respond time required by real-time application. To clarify the efficiency of the proposed framework, the evaluation focuses on data

intensive Serverless functions that include large size of data. A set of 5 functions is used as FaaS by applying their code and dependencies and triggering events. The functions use machine learning (ML) algorithms to extract different features from images. The functions were triggered to be executed once at the cloud side and the other time at the fog layer. The response time were calculated in both cases at different situations.

3.1. Setting up the environment

A huge number of images were collected from distributed cameras and used as input data. Then the set of the five serverless functions perform different calculations and machine learning functions on the captured images. The serverless functions are invoked in two different ways. Once by passing the data of captured images to the cloud and execute the Serverless function at the cloud side. The other time the functions are executed at the fog layer near the data generating source.

3.1.1. Setting the traditional serverless execution at cloud side

The produced images are passed using HTTP to a server that simulates the traditional fog layer to allow calling the Serverless functions from the cloud side. The used server was Dell server with core i7 CPU and 64 GB of memory. A Node.js application was implemented on the server to send HTTP request for triggering Serverless functions execution at the cloud side. The application is also responsible of sending the required images as input data in json file to be stored in Amazon S3. The ML functions were executed as AWS lambda functions using FaaS model. Thus, such implementation the scenario of trigger Serverless functions to be executed at the cloud side and the collected IoT data is sent out to the cloud provider.

3.1.2. Setting FSF execution at the fog layer

To allow executing serverless functions at the fog layer, a cluster is configured using one server acting as a master node and 5 other servers used as working nodes. All the machines used Ubuntu as the operating system. Kubernetes v1.17 was used to manage the cluster and allow splitting each worker node to pods to run containerized functions. Docker-engine v19.03 was used to create containers. The five ML functions and passed to be executed in Docker containers include the required dependencies. The proposed extension to the Kubernetes system is implemented using Node.js and use YAML file to transfer metadata as discussed in Section 3.4.

3.2. Experimental results

To clarify the benefits of the proposed framework it is essential to compare response time of traditional IoT application with IoT applications executed as fog serverless functions. The selected functions are data intensive functions to reflect the real needs of IoT applications. The experiments were done by passing the data collected from the machine that simulated IoT sensors to be executed in the five Serverless functions mentioned in 3.1 once at the cloud and the other time at the fog cluster. Different frequencies of functions execution were applied to get a clear image of performance.

- Each function is triggered to be executed one time per minute.
- Each function is triggered to be executed one time per second.
- Each function is triggered to be executed 60 times per second.

The criteria of choosing these intervals are to trace the delay of warm startup in which the containers that run the functions are not initialized. Then the frequency of triggering the functions increases to demonstrate the scalability of the framework until reaching the highest rate of popular IoT cameras which is 60 Fps. Each one of the three sequences is traced for 10 minutes and the average response time is calculated for both the two approaches. The experimental results are presented in Table 1.

Table 1. Average of response time in (msec) for function invocation during 10 minutes

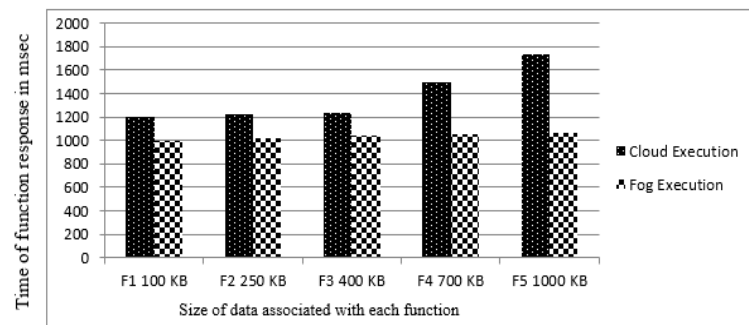
Function name	Cloud execution					Fog cluster execution				
	F1	F2	F3	F4	F5	F1	F2	F3	F4	F5
Function Data size in KB	100	250	400	700	1000	100	250	400	700	1000
Average response time for one request per minute	1200	1220	1230	1500	1730	1000	1020	1040	1050	1070
Average response time for one request per second	300	420	550	1320	1480	140	220	280	310	370
Average response time for 60 requests per second	290	410	520	1300	1430	140	210	270	300	360

3.3. Discussion

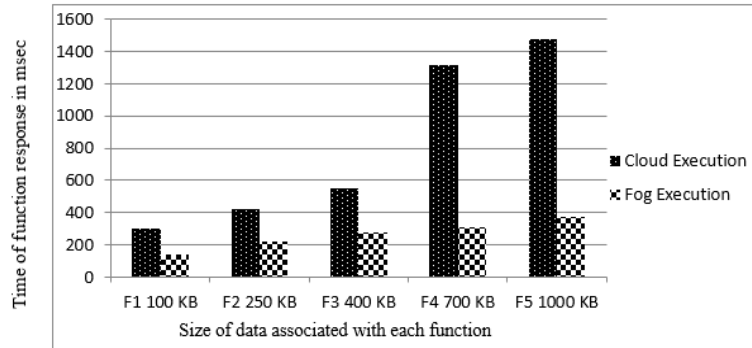
Table 1 demonstrates the performance enhancement for executing serverless functions at the fog clusters. The response time for function decreased significantly when the functions are executed at the fog. The enhancement increases as the size of data associated with the function increases.

Figure 5 illustrates the proposed work contribution by representing the response time for executing different functions at fog clusters and at the cloud side. Figures 5(a)-(c) represent the response time for different functions across different data sizes and execution environments. Figure 5(a) presents the response time for each function when requested one time per minute. The response times for executing F1, F2, F3, F4, and F5 on the cloud are almost the same, although they differ in data size. The reason behind this is the cold start latency, in which warming up the container dominates the response time. While for F4 and F5, the large size of data transferred extends the warming up time and increases the delay. For fog execution, when functions are executed once a minute, they all experience the cold start delay, and their response times are almost the same. However, fog execution presents lower response times compared with cloud execution for all functions.

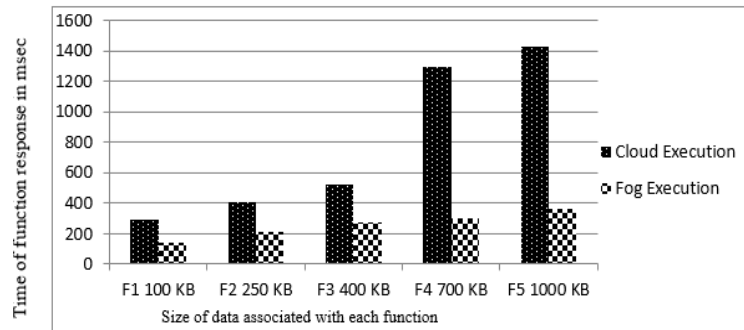
Figure 5(b) presents the response time for each function when requested one time per second. The significant enhancement of response time for the framework is clear as functions are executed on the fog without sending data out to the cloud. The performance enhancement of the framework increases significantly with increasing the data size associated with the function execution. Figure 5(c) presents the response time for each function when requested 60 times per second. The values are very close but smaller than the values of Figure 5(b) as the increasing in the frequency of function execution increases its chance of warm start.



(a)



(b)



(c)

Figure 5. Average of response time for the five Serverless functions for 10 minutes duration, (a) when requested one time per minute, (b) when requested one time per second, and (c) when requested 60 times per second

4. CONCLUSIONS

Serverless computing provides IoT system with the required resource provision and management to run IoT applications. A bottle neck appears when executing an intensive data serverless function on the cloud as large size of data should be transferred from IoT devices to the cloud side. A promising framework FSF is introduced to eliminate the bottle neck by executing Serverless functions at the fog layer. The proposed framework extends Kubernetes system to create a suitable environment to execute data intensive Serverless functions at the fog layer. This work proposed the design and implementation of the framework. The performance of the framework was evaluated against the traditional Serverless execution. The experimental results clarified the enhancement of response time by using the proposed framework to execute data intensive Serverless function at the fog layer. Thus the proposed framework provides IoT system with the ability to executing real-time data intensive application as Serverless functions. Our future work will be concerned with predicting the incoming requests to execute functions depending on the execution history and environment conditions. Thus, the cluster performance could be increased by warming up the relevant container before receiving the incoming request.




REFERENCES

- [1] Y. Hajjaji, W. Boulila, I. R. Farah, I. Romdhani, and A. Hussain, "Big data and IoT-based applications in smart environments: A systematic review," *Computer Science Review*, vol. 39, p. 100318, Feb. 2021, doi: 10.1016/j.cosrev.2020.100318.
- [2] P. C. Siswipraptini, R. N. Aziza, I. Sangadji, Indrianto, R. R. A. Siregar, and G. Sondakh, "IoT for smart home system," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 23, no. 2, pp. 733–739, Aug. 2021, doi: 10.11591/ijeecs.v23.i2.pp733-739.
- [3] M. Fotouhi, M. Bayat, A. K. Das, H. A. N. Far, S. M. Pournaghi, and M. A. Doostari, "A lightweight and secure two-factor authentication scheme for wireless body area networks in health-care IoT," *Computer Networks*, vol. 177, p. 107333, Aug. 2020, doi: 10.1016/j.comnet.2020.107333.
- [4] L. Campanile, M. Iacono, A. H. Levis, F. Marulli, and M. Mastroianni, "Privacy regulations, smart roads, blockchain, and liability insurance: Putting technologies to work," *IEEE Security and Privacy*, vol. 19, no. 1, pp. 34–43, Jan. 2021, doi: 10.1109/MSEC.2020.3012059.
- [5] O. Alzakholi, L. Haji, H. Shukur, R. Zebari, S. Abas, and M. Sadeeq, "Comparison Among Cloud Technologies and Cloud Performance," *Journal of Applied Science and Technology Trends*, vol. 1, no. 2, pp. 40–47, Apr. 2020, doi: 10.38094/jastt1219.
- [6] Z. Liu, Z. Wu, C. Gan, L. Zhu, and S. Han, "DataMix: Efficient privacy-preserving edge-cloud inference," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12356 LNCS, 2020, pp. 578–595.
- [7] T. Alam and M. Benaida, "Blockchain, fog and IoT integrated framework: review, architecture and evaluation," *SSRN Electronic Journal*, vol. 62, no. 2, 2020, doi: 10.2139/ssrn.3639001.
- [8] M. Shahrad *et al.*, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," *Proceedings of the 2020 USENIX Annual Technical Conference, ATC 2020*, 2020, pp. 205–218.
- [9] A. M. Potdar, D. G. Narayan, S. Kengond, and M. M. Mulla, "Performance Evaluation of Docker Container and Virtual Machine," *Procedia Computer Science*, vol. 171, pp. 1419–1428, 2020, doi: 10.1016/j.procs.2020.04.152.
- [10] W. Tärneberg, V. Chandrasekaran, and M. Humphrey, "Experiences creating a framework for smart traffic control using AWS IOT," *Proceedings - 9th IEEE/ACM International Conference on Utility and Cloud Computing, UCC 2016*, 2016, pp. 63–69, doi: 10.1145/2996890.2996911.
- [11] S. Benedict, "Serverless blockchain-enabled architecture for IoT societal applications," *IEEE Transactions on Computational Social Systems*, vol. 7, no. 5, pp. 1146–1158, Oct. 2020, doi: 10.1109/TCSS.2020.3008995.
- [12] P. Persson and O. Angelsmark, "Kappa," in *Proceedings of the 2nd International Workshop on Serverless Computing*, Dec. 2017, pp. 16–21, doi: 10.1145/3154847.3154853.
- [13] M. Gusev *et al.*, "A deviceless edge computing approach for streaming IoT applications," *IEEE Internet Computing*, vol. 23, no. 1, pp. 37–45, Jan. 2019, doi: 10.1109/MIC.2019.2892219.
- [14] A. Tolba, "A two-level traffic smoothing method for efficient cloud-IoT communications," *Peer-to-Peer Networking and Applications*, vol. 14, no. 5, pp. 2743–2756, Sep. 2021, doi: 10.1007/s12083-021-01106-5.
- [15] M. Kumar, M. Di Francesco, and S. N. Srirama, "Serverless computing for the Internet of Things," Master Thesis, Aalto University, 2018.
- [16] T. Rausch, A. Rashed, and S. Dustdar, "Optimized container scheduling for data-intensive serverless edge computing," *Future Generation Computer Systems*, vol. 114, pp. 259–271, Jan. 2021, doi: 10.1016/j.future.2020.07.017.
- [17] A. Kumari, B. Sahoo, R. K. Behera, S. Misra, and M. M. Sharma, "Evaluation of integrated frameworks for optimizing QoS in serverless computing," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12955 LNCS, 2021, pp. 277–288.
- [18] M. Manca, Fabio, Paternò, C. Santoro, and L. Corcella, "Supporting end-user debugging of trigger-action rules for IoT applications," *International Journal of Human Computer Studies*, vol. 123, pp. 56–69, Mar. 2019, doi: 10.1016/j.ijhcs.2018.11.005.
- [19] C. G. García, D. M. Llorián, C. P. G-Bustelo, and J. M. Cueva-Lovelle, "A review about smart objects, sensors, and actuators," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 4, no. 3, p. 7, 2017, doi: 10.9781/ijimai.2017.431.
- [20] S. Tang, D. R. Shelden, C. M. Eastman, P. P.-Bozorgi, and X. Gao, "A review of building information modeling (BIM) and the internet of things (IoT) devices integration: Present status and future trends," *Automation in Construction*, vol. 101, pp. 127–139, May 2019, doi: 10.1016/j.autcon.2019.01.020.
- [21] Y. Chen, C. Li, L. Gong, X. Wen, Y. Zhang, and W. Shi, "A deep neural network compression algorithm based on knowledge transfer for edge devices," *Computer Communications*, vol. 163, pp. 186–194, Nov. 2020, doi: 10.1016/j.comcom.2020.09.016.
- [22] M. I. Khaleel and M. M. Zhu, "Adaptive virtual machine migration based on performance-to-power ratio in fog-enabled cloud data centers," *Journal of Supercomputing*, vol. 77, no. 10, pp. 11986–12025, Oct. 2021, doi: 10.1007/s11227-021-03753-0.




- [23] Z. Ageed, M. R. Mahmood, M. M. A. Sadeeq, M. B. Abdulrazzaq, and H. Dino, "Cloud computing resources impacts on heavy-load parallel processing," *IOSR Journal of Computer Engineering (IOSR-JCE)*, vol. 22, no. 3, pp. 30–41, 2020.
- [24] Y. A. A. S. Aldeen and H. M. Abdulhadi, "Data communication for drone-enabled internet of things," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 22, no. 2, pp. 1216–1222, May 2021, doi: 10.11591/IJEECS.V22.I2.PP1216-1222.
- [25] A. Almomani, A. Al-Nawasrah, W. Alomoush, M. Al-Abweh, A. Alrosan, and B. B. Gupta, "Information management and IoT technology for safety and security of smart home and farm systems," *Journal of Global Information Management*, vol. 29, no. 6, pp. 1–23, Nov. 2021, doi: 10.4018/jgim.20211101.0a21.
- [26] M. Sarrab, S. Pulparambil, and M. Awadalla, "Development of an IoT based real-time traffic monitoring system for city governance," *Global Transitions*, vol. 2, pp. 230–245, 2020, doi: 10.1016/j.glt.2020.09.004.
- [27] S. Ren, J. S. Kim, W. S. Cho, S. Soeng, S. Kong, and K. H. Lee, "Big data platform for intelligence industrial IoT sensor monitoring system based on edge computing and AI," in *3rd International Conference on Artificial Intelligence in Information and Communication, ICAIIC 2021*, Apr. 2021, pp. 480–482, doi: 10.1109/ICAIIIC51459.2021.9415189.
- [28] X. Qi and C. Liu, "Enabling deep learning on iot edge: approaches and evaluation," in *Proceedings - 2018 3rd ACM/IEEE Symposium on Edge Computing, SEC 2018*, Oct. 2018, pp. 367–372, doi: 10.1109/SEC.2018.00047.

BIOGRAPHIES OF AUTHORS



Mohamed Elkholy    received the B.Sc. degree in computer engineering and from Military Technical college, Cairo, Egypt in 1995. He received M.Sc. degrees in computer engineering from Arab Academy for Science and Technology Alexandria, Egypt, in 2011 and the Ph.D. degree in Information Technology from Alexandria University, Egypt, in 2016. He has been an associated professor of computer engineering with Pharos University in Alexandria, since 2017. His current research interests include Service-Oriented Architecture and its applications such as Web services, Microservices. His interests also include cloud computing, software containers, and machine visualization. He has published 10 journal and conference papers in the fields of software engineering. He is reviewer in several journals. He can be contacted at email: Mohamed.elkholy@pua.edu.eg.



Marwa A. Marzok    received the B.Sc. degree in Science and Education majoring in Physics from Alexandria University, Alexandria, Egypt in 2000, and M.Sc. degrees in Information Technology from Alexandria University, Alexandria, Egypt, in 2011. She received the Ph.D. degree in Information Technology from Alexandria University, Alexandria, Egypt 2017. She worked as Associated Professor in Alexandria University from 2017 to 2020. Now she is with in Matroh Univeristy. She has valuable publications in the field of AI and decision making. Her area of interest is IoT, cloud computing and image processing. She can be contacted at email: mabelazee@nctu.edu.eg.