SoftwareX 26 (2024) 101714

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

CloudEdgeAssetOptimizer: Tool to optimize the Cloud-Edge computing network resources at given requirements of processing delay, battery capacity and cost

Paulius Tervydis [a,b,*], Linas Svilainis [a], Žilvinas Nakutis [a], Alberto Rodríguez-Martínez [c]

[a] Kaunas University of Technology, Studentu 50, LT-51368 Kaunas, Lithuania
[b] Kaunas University of Applied Engineering Sciences, Tvirtoves al. 35, LT-50155 Kaunas, Lithuania
[c] Communications Engineering Dept., UMH, Avda. Universidad S/N, 03202 Elche, Spain

ARTICLE INFO

Keywords:
Edge
Cloud computing
Load balancing
Optimization

ABSTRACT

CloudEdgeAssetsOptimizer is a software tool designed to evaluate and optimize the assets within a Cloud and Edge computing network. Its primary purpose is to provide insights and estimations to ensure efficient resource allocation and decision-making. The developed software simulates the queues in Cloud and Edge devices, providing waiting times, battery consumption of Edge devices, and Servers' load. With aforementioned parameters available, valuable insights for decision-making can be obtained to optimize the network. Alternatively, automated optimization can be performed using the embedded functions. CloudEdgeAssetsOptimizer aims to optimize the operational efficiency, enhance resource utilization, and ultimately improve the overall system performance. CloudEdgeAssetsOptimizer functions utilize the queueing theory principles, therefore it also enables users to explore and fine-tune the system parameters across diverse domains, including telecommunication networks, transportation and manufacturing systems.

Current code version	v.0.1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-23-00475
Permanent link to Reproducible Capsule	
Legal Code License	MIT-0
Code versioning system used	git
Software code languages, tools, and services used	python 3.9.15
Compilation requirements, operating environments & dependencies	numpy 1.24.1, pandas 1.5.2, scipy 1.10.0, matplotlib 3.6.2, PIL 9.4.0, tkinter 8.6
If available Link to developer documentation/manual	
Support email for questions	paulius.tervydis@ktu.lt

1. Motivation and significance

Cloud computing is an attractive alternative when large amounts of data have to be collected and processed: there is no need for the complicated hardware at the front-end devices [1,2]. However, large data streams are generated in addition to the bottleneck load of the Cloud server. Furthermore, usually just a processing results are of the interest. Edge computing was proposed to solve this problem: data is prepossessed in close vicinity of the front-end device [3]. In such case, processing can be done both on Edge and Cloud devices. The load balancing of such network requires optimization [4]. Large research effort has been concentrated on communication issues: execution delay, packet error rate, handovers and latency [4–6]. Software packages that account all the intermediate network loads, network setup delays, errors and latency or even node energy consumption are available [7–9]. The application that requires large processing resources is considered here [10]. Only atomic tasks are considered [11]. In such case delays introduced in communication network are negligible compared to the processing time. It is assumed that Edge computing is run on the autonomous devices, i.e. Edge computing consumes the battery

* Corresponding author at: Kaunas University of Technology, Studentu 50, LT-51368 Kaunas, Lithuania.
E-mail address: paulius.tervydis@ktu.lt (Paulius Tervydis).

https://doi.org/10.1016/j.softx.2024.101714
Received 23 July 2023; Received in revised form 1 January 2024; Accepted 22 March 2024
Available online 1 April 2024

Table 1
Simulation software comparison.

Simulator	Implementation	Type	Modeling	Entities
iFogSim	Java, based on CloudSim	Event driven	Network (link bandwidth, delay, network usage), Energy, Cost	Sensors, Actuators, Fog devices and Data centers
FogTorch	Java	NA	Network latency, bandwidth	Things, Fog and Cloud
EdgeCloudSim	Java, based on CloudSim	Event driven	Network (WAN and WLAN link models), Mobility	Mobile client, Edge server, Cloud
EdgeFog Cloud	Python	NA	Cost	Edge, Fog, Data store
EmuFog	Java, MaxiNet	Emulator	Network latency, Scalability, Cost	Fog nodes, Network devices (routers)
FogBus	Java	NA	Network latency and usage, Scalability, Cost	IoT and Fog devices, Data centers
MyiFogSim	Java, based on iFogSim/CloudSim	Event driven	Network (link bandwidth, delay, network usage), Energy, Cost	Mobile devices (sensors, actuators), Data centers
YAFS	Python	Event driven	Network bandwidth, Mobility, Scalability	Sensors, actuators, Fog and Cloud
FogNetSim++	C++, based on CloudNetSim++/ OMNET++	Network driven	Network (packet drop, retransmission, bandwidth, BER), Mobility, Scalability, Energy, Cost	Mobile devices, Fog and Broker nodes, Base stations
CloudEdgeAsset Optimizer	Python	Analytical, numerical simulation	Load balancing, Energy, Cost	Edge and Cloud devices

resources [12]. Cloud computing, in turn, is related to the server cost and these resources also have a limit. Queue waiting time is increased if processing requests arrive to Cloud server when it is close to its limit. Then, routing at least part of the processing from Cloud to a Edge device can reduce the waiting time in queue. On the other hand, battery can be completely depleted if processing all the data requests on Edge device. Software package described here is intended to balance the aforementioned Edge and Cloud resources based on queuing theory [13–15].

2. Edge-Fog-Cloud simulation software comparison

The simulation software allows researchers and practitioners to model and analyze complex Edge–Fog–Cloud systems, providing valuable insights into their behavior and performance. This aids in the design, development, and validation of novel approaches for Fog environments, offering a controlled environment for testing various scenarios. Additionally, simulations help in the selection of appropriate resources and architectures, reducing the time and cost associated with the physical implementations. There are many simulation software options available for the Fog (Edge and Cloud) computing due to the multifaceted nature of this technology and its wide range of applications [16]. Detailed studies, surveys and comparisons of the Fog computing simulation tools are presented in [16–18]. Table 1 is provided for the purpose of comparing the simulation tool proposed in this publication to the state-of-the-art Edge–Fog–Cloud computing simulators.

Majority of state-of-the-art Edge–Fog–Cloud computing simulators are of two types: (i) event driven, and (ii) network driven. An event driven simulators focus on responding to the events that trigger the state changes in a system, while a network-driven simulators are specialized for modeling the behavior of computer networks, with a primary focus on the data flow and interactions between nodes. These two approaches are chosen based on the specific characteristics and requirements of the system being simulated. Such computing simulations generate a list of events or tasks and simulate data packet or data flow transmission over a well-defined network.

The remaining paragraphs summarize the main differences of our software compared to the state-of-the-art counterparts.

The detailed comparison of Cloud-Edge simulation software in [19] highlights that simulation time may extend to tens of seconds, and this duration escalates with an increase in the simulated number of Edge and Cloud devices. The prolonged simulation time poses challenges, particularly when seeking optimal solutions. In response to this concern, our software may be potentialy usefull tool for the community, offering swift estimations facilitated by integrated analytical models. By prioritizing efficiency, our software aims to address the time-related obstacles encountered in existing simulation tools, providing the community with a faster and more responsive solution for estimating optimal configurations in Cloud-Edge scenarios.

The main driver to develop this software was our research project [10], where Cloud data processing is used to solve the inverse solution and the waiting time to get the results is significant. The main problem was to find the optimal numbers of Edge devices and Cloud servers which are required to meet the desired waiting time requirements for a given rate of data processing requests. Therefore, our tool is not evaluating the network delay as it was three to five orders smaller than the data processing duration. It was another reason, why the analytical or numerical simulation model type was selected for our software instead of event or network driven types. Event based simulation model was also developed to verify if the analytical model guarantees the correct results. The comparison results are given in the first illustrative example below. The analytical model performs numerical calculations and provides the results much faster than an event driven model.

Another problem is that the majority of analyzed simulation tools require experienced users to perform the modeling, because GUI is missing, input and output data is provided in a form of CSV or XML files. Therefore, the included GUI in our software proves useful for users to set parameters and conduct simulations.

3. Software description

The CloudEdgeAssetsOptimizer is written in Python and can be easily modified by users for their needs. The program code can be downloaded from the GitHub.

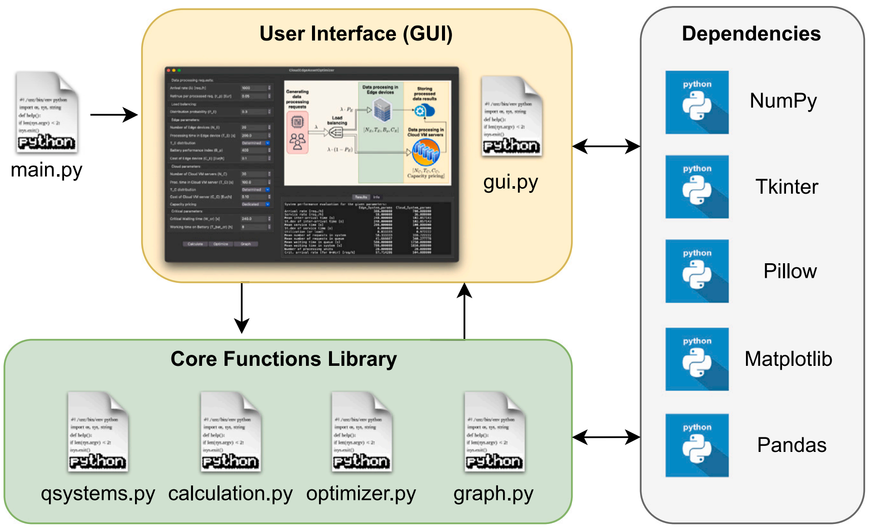


Fig. 1. Software architecture.

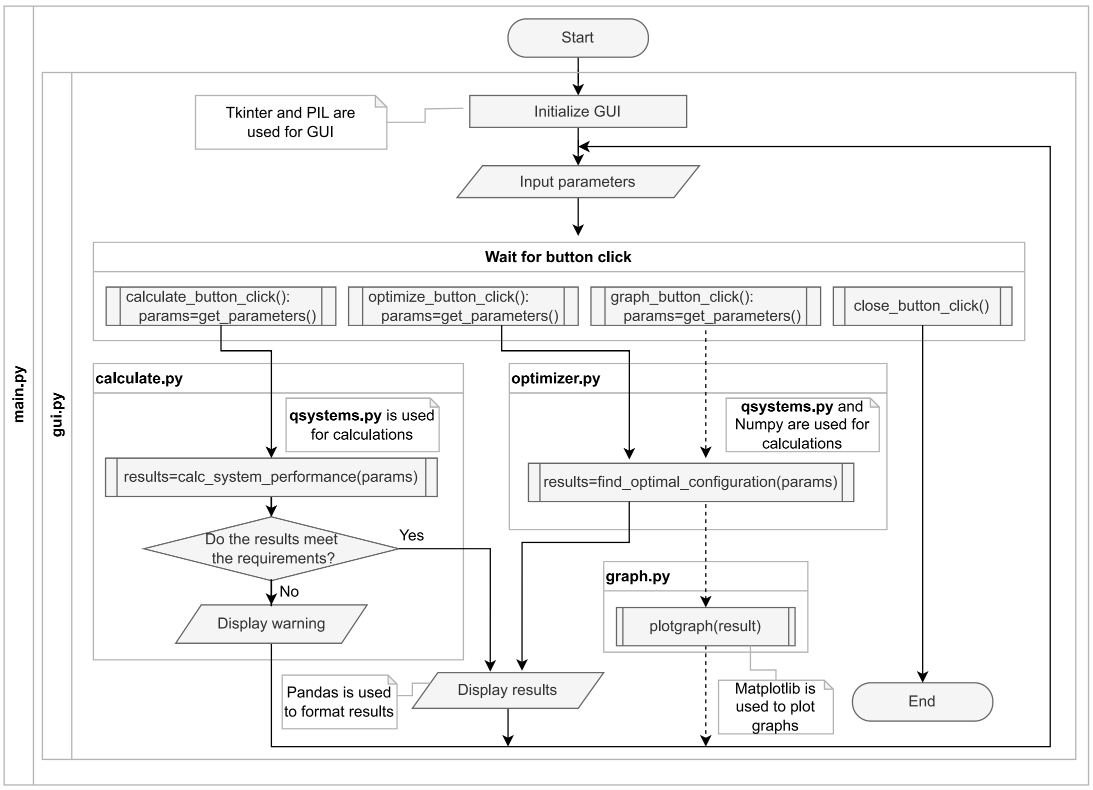

Start
Initialize GUI
Tkinter and PIL are used for GUI
Input parameters
Wait for button click
calculate_button_click(): params=get_parameters()
optimize_button_click(): params=get_parameters()
graph_button_click(): params=get_parameters()
close_button_click()
main.py
gui.py
calculate.py
qsystems.py is used for calculations
results=calc_system_performance(params)
Do the results meet the requirements?
Yes
No
Display warning
optimizer.py
qsystems.py and Numpy are used for calculations
results=find_optimal_configuration(params)
graph.py
plotgraph(result)
Pandas is used to format results
Display results
Matplotlib is used to plot graphs
End

Fig. 2. Flow chart of the CloudEdgeAssetsOptimizer.

3.1. Software architecture

The high-level visualization of the software architecture is given in Fig. 1.

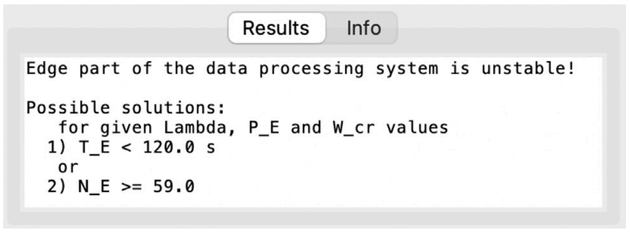
The flow chart presented in Fig. 2 shows the data flow algorithm and the relationship of different system's components used by the developed GUI program.

The application can be divided into two main components: the core functions library and the user interface.

3.1.1. Core functions library

This is the core part of the CloudEdgeAssetsOptimizer, containing the core functions that can be used to model single server and multi-server queuing systems. It is designed to be reusable and independent of the user interface. These functions also can be used to model and to evaluate not only various Edge Cloud data processing systems. A user can use these functions based on their functionalities. For example:

- `qsystems.py`: contains functions that utilize queuing theory to simulate various single server and multi-server queuing systems. The `ssqs` function can be used to estimate parameters of single server queuing systems. Function `msqs` can be used to estimate parameters of various multi-server queuing systems. For



```
Edge part of the data processing system is unstable!

Possible solutions:
   for given Lambda, P_E and W_cr values
  1) T_E < 120.0 s
   or
  2) N_E >= 59.0
```

Fig. 3. Example of a warning message.

the sake of simplicity, it is considered that all the servers in such system are the same, and the arriving data processing requests are equally distributed. Such assumptions make it easier to estimate how the performance parameters depend on the number of servers. Function `msqsa` can be used if multi-server system has servers with different parameters and even if load between them is not equally distributed. Therefore, this function is the most versatile function in the library. Function `msqs_ar_cr` can be used to estimate what arrival rate is critical to the given waiting time. And function `msqs_sn_cr` can be used to estimate what number of servers is needed to ensure specified waiting time when arrival rate is given.
These functions are documented. Examples are given to guide users on how to use the functions effectively in their own code.

- `calculation.py`: is used to estimate the simulated Cloud-Edge system parameters. It is also used to verify and to give recommendations, if system parameters are not valid. For example, a warning message with possible problem solutions is displayed if user entered number of Edge devices that makes the system unstable for the provided λ, P_E and W_{cr} values (Fig. 3). Such guidance is useful for a user if needed to estimate some parameters manually.
- `optimizer.py`: is used to find optimal number of Edge devices and Cloud VM servers that ensure that waiting time is bellow critical, considering the rate of data processing requests, battery time, load balancing and cost.
  The optimal numbers of Edge devices N_E and Cloud servers N_C are found using the optimization model which is implemented in the module by the following criteria: mean waiting time of data processing in Edge devices and Cloud servers must be $< W_{cr}$, mean working battery time of Edge device must be $> T_{E\ bat\ cr}$, and when maximum profit is achieved, considering device costs (C_E, C_C), revenue for data processing (r_p) and Cloud pricing strategy. To guarantee the solution the calculations are performed for all the possible N_E and N_C combinations.
  The optimization is performed in the following steps:
  1. Arrays of Edge device numbers $N_E \in [N_{E\ bat\ cr}, \dots, 2N_{Emax}]$ and Cloud server numbers $N_C \in [0, \dots, 2N_{Cmax}]$ are generated. Maximum numbers N_{Emax} and N_{Cmax} are estimated using `msqs_sn_cr()` function, it calculates the critical number of devices to ensure that waiting time is bellow critical. The calculated N_{Emax} and N_{Cmax} values are multiplied by 2 to guarantee that all the possible $\{N_E, N_C\}$ combinations around $N_{E\ opt}$ and $N_{C\ opt}$ are estimated. The minimum number of Edge devices is $N_{E\ bat\ cr} = \lceil \lambda_E T_{bat\ cr} / B_p \rceil$ to ensure that the working time on battery for an Edge device is $> T_{bat\ cr}$, here B_p – battery performance index – number of continuous data processing cycles to discharge full battery.
  2. Profit, revenue and cost that could be obtained by data processing in such system is calculated using the `model` function in the module for each number pair of data processing devices from N_E and N_C arrays. The model estimates what are the delay critical intensities of data processing requests using `msqs_ar_cr` function of `qsystems.py` module. If the number of devices is not sufficient to process or the intensity is too big, then only a fraction of requests will be processed.
  3. Optimal N_E and N_C numbers that ensure all the criteria are estimated according to the maximum profit from step 2.

  Another optimization model that uses scipy.optimization toolbox was developed to verify if the proposed optimization model provides correct results. The comparison revealed that the proposed code runs almost 10 times faster and provide the same results. The test code is given in this paper's GitHub repository.
- `graph.py`: has functions to display optimization results in graphical form. Such representation provide better insights about how different parameters are related and their impact to the overall system performance.

3.1.2. *User interface*

The graphical user interface (GUI) provides a convenient way for users to interact with the core functions library without writing code directly.

- `main.py`: this script is used to start the GUI of the main program. The program is started by running the command in the terminal: `python main.py`.
- `gui.py`: it contains the code for the main graphical user interface (Fig. 4). The graphical user interface is made using standard Tkinter module, therefore it should work with a typical Python installation without additional GUI packages.

3.1.3. *Dependencies*

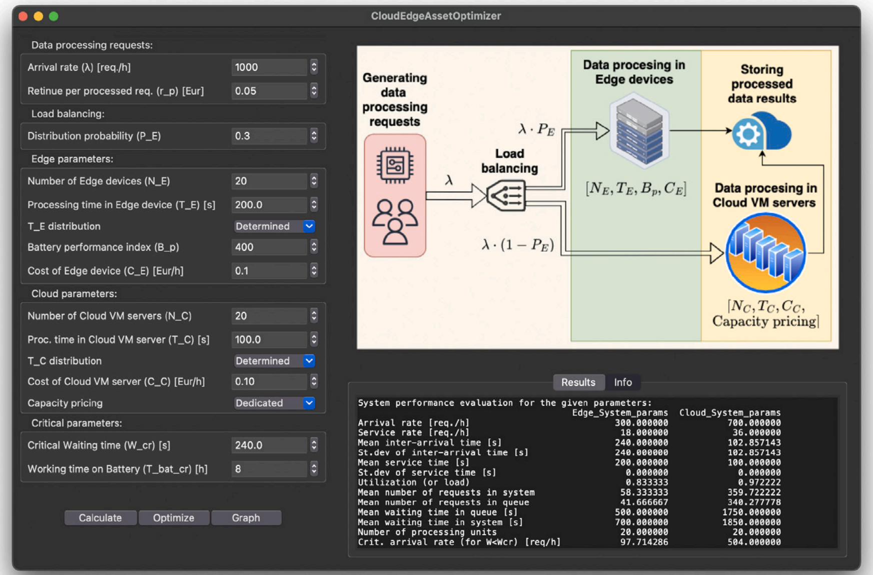
The application relies on these external dependencies: `numpy` and `pandas` for data processing, `pillow` to show the data processing network structure in the GUI, `matplotlib` to show graphical dependencies of system parameters and `tk` (Tkinter) for the GUI.

3.2. *Software functionalities*

The core functions of this library can estimate such parameters of single and multi-server systems: mean waiting time in queue, mean waiting time (is a sum of processing time and waiting in queue), mean number of entities in the system and queue, server utilization. These parameters can be estimated as a function of: arrival rate, service rate, mean service time and variation of service time. Supplementary functions can be used and some of them are given in examples and implemented in the gui script, that can evaluate system cost, battery time and other parameters. By utilizing such functionality it is able to perform multi-criteria evaluation of Cloud-Edge data processing networks (Fig. 5).

For example, it is possible to estimate what are the optimal numbers of Edge devices N_E and Cloud servers N_C required for data processing. This estimation is based on factors such as arrival rate λ of data processing requests, load balancing or distribution between Edge an Cloud (estimated by P_E and P_C probabilities), processing capabilities (mean time of data processing in Edge devices T_E and Cloud servers T_C), and user demand for the waiting time W_E and W_C. By analyzing these parameters, the software can determine the optimum number of devices needed to handle the workload efficiently, without causing the excessive waiting times or delays.

On the other hand, CloudEdgeAssetsOptimizer can estimate the critical arrival rate λ_{cr} of requests or tasks within the network. By considering factors such as data processing performance requirements (W_E, W_C) and data processing network capacity (N_E, N_C, T_E, T_C)


CloudEdgeAssetOptimizer
Generating data processing requests
Load balancing
λ
$\lambda \cdot P_E$
$\lambda \cdot (1 - P_E)$
Data procesing in Edge devices
$[N_E, T_E, B_p, C_E]$
Storing processed data results
Data procesing in Cloud VM servers
$[N_C, T_C, C_C,$ Capacity pricing$]$
Results
Info
Calculate
Optimize
Graph

Data processing requests:
Arrival rate (λ) [req./h] 1000
Retinue per processed req. (r_p) [Eur] 0.05
Load balancing:
Distribution probability (P_E) 0.3
Edge parameters:
Number of Edge devices (N_E) 20
Processing time in Edge device (T_E) [s] 200.0
T_E distribution Determined
Battery performance index (B_p) 400
Cost of Edge device (C_E) [Eur/h] 0.1
Cloud parameters:
Number of Cloud VM servers (N_C) 20
Proc. time in Cloud VM server (T_C) [s] 100.0
T_C distribution Determined
Cost of Cloud VM server (C_C) [Eur/h] 0.10
Capacity pricing Dedicated
Critical parameters:
Critical Waiting time (W_cr) [s] 240.0
Working time on Battery (T_bat_cr) [h] 8

```
System performance evaluation for the given parameters:
                                       Edge_System_params  Cloud_System_params
Arrival rate [req./h]                          300.000000           700.000000
Service rate [req./h]                           18.000000            36.000000
Mean inter-arrival time [s]                    240.000000           102.857143
St.dev of inter-arrival time [s]               240.000000           102.857143
Mean service time [s]                          200.000000           100.000000
St.dev of service time [s]                       0.000000             0.000000
Utilization (or load)                            0.833333             0.972222
Mean number of requests in system               58.333333           359.722222
Mean number of requests in queue                41.666667           340.277778
Mean waiting time in queue [s]                 500.000000          1750.000000
Mean waiting time in system [s]                700.000000          1850.000000
Number of processing units                      20.000000            20.000000
Crit. arrival rate (for W<Wcr) [req/h]          97.714286           504.000000
```

Fig. 4. The graphical user interface.

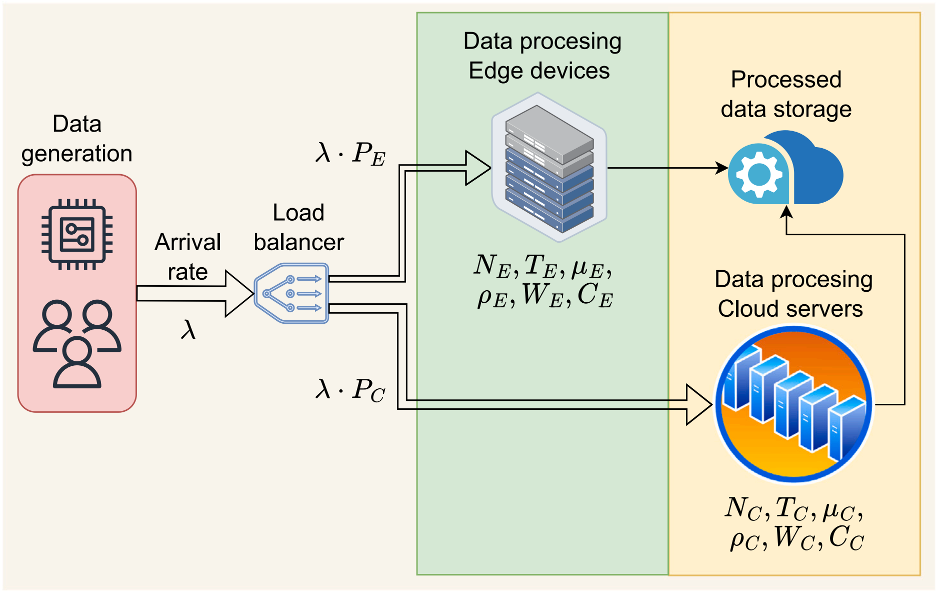


Fig. 5. Cloud and Edge data processing system.

and utilization of processing devices (ρ_E, ρ_C), the software can estimate what is the critical threshold of request arrival rate λ_{cr}. This information allows the network administrators to ensure that the mean waiting time for data processing remains below a critical threshold, thus optimizing user experience and satisfaction.

Another important capability of the CloudEdgeAssetsOptimizer is its ability to assess the performance of battery-powered Edge devices. By considering the power consumption characteristics of these devices, the software can estimate utilization of such device ρ_E and if it will operate for a specific duration without requiring a battery recharge. This estimation is important for planning the deployment of Edge devices and ensuring uninterrupted operation within the network.

Furthermore, CloudEdgeAssetsOptimizer takes into account economical criteria when determining the optimal number of assets within the network. It considers factors such as the cost of devices (C_E, C_C), retinue and profit of service provider. By analyzing these factors, the software can recommend the optimal number of assets that not only meet the technical requirements but also provide the most cost-effective solution. The CloudEdgeAssetsOptimizer goes beyond traditional analysis by considering different pricing strategies. Users can evaluate the cost of the system under fixed pricing (reserved resources) or pricing that depends on utilization ρ_C. This capability provides valuable insights for making the informed decisions about pricing strategies and resource allocation.

4. Illustrative examples

To illustrate how the proposed package can be applied in the multi-criteria problems, two practical examples are presented. Examples are aimed to show how different functions and methods can be used to

```python
from qsystems import msqs
import pandas as pd

N_C = 10; T_C_s = 100; W_cr_s = 300
stdT_C_s = 50 # for MG1 system
# for MD1 system stdT_C = 0 (handled in auto by msqs)
# for MM1 system stdT_C = T_C (handled in auto by msqs)
lambda_C_list = range(0,350,1)
sysparam_mm1 = []; sysparam_md1 = []; sysparam_mg1 = []
for lambda_C in lambda_C_list:
    sysparam_md1.append(
        msqs(ar=lambda_C,sn=N_C,s1=T_C_s/3600,qs="md1"))
    sysparam_mm1.append(
        msqs(ar=lambda_C,sn=N_C,s1=T_C_s/3600,qs="mm1"))
    sysparam_mg1.append(
        msqs(ar=lambda_C,sn=N_C,s1=T_C_s/3600,
        vs=(stdT_C_s/3600)**2,qs="mg1"))

df_mm1 = pd.DataFrame(sysparam_mm1)
df_md1 = pd.DataFrame(sysparam_md1)
df_mg1 = pd.DataFrame(sysparam_mg1)
lambda_cr_md1=df_md1[df_md1['w']*3600<W_cr_s].tail(1)['ar']
lambda_cr_mg1=df_mg1[df_mg1['w']*3600<W_cr_s].tail(1)['ar']
lambda_cr_mm1=df_mm1[df_mm1['w']*3600<W_cr_s].tail(1)['ar']
print("When mean(T_C) = %.2f s, N_C = %d :"%(T_C_s,N_C))
print(" - lambda_cr = %d req./h, if std(T_C) = 0 s"%lambda_cr_md1)
print(" - lambda_cr = %d req./h, if std(T_C) = 50 s"%lambda_cr_mg1)
print(" - lambda_cr = %d req./h, if std(T_C) = 100 s"%lambda_cr_mm1)

# >>> When mean(T_C) = 100.00 s, N_C = 10 :
# >>>  - lambda_cr = 288 req./h, if std(T_C) = 0 s
# >>>  - lambda_cr = 274 req./h, if std(T_C) = 50 s
# >>>  - lambda_cr = 240 req./h, if std(T_C) = 100 s
```

Fig. 6. Python code to solve the task of Example 1.

asses and to optimize usage of data processing resources. The first example demonstrates usage of core functions, that are the basis of the CloudEdgeAssetsOptimizer. This may be useful if a user wants to write their own code. The second example illustrates how to use the CloudEdgeAssetsOptimizer in the graphical user interface. The third example demonstrates how to estimate Cloud-Edge networks with different topologies and node interconnections.

4.1. Example 1: Finding maximum arrival rate for given system and performance requirements

Consider a data processing system with $N_C = 10$ Cloud servers. Mean data processing time $T_C = 100$ s. What is maximum arrival rate of data processing requests per hour λ_{cr} if critical mean waiting time $W_{cr} = 300$ s? Estimate how the λ_{cr} depends on the distribution of processing time, when: (a) processing time is constant — standard deviation $std(T_C) = 0$ s, (b) standard deviation of processing time $std(T_C) = 50$ s, (c) processing time is exponentially distributed — standard deviation $std(T_C) = mean(T_C) = 100$ s.

To solve this task the multi-server `msqs` model can be used from the core functions library. The Python code listing with answers is given in Fig. 6.

How the mean waiting time depends on arrival rate for different distributions of T_C is given in Fig. 7.

The results in Fig. 7. show that the mean waiting time is a function of arrival rate. It also depends on the distribution of processing (service) time. In all three cases the mean data processing time is the same, but the higher variance of service times the greater mean waiting time. Such arrival rate and waiting time dependency is very important if it is required to ensure quality of service. For the given example if it is necessary to ensure that the mean waiting time $W < W_{cr} = 300$ s, the critical arrival rate should be in the range $\lambda < \lambda_{cr} = [240..288]$ req./h. This arrival rate can be managed by load balancing, by limiting number of users or by increasing number of servers. Such multi-criteria optimization principle is also implemented in the CloudEdgeAssetsOptimizer.

The x markers in the Fig. 7 show the results obtained by the event driven model developed with SimEvents toolbox using Matlab Simulink. The model code is also given in this papers GitHub repository. It was developed to prove that the analytical model provides the same results as the state-of-the-art simulation software.

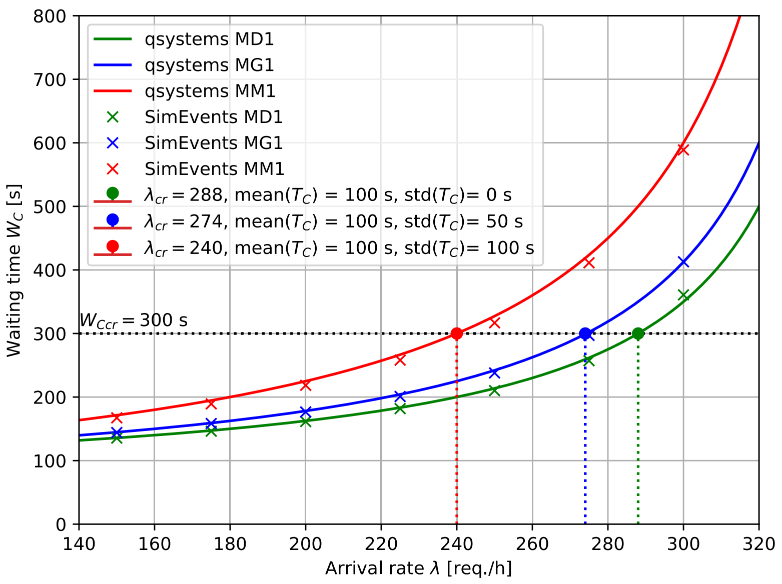


Fig. 7. Waiting time as a function of arrival rate.

4.2. Example 2: Optimization of assets in Cloud-Edge computing network

Consider that data processing system has Edge devices and Cloud VM servers. Arrival rate of requests $\lambda = 1000$ req./h. Mean processing time in an Edge device is $T_E = 200$ s. Mean processing time in a Cloud server is $T_C = 100$ s. It is required that the mean waiting time for data processing result must be less than $W_{cr} = 240$ s. Cost for Edge device $C_E = 0.5$ Eur/h. Cost for server $C_C = 0.15$ Eur/h, if "dedicated capacity" pricing option is used. Revenue per processed request $r_p = 0.1$ Eur. Battery of Edge devices should last for at least $T_{bat\ cr} = 8$ h. It is determined that fully charged battery is discharged per $B_p = 400$ continuous processing cycles. Energy consumption is related to Edge devices utilization. It is required to find the optimal numbers of Edge N_{Eopt} devices of and of Cloud servers N_{Copt} for maximum profit, when $P_E = 0.2$.

This example will also illustrate the usage of the CloudEdgeAssetOptimizer's GUI. The parameters are entered in the parameter entries on the left side of GUI window as shown in Fig. 8.

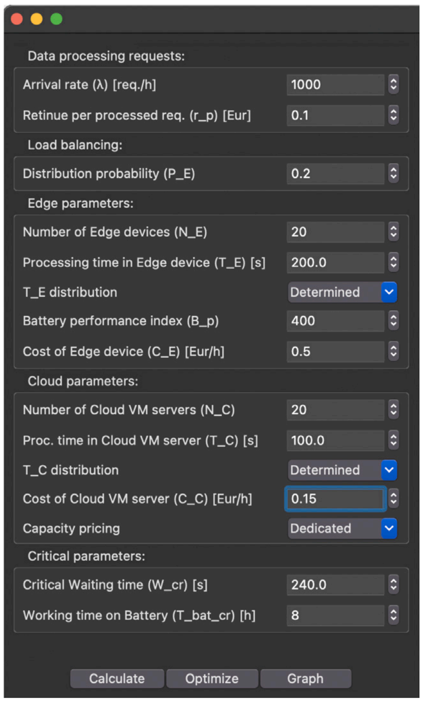
Having the parameters it is possible to calculate system performance describing parameters by pressing "Calculate" button. The results are presented in the "Results" tab on the right side of GUI window. Optimization is performed by pressing "Optimize" button. The optimization results and the list of parameters that the software is evaluating are presented in Fig. 9.

The optimization is made according multiple criteria: waiting time, battery time, optimum number of data processing devices and maximum profit. For the given example, one of the main requirements was that the mean waiting time should be less than 240 s. The optimizer managed to find combination that meets this and other requirements.

Graphical representation of system cost and profit estimation can be reviewed by pressing "Graph" button. Fig. 10. shows the 3-D surface and contour plots that can be used to investigate how profit depends on the number N_E of Edge devices and on the number N_C of Cloud VM servers.

The program also show the slices that are marked by dashed lines along the optimum in the contour plot. For example, how the cost, revenue and profit depend on the N_C is shown in Fig. 11.

The software can estimate how such system performance parameters depend on any other parameter that is in GUI window.



Data processing requests:

Arrival rate (λ) [req./h]	1000
Retinue per processed req. (r_p) [Eur]	0.1

Load balancing:

Distribution probability (P_E)	0.2

Edge parameters:

Number of Edge devices (N_E)	20
Processing time in Edge device (T_E) [s]	200.0
T_E distribution	Determined
Battery performance index (B_p)	400
Cost of Edge device (C_E) [Eur/h]	0.5

Cloud parameters:

Number of Cloud VM servers (N_C)	20
Proc. time in Cloud VM server (T_C) [s]	100.0
T_C distribution	Determined
Cost of Cloud VM server (C_C) [Eur/h]	0.15
Capacity pricing	Dedicated

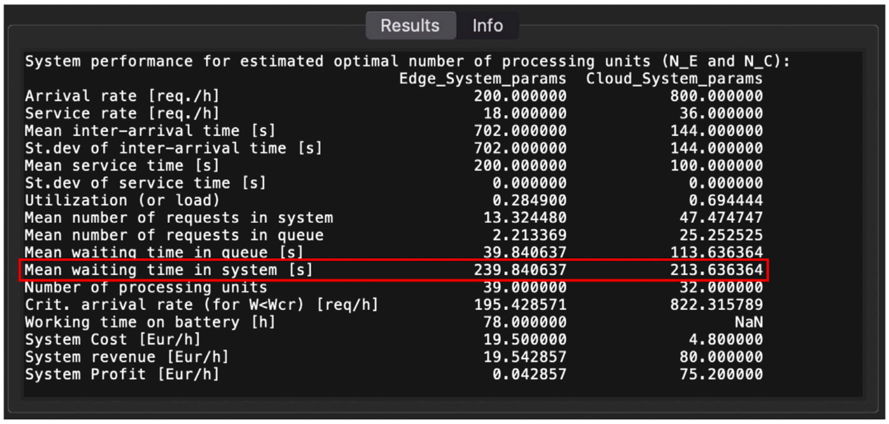
Critical parameters:

Critical Waiting time (W_cr) [s]	240.0
Working time on Battery (T_bat_cr) [h]	8

Fig. 8. Parameter entries in the CloudEdgeAssetOptimizer's GUI window.

4.3. Example 3: Simulation of Cloud-Edge network topology

To simulate a network, it is necessary to understand how network nodes are interconnected and how data flows are created and transmitted across the network. For example, in the provided network structure (Fig. 12), data flows are initiated by reading data from sensors. The data then undergoes processing, which can be executed by either Edge devices or Cloud servers. The simulation can be utilized to estimate how network performance depends on the distribution of data flows between Edge and Cloud processing devices. Additionally, it aids in determining the number of devices needed to meet waiting time requirements, estimating costs, and exploring various scenarios.



```
System performance for estimated optimal number of processing units (N_E and N_C):
                                         Edge_System_params  Cloud_System_params
Arrival rate [req./h]                            200.000000           800.000000
Service rate [req./h]                             18.000000            36.000000
Mean inter-arrival time [s]                      702.000000           144.000000
St.dev of inter-arrival time [s]                 702.000000           144.000000
Mean service time [s]                            200.000000           100.000000
St.dev of service time [s]                         0.000000             0.000000
Utilization (or load)                              0.284900             0.694444
Mean number of requests in system                 13.324480            47.474747
Mean number of requests in queue                   2.213369            25.252525
Mean waiting time in queue [s]                    39.840637           113.636364
Mean waiting time in system [s]                  239.840637           213.636364
Number of processing units                        39.000000            32.000000
Crit. arrival rate (for W<Wcr) [req/h]           195.428571           822.315789
Working time on battery [h]                       78.000000                  NaN
System Cost [Eur/h]                               19.500000             4.800000
System revenue [Eur/h]                            19.542857            80.000000
System Profit [Eur/h]                              0.042857            75.200000
```

Fig. 9. Optimization results.

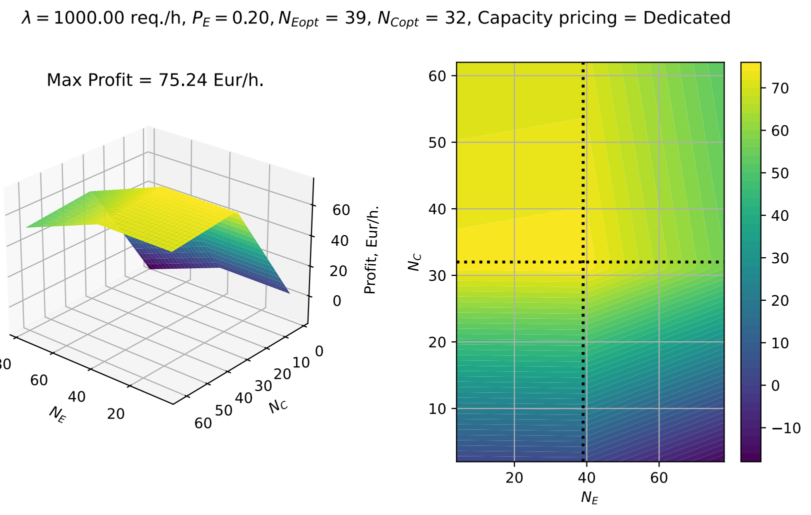


Fig. 10. Optimal numbers of N_{Eopt} and N_{Copt}, when $P_E = 0.2$.

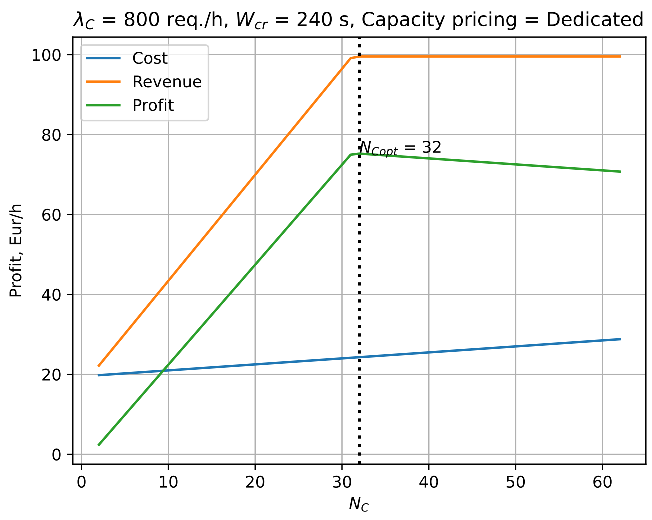


Fig. 11. Cost, revenue and profit vs N_C.

The separate `network.py` module was developed to define network nodes as distinct object classes and to leverage the same queuing systems models employed in other examples within the CloudEdgeAssetsOptimizer package.

All network node classes inherit from the general Node class, incorporating specific changes that pertain to individual network node features. Further details can be found in the comment lines within the `network.py` file. The subsequent code block (Fig. 13) includes an example demonstrating how to create network nodes and how to interconnect them.

To simulate the entire network topology illustrated in Fig. 12, it is essential to incorporate additional components such as Edge gateways, Edge devices, Cloud servers, sub-networks, and establish their interconnections. The complete code, along with detailed instructions on how to achieve this simulation and the corresponding results, is available on GitHub. When the code is executed, the results for each network node can be visualized. Each connection and network node can be assigned unique parameters. By fine-tuning network, equipment, and data flow parameters, users can identify the optimal configuration tailored to specific requirements. If a new parameter is required, the open-source code can be modified accordingly.

5. Impact

The developed software can be used to optimize the usage or distribution of Cloud-Edge data processing network assets. It may be useful tool to do multi-criteria evaluation or to ensure quality of service. Multiple parameters are evaluated to meet the requirements of waiting time, battery time, capacity pricing options. The core functions of the developed software are based on queuing theory, therefore can be used to evaluate systems where queuing occurs.

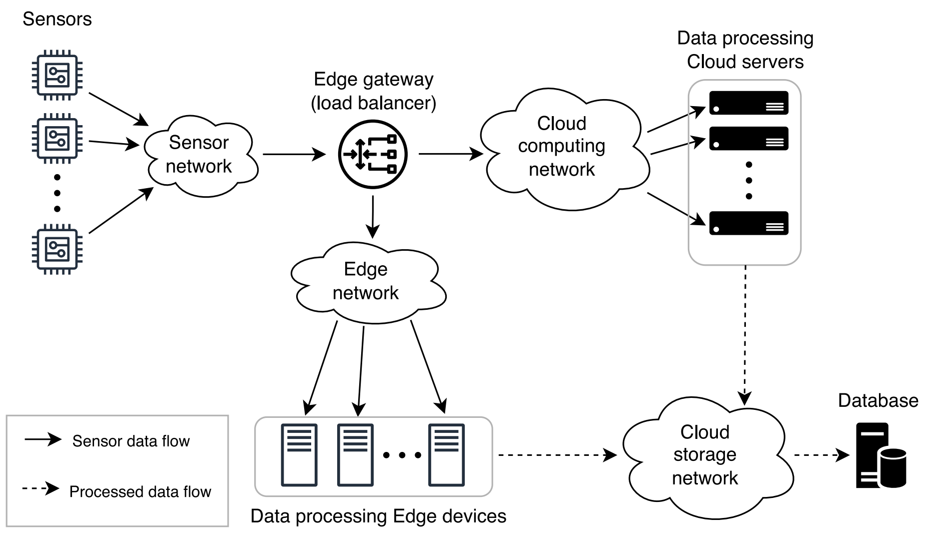


Fig. 12. Topology of simulated Cloud-Edge network.

```python
from network import *
# Number of Sensors is defined:
sensors_no = 9
# List of sensors is created:
sensors = []
for s_no in range(1,sensors_no+1):
    sensors.append(Sensor("Sensor %d"%s_no,↩
        ↪sensor_data_polling_rate))
# Sensor network is created
sensor_network = DataChannel("Sensor Network",↩
    ↪sensor_network_service_rate)
for sensor in sensors:
    sensor.connect_to(sensor_network)
# Full code with results is in the GitHub repository:
# example7_network.ipynb
```

Fig. 13. Python code fragment to simulate a network topology.

6. Conclusions

In this paper, the CloudEdgeAssetOptimizer software is proposed, an open-source Python toolkit that can be used to estimate and to optimize Cloud-Edge computing network resources. The toolkit implements reusable core functions and graphical user interface. The core functions can be used for other project where estimation on minimization of waiting times in queues is crucial. The developed GUI is user friendly, default parameters are already provided, if wrong parameters are entered it explains what and how to correct. More advanced users may adjust or add additional functionalities.

The developed software can be enhanced by incorporating additional Cloud pricing scenarios, enabling users to make more precise evaluations based on specific pricing models offered by particular Cloud service providers. This enhancement would provide a more accurate cost estimation for different deployment strategies.

Additionally, more advanced algorithms for load dynamics estimation could be implemented. This improvement could be utilized for dynamic resource allocation in response to changing data processing demands, ensuring optimal performance even under dynamic load conditions.

Furthermore, a more comprehensive energy consumption estimation model could be integrated to assess power usage across Edge devices and Cloud servers. This implementation would enable the evaluation of energy consumption under different conditions and resource allocation strategies.

Moreover, integrating more detailed network latency estimation capabilities, which consider network geographical distribution and communication delays, would enable users to estimate various networking and load distribution scenarios, thereby enhancing responsiveness and user experience.

CRediT authorship contribution statement

Paulius Tervydis: Methodology, Software, Writing – original draft, Visualization. **Linas Svilainis:** Conceptualization, Project administration, Supervision, Writing – review & editing. **Žilvinas Nakutis:** Conceptualization, Methodology, Writing – review & editing. **Alberto Rodríguez-Martínez:** Formal analysis, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: there are no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

All the data are provided in the GitHub repository.

Acknowledgments

This project has received funding from European Regional Development Fund (project No. 01.2.2-LMT-K-718-03-0026) under grant agreement with the Research Council of Lithuania (LMTLT).

References

[1] Ampatzidis Y, Partel V, Costa L. Agroview: Cloud-based application to process, analyze and visualize UAV-collected data for precision agriculture applications utilizing artificial intelligence. Comput Electron Agric 2020;174:105457. http://dx.doi.org/10.1016/j.compag.2020.105457.

[2] Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, et al. A view of cloud computing. Commun ACM 2010;53(4):50–8. http://dx.doi.org/10.1145/1721654.1721672.

[3] Shi WS, Cao J, Zhang Q, Li YHZ, Xu LY. Edge computing: Vision and challenges. IEEE Internet Things J 2016;3(5):637–46. http://dx.doi.org/10.1109/JIOT.2016.2579198.

[4] Mattia GP, Beraldi R. P2PFaaS: A framework for faas peer-to-peer scheduling and load balancing in Fog and Edge computing. SoftwareX 2023;21:101290. http://dx.doi.org/10.1016/j.softx.2022.101290.

[5] Malik AW, Qayyum T, Rahman AU, Khan MA, Khalid O, Khan SU. XFogSim: A distributed fog resource management framework for sustainable IoT services. IEEE Trans Sustain Comput 2021;6(4):691–702. http://dx.doi.org/10.1109/TSUSC.2020.3025021.

[6] Qayyum T, Malik AW, Khan Khattak MA, Khalid O, Khan SU. FogNetSim++: A toolkit for modeling and simulation of distributed fog environment. IEEE Access 2018;6. http://dx.doi.org/10.1109/ACCESS.2018.2877696, 63570-6383.

[7] Sonmez C, Ozgovde A, Ersoy C. EdgeCloudSim: An environment for performance evaluation of edge computing systems. Trans Emerg Telecommun Technol 2018;29(11):e3493. http://dx.doi.org/10.1002/ett.3493.

[8] Gupta H, Dastjerdi AV, Ghosh SK, Buyya R. IFogSim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. Softw Pract Exper 2017;47(9):1275–96. http://dx.doi.org/10.1002/spe.2509.

[9] Li ZH, Xie T, He GN. MEC-sim: An extensible simulator for mobile edge computing system. In: 2021 Chinese control conference. 2021, p. 6729–35. http://dx.doi.org/10.23919/CCC52363.2021.9549353.

[10] Svilainis L, Nakutis Z, Tervydis P, Chaziachmetovas A, Aleksandrovas A. Resource tradeoff analysis of plant physiological status sensor with cloud connectivity. In: 2021 11th IEEE international conference on intelligent data acquisition and advanced computing systems: technology and applications. 2021, p. 241–6. http://dx.doi.org/10.1109/IDAACS53288.2021.9660944.

[11] Sadatdiynov K, Cui LZ, Zhang L, Huang JZ, Salloum S, Mahmud MS. A review of optimization methods for computation offloading in edge computing networks. Digit Commun Netw 2022;9(2):450–61. http://dx.doi.org/10.1016/j.dcan.2022.03.003.

[12] Abner M, Wong PKY, Cheng JCP. Battery lifespan enhancement strategies for edge computing-enabled wireless bluetooth mesh sensor network for structural health monitoring. Autom Constr 2022;140:104355. http://dx.doi.org/10.1016/j.autcon.2022.104355.

[13] Urgaonkar R, Wang SQ, He T, Zafer M, Chan K, Leung KK. Dynamic service migration and workload scheduling in edge-clouds. Perform Eval 2015;91:205–28. http://dx.doi.org/10.1016/j.peva.2015.06.013.

[14] Afolalu SA, Ikumapayi OM, Abdulkareem A, Emetere ME, Adejumo O. A short review on queuing theory as a deterministic tool in sustainable telecommunication system. Mater Today Proc 2021;44(1):2884–8. http://dx.doi.org/10.1016/j.matpr.2021.01.092.

[15] Wu GW, Xu ZQ, Zhang H, Shen SG, Yu S. Multi-agent DRL for joint completion delay and energy consumption with queuing theory in MEC-based IIoT. J Parallel Distrib Comput 2023;176:80–94. http://dx.doi.org/10.1016/j.jpdc.2023.02.008.

[16] Gill M, Singh D. A comprehensive study of simulation frameworks and research directions in fog computing. Comput Sci Rev 2021;40:100391. http://dx.doi.org/10.1016/j.cosrev.2021.100391.

[17] Markus A, Kertesz A. A survey and taxonomy of simulation environments modelling fog computing. Simul Model Pract Theory 2020;101:102042. http://dx.doi.org/10.1016/j.simpat.2019.102042.

[18] Margariti SV, Dimakopoulos VV, Tsoumanis G. Modeling and simulation tools for fog computing—A comprehensive survey from a cost perspective. Future Internet. 2020;12(5):89. http://dx.doi.org/10.3390/fi12050089.

[19] Abreu DP, Velasquez K, Curado M, Monteiro E. A comparative analysis of simulators for the cloud to fog continuum. Simul Model Pract Theory 2020;101:102029. http://dx.doi.org/10.1016/j.simpat.2019.102029.