Serverless Functions Allocation in IoT using Machine Learning

Laman Aghabayova (19041277)
Article: 5616 words

A thesis presented for the degree of
MSc in Computing

School of Computing
Teesside University Middlesbrough TS1 3BA
Supervisor: Bohuˇs Z´ıskal
24.08.2020

List of Figures

Serverless Functions Allocation in IoT using Machine Learning

L.Aghabayova

Prague, Czech Republic

Abstract

The IoT (Internet Of Things) represents a set of technologies that helps to create a network of smart devices. These devices can transfer data between each other or send it to the next layer in the network. According to IDC n.d., the number of connected devices will grow up to 41.6 billion by 2025, which leads to significant market growth possibilities. Initially, the IoT has faced high latency problems by using only cloud computing solutions.

Fog computing has recently emerged as an extension for cloud computing, which aims to move computa tional tasks closer to the edge. The introduction of serverless computing has produced new IoT challenges related to the allocation of the functions between the fog and cloud layers. Thus, the article proposes a prediction-based framework that will choose where each function will be executed based on its execution history. The decision will be made using the machine learning model, which will be trained and tested based on the input features.

Keywords: Machine Learning, Serverless computing, Internet Of Things

1. Introduction

The Internet of Things (IoT) aims to expand the possibilities of the Internet for collecting and processing a large amount of data. The IoT technologies join billions of internet-enabled sensors for continuous data exchange and further analysis. As stated in Koniagina et al. 2020, the global market of IoT has been shared between four main sectors: Smart cities (26%), Health (20%), Agriculture (14%). By using smart devices, the given industries can monitor and automate their processes to enhance customer experience and save time and costs. The IoT ecosystem has been divided into four essential parts:

1. *Sensors* are the backbone of the IoT environment that is accountable for collecting and processing data.
2. *Protocols* helps to build a communication channel between the devices and the cloud. The primary accountability of the protocols is to transfer collected data within diverse layers of the IoT network.
3. *IoT Gateway* is the layer between the devices and the cloud. All data coming from the sensors have to go through the IoT Gateway and then reach the cloud.
4. *IoT Cloud* receives the accumulated data and performs complex operations and analysis to make cor responding decisions. Usually, the cloud servers are located far from the end devices, which produces an increase in the request delay.

The concepts cloud computing and IoT are operating together to provide a platform for collecting and processing the data, which is called Cloud Of Things. However, as stated by Jahantigh et al. 2019, cloud computing can produce performance issues when dealing with high volume data. One of the problems with cloud computing is the processing and computing time, which causes real-time analysis difficulties. Further, the traditional-cloud based IoT systems may encounter high latency or downtime issues that may not be

URL: http://www.elsevier.com (L.Aghabayova)

permissible for the IoT devices that need a quick response time. As a result, in January 2014, Cisco intro duced a new concept of Fog Computing, which brings cloud computing capabilities closer to the edge devices Solutions 2015.

Fog Computing is a decentralized framework where the data is processed and analyzed at the edge of the network. It is essential to specify that the Fog Computing can not be offered as a replacement for cloud computing but rather as an extension of the existing network. The critical components in the Fog architecture are fog nodes that have a hierarchical structure. The fog nodes are arranged close to the user, and this data can be managed locally instead of transferring to the cloud. The given fog nodes have more processing power than the end devices and can perform tasks locally to decrease the amount of data sent to the cloud.

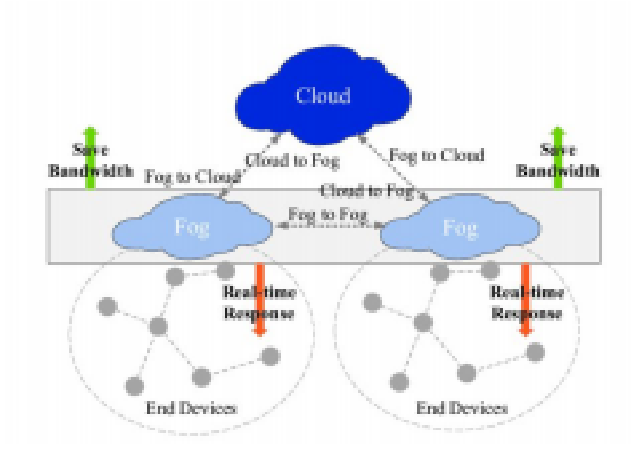


Figure 1: Fog Layer

The FaaS (Function-as-a-Service) is another promising architecture for using both in Fog and Cloud layers. As stated by Adhikari, Amgoth, and Srirama 2019, the primary goal of serverless architecture is to gener alize management and capacity planning decisions from the developers. In such architecture, the resource pricing is calculated based on the number of resources the application has used compared with other types of cloud computing that require pre-purchasing of resources. The essential component in serverless computing is serverless functions that enable the users to execute independent modular chunks of functionality in the cloud. In comparison with microservices, the serverless architecture splits the monolith project into small executable pieces which can be scaled dynamically.

However, because both Fog and Serverless computing are relatively new technologies, there are open issues related to the allocation of the functions between the Fog and Cloud layers. In such a case, the developers have to analyse the history of all serverless functions and, based on that, decide where the serverless function should be executed. Most of the time, it is a time consuming and inaccurate process, because the relationship between parameters influencing the response time is considered intricate.

Thus, the article introduces a new framework for a serverless allocation between the layers using a back propagation algorithm. The proposed framework will firstly generate a dataset based on the configured FaaS environment and then determine which VM parameters are affecting the response time. The framework's essential outcome is a decision-making model that developers can use to decide where to execute the serverless functions based on the history of execution.

2. Related Works

There are multiple resource allocation techniques that exist in cloud computing. A new optimization approach for Cloud computing has been introduced in a paper by Choi and Y. Lim 2016. The authors in the given work take into consideration SLA (Service Level Agreement), which is established between the service provider and the user. The whole algorithm is based on the auction system, where each user makes a bid for a VM bundle within a particular time interval.

Another resource allocation method has been proposed by Deng et al. 2016, which aimed to decrease the transmission latency between the Fog and Cloud layers. The author investigates the possible balance between the delay in resource transmission and power consumption of the fog and cloud nodes. The paper proves that the fog layer is possible for executing various computing services with lower latency compared to the cloud layer.

Lee, Saad, and Bennis 2017 specifies an algorithm for a task distribution between the fog and cloud nodes. The central idea behind the proposed framework is to build fog networks with chosen fog nodes to minimize the latency of tasks requested by the user. Abouaomar et al. 2019 also studied the resource allocation possibilities between the fog and cloud layers based on the device configurations.

The article was written by Alli and Alam 2020 describes the architecture of Fog computing and the new opportunities that can be achieved with it. The authors look deeply into the ecosystem of the IoT-Fog Cloud environment and open challenges such as the computational offloading, which will be investigated in the future sections. In Sarkar et al. 2019, the authors proposed a framework for the usage of serverless computing in smart buildings. The paper outlines the experiment conducted with IoT devices and one of the FaaS platforms to prove that the fog nodes have less latency than the cloud nodes. However, one of the drawbacks of the proposed solution is that the developers need to manually determine the environment for the execution of the functions (either cloud or fog).

The paper introduced by J. Li et al. 2017 also describes the opportunities for using deep learning methods in the fog layer. The authors focus on explaining the advantages and disadvantages of applying ML algorithms in the fog layer instead of the cloud. The article also contains the examples and discussions about frameworks that use the ML algorithms directly in the IoT devices. The paper written by L. Li, Ota, and Dong 2018 describes the advantages and disadvantages of applying ML algorithms in the fog layer instead of the cloud. The article continues with the discussion of frameworks that use the ML algorithms in the IoT devices.

Even though there are already researches related to the tasks offloading between the fog and the cloud, there is still a gap in the knowledge. The gap can especially be perceived in the possibilities of usage serverless and fog computing together as both of the technologies are relatively new. Thus, the paper proposes the machine-learning-based prediction framework that predicts the response time of functions and considers the VMs configurations to make predictions more accurate.

3. Project Specification

As described in the previous sections, with the fog layer's introduction, it became feasible to execute serverless functions straight in the fog nodes without transmitting the request to the cloud. However, because the fog nodes' capabilities are limited, it is sometimes faster to forward the request to the cloud. Fig. 2 illustrates the design of the proposed system.

The primary step in the experiment is to transfer the request for the execution of the serverless function from IoT device. As shown in Fig. 2, the primary step in the experiment is to transfer the request for the execution of the serverless function from IoT device. The project uses the Node-Red platform (described in section 7) for simulating the IoT environment. When the gateway receives the request, the function called ”Intercepter Function” will intercept the request to determine which environment is better for the execution of the function.

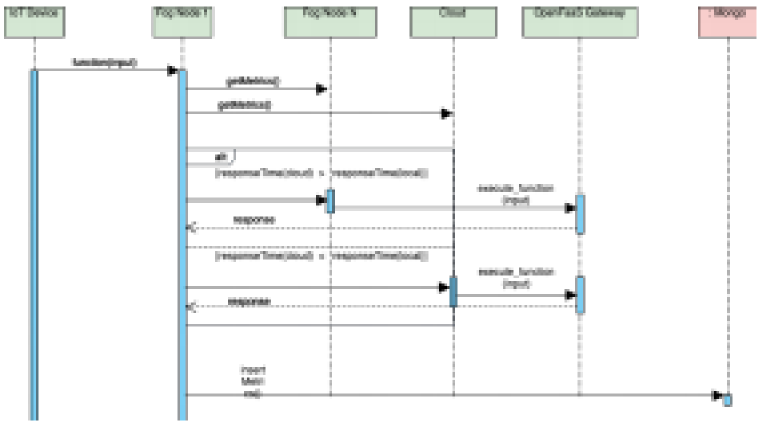


Figure 2: Sequential Diagram of the project

For determining each virtual machine's current state, the intercepter function gets metrics each of them by using method getMetrics(). The returned metrics are then used in our machine learning model for predicting the response time of the function. The intercepter function compares the response time across all virtual machines and selects with the lowest response time.

3.1. Project Requirements

For the simulation of the IoT network, the project follows architectural requirements specified by Cisco 2015.

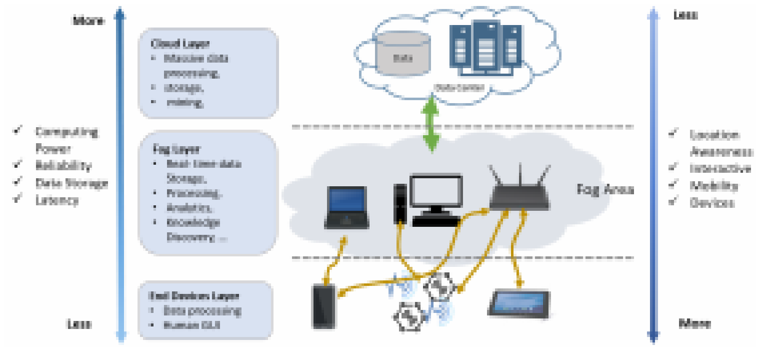


Figure 3: Source Margariti, Dimakopoulos, and Tsoumanis 2020

As shown in Fig. 3, the architecture should consist of three main layers:

1. End-device Layer
2. Fog Layer
3. Cloud Layer

The Fog layer consists of end-devices with low-computational power and a set of fog and cloud nodes that offer processing, storage, or analytics capabilities. The simulating environment's overall infrastructure can be represented as a graph with vertices that represent IoT devices and the edges that connect them between the layers. The characteristics of the IoT devices may be different depending on the requirements. According to Singh and Baranwal 2018, the Quality of Service (QoS) metrics has to be identified in order to evaluate the expected model proposed in the following sections. Although there are many QoS metrics currently existing, the essential evaluation metrics for the project are the following:

- Jitter represents a packet delay that may vary between the data transmission. The jitter can be a result of network congestion or improper network configuration.

- Bandwidth identifies the amount of data that can be transferred within a specific amount of time. It is advisable to have high bandwidth to achieve better data manipulation.

- Connection time. The connection timeout to the server has to be specified and written in the configuration.

- Storage. There are diverse types of storage used in the fog nodes based on the data volume and tasks that need to be processed before sending it to the cloud. As stated by Ai, Peng, and Zhang 2018, the fog layer commonly consists of three levels: data collection, data filtering, and more complex data operations. Further simulation has to be performed in order to identify the necessary storage capacity for the nodes.

- Computational Power. As stated by Puliafito et al. 2020, any device that has computational capa bilities can be added to the fog layer. The device's computational power may vary depending on its requirements (monitoring, processing, or storing data). According to Hub 2017, the devices such as Raspberry Pi 3 or Android Things OS are suitable for running the fog nodes.

- Device Communication. Because the fog nodes communicate both with end-devices and the cloud layer, it is necessary to consider the communication between them. According to Wardana and Perdana 2018, the commonly used protocol for communication is called Message Queuing Telemetry Transport (MQTT) protocol. The given protocol works based on the publish-subscribe mechanism that publishes events to the MQTT broker. Thus, the given parameter requirement is to start an MQTT client to simulate the communication between devices.

3.2. Serverless Functions used in the project

For the experiment, it is essential to define the serverless functions and deploy them to each specified node. It is also crucial to mention that the functions used in the experiment are purely research-oriented and are not the same as in the real environment. The primary goal for the functions is to receive and perform appropriate operations that are being processed with various time based on the metrics of the machine. The proposed model contains the following functions:

1. *Image-processing-function*. The given function is written in Python, and the main algorithm behind it is to receive the random image with salt and pepper noise and to remove the noise from the images by using a corresponding filter. It is essential to mention that the algorithm follows the architecture specified in Rupani et al. 2017, which describes the techniques for image processing in the IoT environment.

As a result, the project contains two types of *Image-processing-functions* with contraharmonic mean fil tering and midpoint noise filtering.

2. *Sensor-data-function*. The given function accepts the JSON file, which consists of sensor data gener ated by the IoT simulation platform. The function sorts the received data by a field called "timestamp" and saves sorted data to the appropriate database.

3. *Intercepter-function*. The given function is responsible for making the final decision where the request should be executed. As specified in Fig. 2, the initial step for the *Intercepter-function* is to get metrics of all VMs at a given time. In the real environment, both Fog and Cloud layers are encountered with network latency. Thus, for achieving the latency in the request, the *Intercepter-function,* simulates the delay when sending requests to the serverless functions. The delay estimation for the given function has been taken according to J. Li et al. 2017. The Fig. 4 represents the final algorithm of the *Intercepter-function* as a Flowchart.

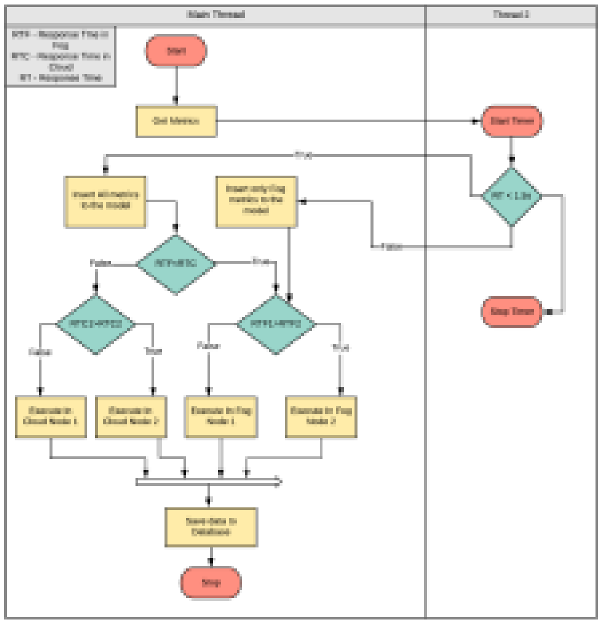


Figure 4: Flowchart of *Intercepter-function*

3.3. Features Engineering

The feature selection is the process of extracting the features with the highest correlation rate. The feature selection is one the most critical process when building a machine learning model, as it can either speed up or slow down (in case of unmodified data) the training.

For the evaluation of the features, it was decided to choose Pearson, which is a widely used method to measure the strength of the relationship between two variables X and Y. The Pearson correlation coefficient *r*, takes values within a range from +1 to -1.

Table 1: Pearson Correlation

Strength of Association Negative

Positive
.1 to .3
.3 to .5
.5 to 1.0

Small -0.1 to -0.3 Medium -0.3 to -0.5 Large -0.5 to -1.0

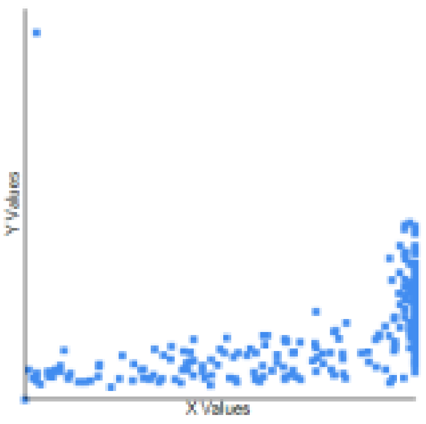
The following formula will be used for calculating the Pearson *r* correlation:

$$r_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - \left(\sum x_i\right)^2}\sqrt{n\sum y_i^2 - \left(\sum y_i\right)^2}} \quad (1)$$

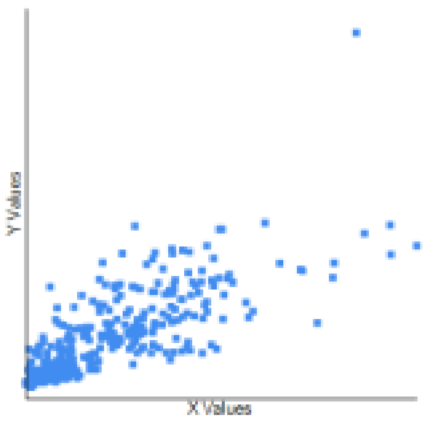
where, r_{xy} - Pearson *r* coefficient; n - number of observations; x_i - ith observation for X variable; y_i - ith observation for Y variable

Because the Pearson *r* correlation changes according to the number of observations, it is crucial to split the features selection into several iterations.

The Fig. 5a and 5b illustrate the relationship between diverse metrics and the response time. As can be seen from Fig. 5b, the association between the input size of the request and the response time is robust.



(a) Relationship between CPU Percent and Response Time



(b) Relationship between Input Size and Response Time

After the multiple iterations, it was decided to choose the following metrics as the input parameters for the proposed model:

• CPU Percent

• Docker CPU

• Docker Memory

• Input Size

• Bandwidth

• Processes Running

• Network Speed

3.4. Modelling

The proposed model will use a *backpropagation algorithm* for training a neural network model. The backpropagation was chosen because it ensures lower error rates and makes the model more reliable by adjusting weights based on the results from previous iterations. The main idea behind the backpropagation is to compute the gradient of the loss function with respect to their weights at each layer and then to iterate backward from the last layer in order to reduce the error rate.

3.5. Neural Network Structure

Neural Network is one of the techniques used in Deep Learning for predicting target segment data. In contrast with supervised learning, the deep learning architecture consists of more that one layer that can consist of both labeled and unlabeled data. In each layer, the weights matrix is modified using the training data, thus making the prediction stronger.

As can be seen from Fig. 6, the neural network consists of three main layers: input, hidden, and output layers. The input layer is built at the beginning of the workflow and is responsible for processing data and features of the dataset. The intermediate layer between the input and output layers is called the hidden layer. The hidden layer is a mathematical function that provides the output, which is then sent as an input to the next layer. The main goal of the hidden layer is to reduce the training error closer to zero.

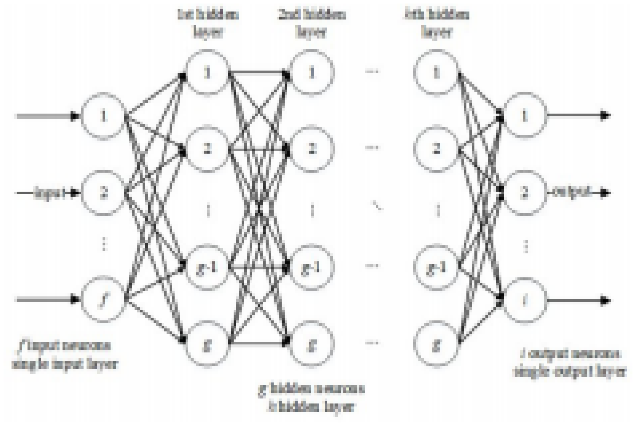

1st hidden layer
2nd hidden layer
kth hidden layer
input
output
f input neurons single input layer
g hidden neurons k hidden layer
l output neurons single output layer

Figure 6: Neural Network Structure Ooi, M. H. Lim, and Leong 2019

3.5.1. Backpropagation

Because the experiment works with a relatively small dataset, according to Shaikhina and Khovanova 2017, the backpropagation is one of the possible choice when working with a limited data. Backpropagation is an algorithm for minimizing the cost function by adjusting the weights and biases in the neural network. The algorithm computes the weights from the last layer and moves backward until it will reach the first layer. The main element in the backpropagation is the partial derivative of the loss function and any weight (or bias b) of the neural network. The given algorithm can be divided into the following steps:

1) Feed-forward. In the given step, the hidden layers calculate the weighted sums of the inputs and pass the results to the sigmoid activation function.

$$Z_m = W_m * X^T + b_m$$

$$A_m = \sigma(Z_m)$$

- Z_m represents the cost function of mth node
- W_m and b_m are weights and biases of the mth layer

2) Backpropagation .The key term in the given step is an error function, which represents the error between the real output ~*y* and predicted output ~*y* . When the feed-forward process will reach the output layer, the backpropagation algorithm will propagate errors from the last (output) layer back to the input layer by applying the *chain rule*.Thus it is necessary to compute the error function to the previous layer. Let's denote $\delta^{(}l)$ as the error of the node j in layer l, then the error functions $\delta^{L}, \delta^{L-1}, \delta^{L-2}, .., \delta^{2}$ are calculated in the following way:

$$\delta^{L} = a^{L} - yt \quad (2)$$

$$\delta^{l} = ((\theta^{(l)})^{T}\delta^{(l+1)} * g^{0}(z^{(l)})$$

As can be seen from Formula 2, the error of the last layer is calculated by obtaining the difference between the predicted vector of outputs of layer L and actual output y. The errors from other layers are calculated by multiplying the errors from the next year by weights (theta values) of the current layer. The obtained result is multiplied by the derivative of the activation function.

As a result, the partial derivatives of the cost function can be rewrited with the respect to the δ values:

$$\partial J(\theta)$$
$$\partial\theta^{l}_{ij} = (\delta^{(l+1)}$$
$$_{j})^{T}a^{(l)} (3)$$

3.5.2. Prediction Model Evaluation

For the evaluation of the proposed model, the project uses the coefficient of determination or R squared (R^2). The given coefficient usually is used for regression analysis and determines the accuracy of the predicted model. The R squared ranges between 0 and 1, where 1 is the highest fit, and 0 is the lowest.

The main goal in the project is to achieve a R^2 between 0.5 and 1, which determines that the output value can be predicted without high rate of error. The coefficient of determination is calculated by using following formula:

$$R^{2} = 1 - SE^{\wedge}y$$

$$SE^{-}{}_{y} (4)$$

where, $SE^{\wedge}{}_{y} = \overset{P}{} (y - {}^{\wedge}y)^{2}$

$SE^{-}{}_{y} = \overset{P}{} (y - {}^{-}y)^{2}$

The ^{-}y in the equation represents the mean of all y values from the dataset, and $^{\wedge}y$ represents the predicted value of y.

4. Technical Implementation

4.1. IoT Simulation

The IoT simulation process tried to meet the QoS requirements specified in section 3.1. The project uses Node-red open source platform for simulating the IoT environment.

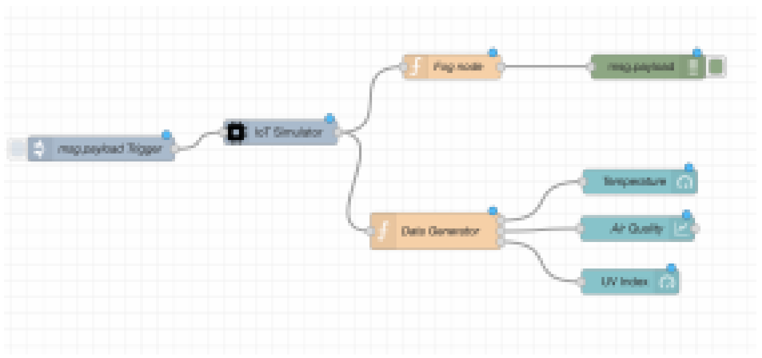


Figure 7: IoT simulation with Node-Red

As can be seen from Fig. 7, the simulation environment consists of multiple layers:

1. The IoT simulator - is the JavaScript file that generates mock device data (e.g., Temperature, Air Quality, and UV Index) and sends it to the fog node.
2. As was described in section 3.1, the fog layer can contain multiple tiers of nodes, each of which is responsible for a specific task. The simulated environment consists of two tiers. The first tier contains a *proxy-node*, which is responsible for making decisions related to the serverless allocation. The second tier consists of fog and cloud nodes responsible for the execution of the serverless function.

According to the QoS metrics, the fog nodes can be simulated using virtual machines with predefined configurations.

In the proposed model, the data is obtained from the functions running both on the Fog and Cloud layers. Thus to simulate the IoT environment, each segment will have two Virtual Machines running with the configurations shown in the Tables 2 and 3.

	# vCPU cores	Virtual Memory	CPU Capabilities
VM1	1	2048 MB	2.1GHz
VM2	1	1096 MB	1.82GHz

Table 2: Virtual Machine parameters for the Local environment

	# vCPU cores	Virtual Memory	CPU Capabilities
VM1	2	7500 MB	2.2GHz
VM2	1	3500 MB	1.82GHz

Table 3: Virtual Machine parameters for the Cloud environment

It is necessary to mention that the VMs computational capacity shown in Fig.2 and 3 are defined based on the QoS metrics and from the project written by Mahmud and Buyya 2019, which describes the ways to simulate the Fog environment.

4.2. OpenFaaS Configuration

As described in the previous section, the experiment includes four virtual machines which are executing serverless functions specified in described 3.2. The proposed model was decided to use OpenFaaS open-source platform to build and deploy serverless functions.

The implementation of OpenFaaS architecture has to follow requirements specified in official *OpenFaaS* documentation.

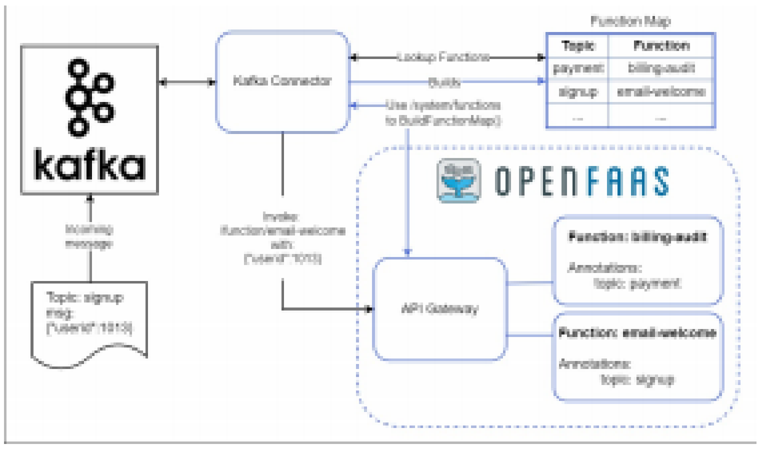


Figure 8: Kafka architecture (Source https://www.openfaas.com/blog/kafka-connector/)

Fig. 8 represents the implementation of the serverless architecture, which is taken from the official OpenFaaS documentation. At the top of the architecture is the Kafka message broker, which can be used to communicate between the fog nodes. The integration between the OpenFaaS platform and Kafka broker is achieved by using *Kafka Connector*. The Kafka connector then uses the OpenFaaS Gateway for functions invocation.

Fig. 9 represents the stack of technologies that will be used for the OpenFaaS implementation. At the bottom of the stack is the Docker platform. Docker is an open-source platform that enables developers to package and run applications inside the containers (Docker n.d.). The given platform creates an isolated environment inside the host machine and enables us to deploy as many containers as necessary.

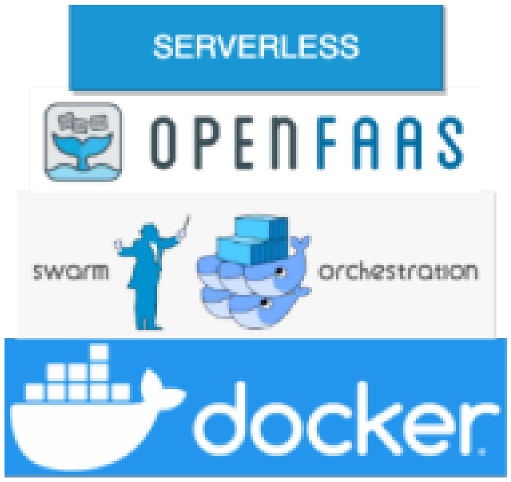


Figure 9: Technology Stack used in the experiment

The Docker Swarm lies at the top of the Docker and is a container orchestration tool that manages mul tiple containers configured together in a cluster. The user can either create a new Swarm or join already existing Swarm. Docker Swarm is usually created by one or multiple Docker Engines, which are called nodes.

In order to start working with custom serverless functions, it is necessary to run OpenFaaS main com ponents at the top of Docker Swarm. These components can be deployed locally by running OpenFaaS script inside the project folder which is cloned from the official GitHub repository (https://github.com/openfaas/faas.git)

4.3. Backpropagation Implementation

For the implementation of the algorithm described in section 3.5.1, it is necessary to identify steps that will help to create a foundation of source code.

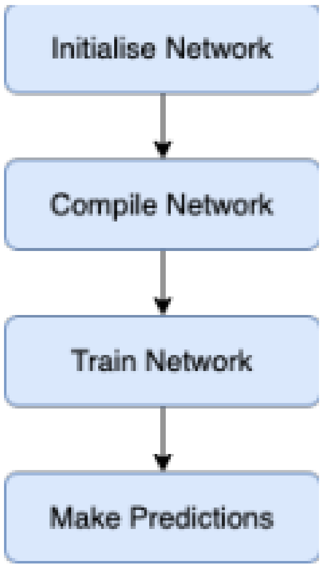


Figure 10: Steps for the model training

As it can be viewed from Fig. 10, the implementation has been divided into four main steps, which will be described profoundly later. For the accomplishment of the steps specified below, the project will use the open-source library called *Keras*. Keras provides with python deep learning API for working with machine learning algorithms more flexibly .

1. Initialize Network. According to the official documentation, the Keras consists of a sequence of layers wrapped into the container called Sequential class (https://keras.io/about/). Thus, after creating the Sequential class, the rest layers can be attached accordingly. For instance, for the project, the following code example has been used to initialize the network:

```
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='linear'))
model.add(Dense(1))
```

As stated in Sheela and Deepa 2013, currently, there is no general rule or theorem to determine the adequate number of hidden layers of neurons for a neural network. Thus, the most efficient way to specify the number of hidden layers and neurons is through tests.

As a result, after several test results, the optimal number of hidden layers for the project has been chosen two with 12 and 8 hidden neurons, respectively.

2. Compile Network. The compilation phase converts created layers into the executable format. The compilation phase needs to be done straight after the initialization of the network and before the training phase. The API for the compilation of the network accepts two parameters: the optimization algorithm for using in the training phase and the cost function to evaluate the network.

```
model.compile(loss='mean_squared_error', optimizer='me')
```

The optimization algorithm may differ depending on the requirements of the network. In the given ex ample, the MSE (Mean Squared Error) is used as a default loss function for the model.

3. Train Network. Once the network has been initialized and compiled, it can start to adapt the weights and biases according to the dataset specified in the project. The network receives the matrix X as an input that stores all machine metrics specified in section 3.3 and an array of outputs Y which stores a response time of each request.

Table 4: Partial X input for function-sensor-data

#	CPU	Docker Memory	Input Size	Network Sent Data	Processes Running	Virtual Memory Used
1	20.5	0.74	0.521	113042737	2	68.4
2	87.0	0.22	0.701	115669445	4	69.8
3	100.0	0.24	0.761	116392406	9	69.1
4	56.5	0.25	0.781	116712616	1	68.8
5	100.0	8.352	1.521	153636163	7	69.4
6	15.7	1.07	1.861	163320754	9	70.3

Table 5: Y output for function-sensor-data

#	Response Time
1	2.248
2	4.684
3	5.221
4	3.678
5	6.276
6	6.769

Table 5: Y output for function-sensor-data

Tables 4 and 5 illustrate the example of data that is used for training the model.

```
loss, accuracy = model.evaluate(X, y)
```

The given method returns the loss and accuracy of the network in order to evaluate the performance of the network.

5. Results and Analysis

The given section represents the experiment results, which are evaluated by using the coefficient of de termination R^2 defined in section 3.5.2. By using Keras API, it is probable to calculate training loss and validation loss of the model. The experiment is considered successful if the loss value decreases with each epoch during the model training.

Node Name	Function Name	R squared
Local Fog Node	Function-Sensor-Data Function-Image Function-Second-Image	0.85 0.78 0.69
Cloud Fog Node	Function-Sensor-Data Function-Image Function-Second-Image	0.82 0.71 0.66

As described in section 3.5.2, the R squared is essential for the evaluation of the proposed model, as it indicates how accurate our predicted model is. Fig. 5 represent the values of R squared for the final models. Generally, the experiment was divided into two parts: cloud node data analysis and local node data analysis. Each part has gone through two main steps:

1. (a) Process: To start the model training, by running 150 epochs.
   (b) Pre-Condition: To have valid data for training.
   (c) Goal: To decrease the training and validation loss.
   (d) Purpose: To make the model more accurate with each iteration. In other words, to minimize the difference between the predicted value y (response time) and real value y when making predictions.
2. (a) Process: Fit the trained model with new values.
   (b) Pre-Condition: To have valid data for testing.
   (c) Goal: To get the value of R squared greater than 0.5.
   (d) Purpose: To validate the trained model, if it is suitable for predicting the future.

5.1. Results from Fog environment

By following the steps specified above, firstly, it is necessary to run iterations and to decrease the training loss with each iteration.

As a result, The Fig. 11 validates that the step achieved the expected outcomes for the *Sensor-data function* (the same output as perceived by other functions). In other words, the loss function decreases exponentially with the increase of the number of epochs.

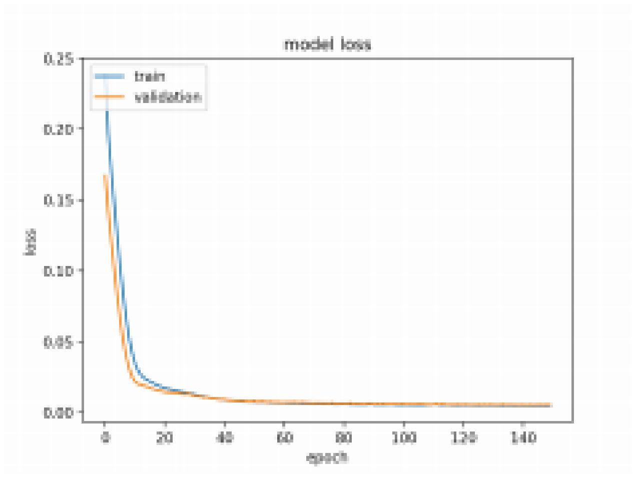


Figure 11: Model Loss for Function-Sensor-Data

The second step is to fit the model with new data and to evaluate the neural network model.

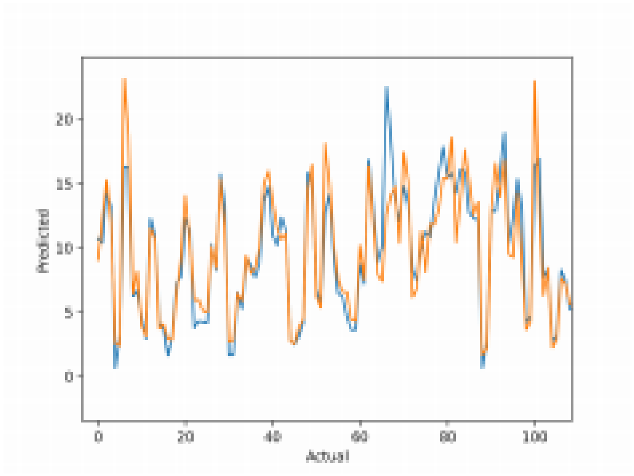


Figure 12: The difference between expected and predicted values for Function-Sensor-Data in Fog

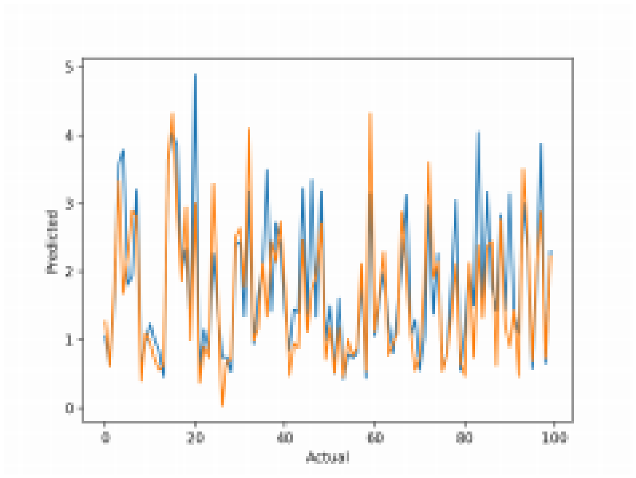


Figure 13: The difference between expected and predicted values for Function-Image in Fog

The Fig. 12 and 13 demonstrate the plot between predicted and expected values. As can be seen from the plots, the predicted response time is close to the actual. Thus, it confirms that the models can be used in our framework for predicted the response time in the future.

The achievement of the goal specified in *step 2* is also evidenced by the graph Fig. 5, which demonstrates that the R squared is higher than 0.5.

5.2. Results From Cloud environment

The Cloud layer applies the same steps that have been achieved in the Fog layer. Thus, Fig. 15 demon strates the results from the first step, which is the exponential decrease of the training loss.

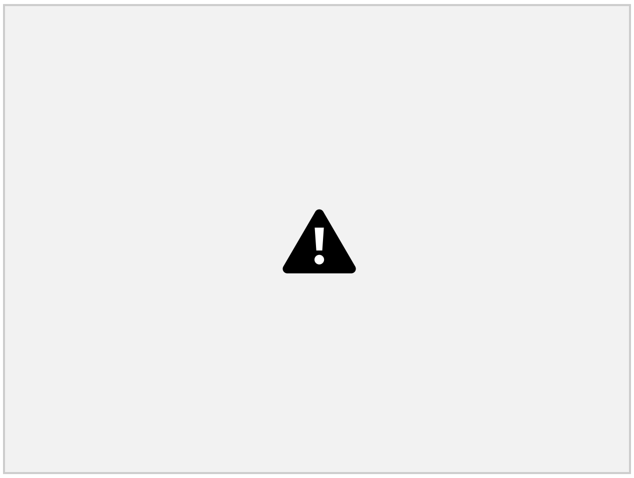


Figure 14: Model Loss for Function-Sensor-Data in Cloud

For the second step, the new test data is used for testing the trained model. The Fig. 15 and 16 illustrate the level of proximity of predicted and actual response time.

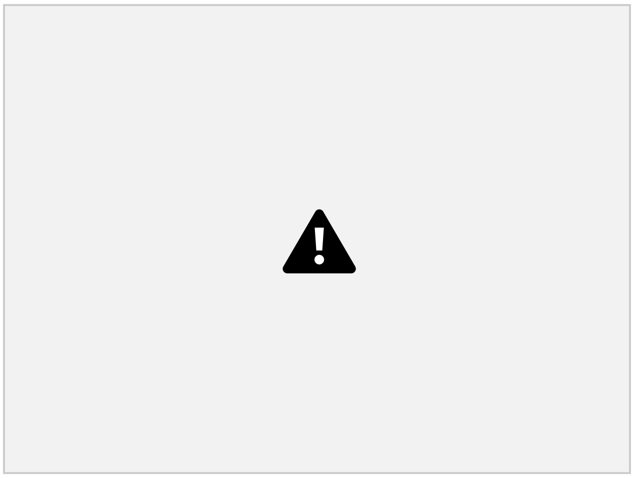


Figure 15: The difference between expected and predicted values for Function-Sensor-Data in Cloud 16

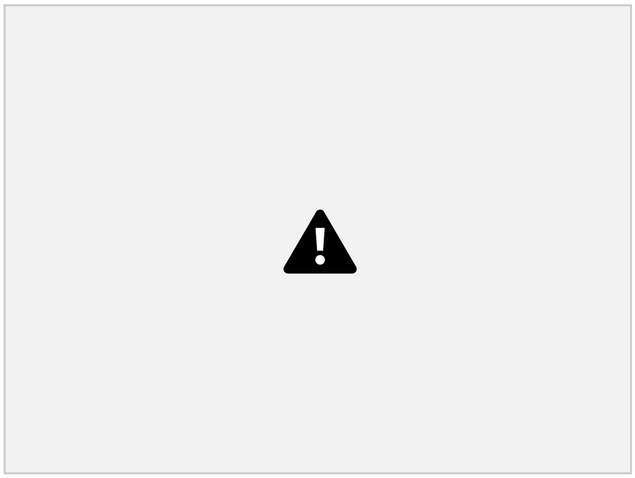


Figure 16: The difference between expected and predicted values for Function-Image in Cloud

5.3. Final Framework

After finishing the steps specified in the given section, we will have a model that can be used for making predictions. It is necessary to mention that the model will periodically be retrained based on the newly received data.

The final framework is represented in the form of API through which the *proxy-node* can determine where to execute to serverless function. The API receives a JSON object that will have the following key and values:

```
{
  ”functionName”: ”value”,
  ”cloud”: ”metric 1, metric 2, ... , metric n”
  ”local”: ”metric 1, metric 2, ... , metric n”
}
```

As can be seen from the JSON object, the body of the request has to contain the function name that needs to be executed, and the current metrics separated by the comma for each environment. As a result, the response contains the name of the *layer* where it is better to execute the function and predicted response time for the given layer. The response also contains predicted response time for another layer (which is ”cloud” in the example below).

```
The function should be executed in ”Fog Layer.”
Predicted response time: [value]
Predicted response time for ”Cloud Layer”: [value]
```

The Fig. 17 and 18 illustrate the example of the request which has been sent by using *Postman* (https: //www.postman.com/).

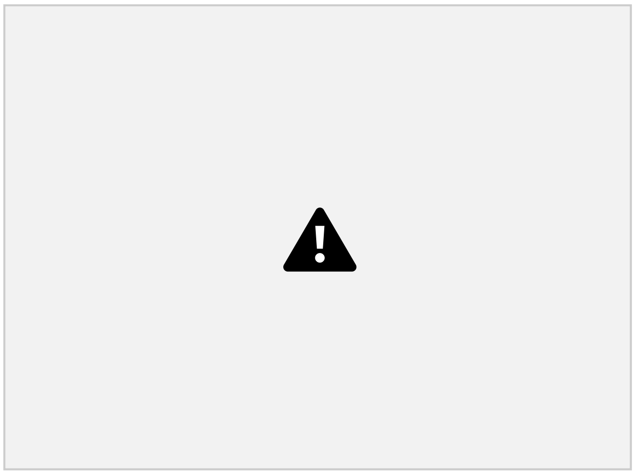


Figure 17: The response example #1

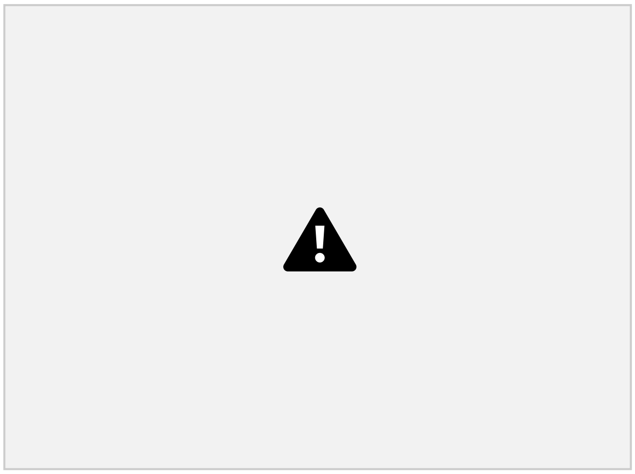


Figure 18: The response example #2

6. Conclusion

The article proposed a framework for predicting the response time of serverless functions in the IoT environment. In the beginning, the problem has been identified, and the key components of the solution have been implemented and discussed in the paper. In the proposed framework, the author firstly investigated which input features can be used for predictions and then created a dataset to feed the model. The ML model was then evaluated using the evaluation criteria, and the results were graphically shown in the article. The author then showed the real usage of the framework by sending the request to the specified API.

Currently, the author uses the ML regression model to predict the numerical value. As future work, there is a possibility to use a classification model to predict more values in a real environment. For instance, to predict the response time at a specific time of the day.

6.1. Critical Evaluation

The author tried to follow the IoT architecture to simulate the environment and generate the dataset. However, the real situation can be more complicated than described in the article. Thus, the proposed framework needs to be tested in the real world to make a full evaluation of the trained model. The critical evaluation can be divided into two parts:

1. *Evaluation of IoT simulation*. It is clear that the real-world IoT platform has more complex architecture than in the simulated. The conducted experiment has generated random values for the network delay, which occurs in the IoT platforms. However, the simulated environment didn't contain the essential component that exists in the real environment: IoT Gateway. The IoT gateway is a central device that collects all data from the devices and sends it to the upper layer. Thus, the simulated environment can be improved by simulating the load of the gateways.

   Moreover, despite the flexibility and convenience of using the Node-Red platform, it is less precise compared to other simulating platforms as iFogSim or YAFS. The given platforms have not been considered only because of their complexity, as it would take much time to learn them before starting the experiment.

   Another possible improvement can be scalability. The simulated environment only contains the min imum number of IoT devices and the fog nodes because of the lack of resources of the experiment. However, the real world example sometimes may contain hundreds of devices that communicate with each other via specified protocols.

   As stated by Law 2019, "a simulation model should always be developed for a particular set of objectives. In fact, a model that is valid for one objective may not be for another". In other words, the author wanted to concentrate specifically on the fog and cloud nodes for building a prediction model and skipped some components in the architecture that may be important in the real world.

2. *Evaluation of ML model*. Frequently, the ML models tend to overfit. Overfitting is the incorrect optimization problem in which the model can show false positives results. In such cases, the model shows high accuracy with test data but can fail with new data from the real-world. To avoid overfitting is necessary to have sufficient data, both training, and testing phase. Because the project's model has been trained by using a small dataset, there is a possibility of model overfitting in the real world. However, the given problem can be solved by obtaining a bigger dataset in the future.

   Further, the dataset has shown strong relationships between the chosen input features and the response time, but in a real IoT environment, the input features may change or be extended. For instance, the experiment didn't investigate the relationship between the device power consumption during the execution of specific function.

7. Acknowledgements

The author is grateful to the supervisor Bohus Ziskal for guiding and giving valuable advice throughout the project; the head of the school Veronika Douchova; the program leader Radek Honzik and all lecturers who gave precious knowledge during the education.

The author also wants to express gratitude to her family, who made it possible to take a master's degree and for their support throughout the studies.

8. Appendices

8.1. Appendix A: Journal Requirements

The following list represents the list of requirements specified by the journal (Elsevier n.d.):

- All sections in the article have to be numbered, and the numbers have to be used as references in the article.
- The introduction has to contain a short background description of the topic and objectives defined by the author.
- The abstract is required, and the references inside it should be avoided.
- If there are more than one Appendices, they should be described as A, B,...
- The title page has to contain the following information: Title, Author names, and affiliations, Present address.
- After the abstract, the author has to specify the main *keywords* (max. 6).
- The article has to contain the Acknowledgements section at the end of the article.
- There are no restrictions on the formatting of the reference.
- There is no specified word limit during the submission.

References

Abouaomar, Amine et al. (2019). “A Resources Representation for Resource Allocation in Fog Computing Networks”. In: *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, pp. 1–6.

Adhikari, Mainak, Tarachand Amgoth, and Satish Narayana Srirama (2019). “A Survey on Scheduling Strate gies for Workflows in Cloud Environment and Emerging Trends”. In: *ACM Computing Surveys (CSUR)* 52.4, pp. 1–36.

Ai, Yuan, Mugen Peng, and Kecheng Zhang (2018). “Edge computing technologies for Internet of Things: a primer”. In: *Digital Communications and Networks* 4.2, pp. 77–86.

Alli, Adam A and Muhammad Mahbub Alam (2020). “The fog cloud of things: A survey on concepts, architecture, standards, tools, and and applications”. In: *Internet of Things*.

Choi, Yeongho and Yujin Lim (2016). “Optimization approach for resource allocation on cloud computing for iot”. In: *International Journal of Distributed Sensor Networks* 12.3, p. 3479247.

Cisco (2015). “Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are”. In:

Deng, Ruilong et al. (2016). “Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption”. In: *IEEE internet of things journal* 3.6, pp. 1171–1181.

Docker (n.d.). *Docker overview — Docker Documentation*. https : / / docs . docker . com / get - started / overview/. (Accessed on 08/03/2020).

Elsevier (n.d.). *Guide for authors - Internet of Things - ISSN 2542-6605*. https://www.elsevier.com/journals/internet-of-things/2542-6605/guide-for-authors. (Accessed on 08/24/2020).

Hub, Arduino Project (Nov. 2017). *Salutis project - Arduino Project Hub*. https://create.arduino.cc/projecthub/suai-ihpcnt/salutis-project-bf49b4. (Accessed on 08/21/2020).

IDC (n.d.). *IDC: The premier global market intelligence firm.* https : / / www . idc . com/. (Accessed on 08/19/2020).

Jahantigh, Motahareh Nazari et al. (2019). “Integration of Internet of Things and cloud computing: a sys tematic survey”. In: *IET Communications*.

Koniagina, Mariia et al. (2020). “Development Trends of an Internet of Things in Context to Information Security Policy of a Person, Business and The State”. In: *Journal of Talent Development and Excellence* 12.2s, pp. 1181–1193.

Law, Averill M (2019). “How to build valid and credible simulation models”. In: *2019 Winter Simulation Conference (WSC)*. IEEE, pp. 1402–1414.

Lee, Gilsoo, Walid Saad, and Mehdi Bennis (2017). “An online secretary framework for fog network formation with minimal latency”. In: *2017 IEEE International Conference on Communications (ICC)*. IEEE, pp. 1– 6.

Li, Jianhua et al. (2017). “Latency estimation for fog-based internet of things”. In: *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*. IEEE, pp. 1–6.

Li, Liangzhi, Kaoru Ota, and Mianxiong Dong (2018). “Deep learning for smart industry: Efficient manu facture inspection system with fog computing”. In: *IEEE Transactions on Industrial Informatics* 14.10, pp. 4665–4673.

Mahmud, Redowan and Rajkumar Buyya (2019). “Modelling and simulation of fog and edge computing environments using iFogSim toolkit”. In: *Fog and edge computing: Principles and paradigms*, pp. 1–35.

Margariti, Spiridoula V, Vassilios V Dimakopoulos, and Georgios Tsoumanis (2020). “Modeling and Simula tion Tools for Fog Computing—A Comprehensive Survey from a Cost Perspective”. In: *Future Internet* 12.5, p. 89.

Ooi, Ching Sheng, Meng Hee Lim, and Mohd Salman Leong (2019). “Self-Tune Linear Adaptive-Genetic Algorithm for Feature Selection”. In: *IEEE Access* 7, pp. 138211–138232.

Puliafito, Carlo et al. (2020). “MobFogSim: Simulation of mobility and migration for fog computing”. In: *Simulation Modelling Practice and Theory* 101, p. 102062.

Rupani, Ajay et al. (2017). “A robust technique for image processing based on interfacing of Raspberry-Pi and FPGA using IoT”. In: *2017 International Conference on Computer, Communications and Electronics (Comptelix)*. IEEE, pp. 350–353.

Sarkar, Suvajit et al. (2019). “Serverless Management of Sensing Systems for Fog Computing Framework”. In: *IEEE Sensors Journal*.

Shaikhina, Torgyn and Natalia A Khovanova (2017). “Handling limited datasets with neural networks in medical applications: A small-data approach”. In: *Artificial intelligence in medicine* 75, pp. 51–63.

Sheela, K Gnana and Subramaniam N Deepa (2013). “Review on methods to fix number of hidden neurons in neural networks”. In: *Mathematical Problems in Engineering* 2013.

Singh, Manisha and Gaurav Baranwal (2018). “Quality of service (qos) in internet of things”. In: *2018 3rd International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*. IEEE, pp. 1–6.

Solutions, Cisco Fog Computing (2015). “Unleash the power of the Internet of Things”. In: *Cisco Systems Inc*.

Wardana, Aulia Arif and Riza Satria Perdana (2018). “Access control on internet of things based on pub lish/subscribe using authentication server and secure protocol”. In: *2018 10th International Conference on Information Technology and Electrical Engineering (ICITEE)*. IEEE, pp. 118–123.