Chapter 3
Comparison of Task Scheduling Algorithms for Traffic Surveillance Application Using Fog Computing

Mluleki Sinqadu, Zelalem Sintayehu Shibeshi, and Khuram Khalid

3.1 Introduction

Recent advances in technologies, particularly digital technologies, have led to the presentation of another term known as the Fourth Industrial Revolution (4IR), which is promoted as the answer to everything from accumulations in education and improvement skills to organizations reducing expenses and serving their clients better. The Internet of Things (IoT) is one of the pillars of 4IR, characterized as the interaction among automation and data interchange. IoT is the prominent technology that creates the possibilities of communication between physical objects and the Internet [1]. Because of its advantages, many IoT-enabled systems and applications have been created. The applications are particularly important in domains such as smart homes, medical services, smart transportation, to give some examples. At the centre of the assessment and approval of these systems is the capacity to plan proper application models and the analysis of fog computing, which consists of fog devices, IoT devices and cloud servers. Taking into account that implementation of the fog environment can be very costly, researchers have decided to develop simulators that can be used to test real-life scenarios of IoT applications and cloud-based applications in an adaptable way.

M. Sinqadu (✉) · Z. S. Shibeshi
Department of Computer Science, University of Fort Hare, Alice, South Africa
e-mail: zshibeshi@ufh.ac.za

K. Khalid
Department of Computer Science, Ryerson University, Toronto, ON, Canada
e-mail: khuram.khalid@ryerson.ca

I. Woungang, S. K. Dhurandher, *4th International Conference on Wireless, Intelligent and Distributed Environment for Communication*, Lecture Notes on Data Engineering and Communications Technologies 94, https://doi.org/10.1007/978-3-030-89776-5_3

iFogSim [1] is an illustration of such a system, which has been used for designing models for fog/edge computing environment and running experiments. iFogSim has been developed by inheriting some basic CloudSim classes [2]. The design of Fog computing in iFogSim, alongside with intelligent surveillance model using a network of distributed cameras, is depicted in [1]. In [3], an API for fog applications was presented, and a Closed-Circuit TV (CCTV)-based vehicle tracking framework was demonstrated using the proposed API, taking note of that CCTV is a self-contained framework made out of cameras, recorders for surveillance. This chapter investigates how iFogSim is used to simulate and evaluate the performance of the proposed traffic surveillance application like the one presented in [4]. We did this by following the fog-based application model presented in [5] for a case scenario of a traffic surveillance application. In the work introduced in this framework, the model is improved with new components, adding new functionalities in the design of the following vehicle tracking process. However, in this chapter, we evaluate the same model using task scheduling algorithms on metrics that have been introduced in Sect. 3.4.

This chapter is organized as follows. In Sect. 3.2, some related works are presented. In Sect. 3.3, a brief introduction of scheduling algorithms that are used for evaluation is provided. In Sect. 3.3, we discuss the evaluation metrics that are used to test our model. In Sect. 3.5, the design of the proposed fog-based application model is described, along with justification for using iFogSim. In Sect. 3.6, the performance analysis of the Traffic Surveillance application model is assessed by simulations, using scheduling algorithms. Finally, Section 3.6 is the conclusion of this chapter.

3.2 Related Work

In real-time fog applications, before a task is executed, the fundamental question one should ask would be whether it should be cloud or fog [6]. For real-time applications that require emergence response, fog is always chosen over the cloud for processing as it can provide cloud resources closer to devices. The following related works review how fog computing can benefit IoT application and also reducing blocking while minimizing latency. In this section, we present works related to our project which demonstrate how different applications are evaluated using task scheduling algorithms.

Intharawijitr et al. [7] presented different policies that can be used to map tasks for fog devices while minimizing latency. The first policy selects which fog device to host the current task, whereas the second policy selects the fog device to reduce overall latency, and the last policy decides which fog device to increase resource usage. Results show that to avoid task blocking, the algorithm that prioritizes latency must be used for mapping tasks.

In [8], the authors proposed a strategy for task scheduling that checks the organization of clusters of fog devices and how the load can be balanced among the cellular network. The strategy allocates resources at the serving cell according to an ordered list of tasks and arranges clusters to meet the requirements of the tasks not yet served. Results indicate that the strategy works better in a cellular network when compared to static clustering.

A model that ensures minimum resource usage for fog resource usage of various IoT devices is presented in [9]. This model uses an approximation of resource usage. Moreover, it reflects the history of resource usage and starts with the resource allocation to frequent users.

On the other hand, Aazam and Huh [9] explored the issue of reasonable task offloading among fog devices in a 5G fog network. They demonstrated a task scheduler that takes into account task latency and energy usage. The fairness of using a fog system to reduce task latency is also demonstrated. The findings show that task scheduling among fog devices can reduce task execution time.

In [10], Deng et al. discussed the difference between power consumption and delay in a cloud-fog system. The authors proposed a model that solves the problem with resource allocation for power consumption and delays on a cloud-fog distribution network. To reduce the energy consumed by a fog layer must increase convex usage for a certain input workload of tasks, while a heuristic is used to improve the energy usage of a cloud. Moreover, the authors try to maximize the energy usage of communications. Results show that by giving away few resources, bandwidth can be saved and hence latency is minimized.

Pham and Huh [11] proposed a task scheduler to advance the efficiency of the method of offloading applications for a large scale. The objective is to achieve a decent compromise among span and financial expense. The scheduler sorts the tasks according to their span and duration of execution; then chooses the appropriate fog device to execute each task in the list.

In [12], Zeng et al. proposed a formulation to choose whether the processing of tasks ought to be completed on client devices or edge devices. The objective is to reduce the time taken to finish the tasks considering task scheduling. The time taken to finish each task incorporates computational time and data transmission time. Results demonstrate that the presented algorithms show incredible performance on processing data to edge devices for different task appearance rates and customer handling rates.

All the studies discussed above are related to our work, which is the implementation of the fog-based Traffic Surveillance Application model. In the following section, we will discuss the selected algorithms to evaluate our model based on four metrics and the design of our proposed model.

3.3 Selected Algorithms

The scheduling algorithms used to evaluate our proposed model are selected based on work done in [13]. Those algorithms are selected because they have been used successfully in cloud computing for evaluating different applications [14, 15]. In this work, those algorithms are applied to a distributed network of smart cameras. The first algorithm, First Come First Serve (FCFS), is a simple scheduling algorithm that executes the queued tasks and processes in the order in which they are received. It is the most straightforward CPU scheduling algorithm. The second algorithm is Short Job First (SJF), which selects the task with the shortest execution time as the next task to run. Preemptive or non-preemptive scheduling algorithms are available. It greatly decreases the total time spent waiting for other processes to complete. The third algorithm used is Round Robin (RR), in which each ready task runs in a cyclic queue for a predefined amount of time. This algorithm also allows for process execution without starvation. Finally, we consider the Generalized Priority (GP) scheduling algorithm, a popular option for real-time applications which ensures that the processor executes the task with the highest priority of all tasks that are currently ready to execute at any given time.

3.3.1 Motivation for Selected Algorithms

The scheduling algorithms discussed above have been evaluated in cloud computing before and proven to provide efficiency. Likewise, fog computing applications also require low latency while carrying out tasks efficiently. These algorithms are now used to evaluate a vehicular network application model that requires low latency and efficiency. To find a well-performing algorithm, different metrics must be used to evaluate all the algorithms and a comparison can be done. The selected algorithms have shown good scheduling of tasks on cloud computing applications; therefore, they are more suitable to evaluate the Traffic Surveillance Application. Our comparison of the studied algorithms reveals how efficient and fast each algorithm satisfies the application.

FCFS algorithm can be used in this model as it schedules the tasks based on arrival queue, which in this case are vehicle received detected from raw video streams which are discussed in Sect. 3.5. SJF is another algorithm that we consider suitable for vehicular networks as it processes the tasks with the shortest execution time as the next task to run. This algorithm is also fitting in this study as it helps in reducing accidents and process vehicles which are faster and likely to break the road rules. Another algorithm we are considering is RR, this algorithm runs each ready task in a cyclic queue; it is also suitable for a vehicular network to ensure that it gets the predefined time and also allow processing of the task without starvation. The last algorithm is GP, which is popular for real-time applications to ensure that tasks with high priority are executed at any given time. This algorithm is also suitable for a vehicular network to ensure the processing of tasks in real-time. All these algorithms are chosen with suitability with the vehicular network, but there is only one algorithm that can be chosen as most suitable, and it is selected based on the demonstration of good performance in all given metrics discussed in Sect. 3.6. Therefore, it will be the most suitable algorithm for a real-time IoT application.

In the following section, we will report on the metrics that will be used to evaluate our model considering the aforementioned scheduling algorithms.

3.4 Evaluation Metrics

In a fog distributed network, the performance evaluation is done based on latency, energy consumption, execution time and network usage. These evaluation metrics are briefly explained below:

Average Latency
Average Latency refers to the delay that happens when data is transmitted between surveillance cameras and the Cloud which can be calculated as shown in Eq. (3.1).

$$\text{Average Latency} = \text{CC–ET} \tag{3.1}$$

where CC is the CloudSim Clock and ET is the Emitting Time of a tuple. ET is calculated by the transmission time of a module to another module.

Energy Consumption
The energy consumption for the full topology of the network is calculated using Eq. (3.2).

$$\text{Energy} = \text{CEC} + (\text{NT} - \text{LUUT}) * \text{HP}. \tag{3.2}$$

The “energy consumption is calculated by taking the power of all components in a certain time frame of execution, where CEC is the current energy consumption, NT is the network time, LUUT is the last utilization update time, and HP is the host power in LU.”

Execution Time
Execution Time refers to the time taken by the tuple on consuming fog device resources. It can be mathematically calculated using Eq. (3.3):

$$\text{Execution Time} = \text{CT–SST} \tag{3.3}$$

where CT represents the current time and SST denotes the simulation start time.

Network Usage
“Network Usage refers to the amount of network bandwidth consumed while running simulation. It can be mathematically described using” Eq. (3.4):

$$\text{Network Usage} = (\text{TL} * \text{TC})\,/\text{MST} \tag{3.4}$$

where TL is the total latency and TS is the total size of the tuple, respectively; and MST is the maximum simulation time.

3.5 Design of a Proposed Traffic Surveillance Application Model for Fog Computing

In this section, we discuss the use case and design of our proposed application model.

3.5.1 Fog Computing Application Model Topology

In Fig. 3.1, a fog topology of the Traffic Surveillance Application model is presented which shows all the devices used.

Figure 3.1 illustrates how each process depends on one another in the fog topology tree from cloud to fog devices. The router in the topology represents the fog device while mobile-0 and mobile-1 represent the smart cameras physical infrastructure. Lastly, we have motor-0 and motor-1 referring to smart cameras and the pan–tilt–zoom (PTZ) actuation to vehicle detection.

3.5.2 Traffic Surveillance Application Model Design

Figure 3.2 shows our proposed fog-based application model with all five modules which have different tasks. These modules are interconnected by tuples which help in transmitting the data from one module to another.

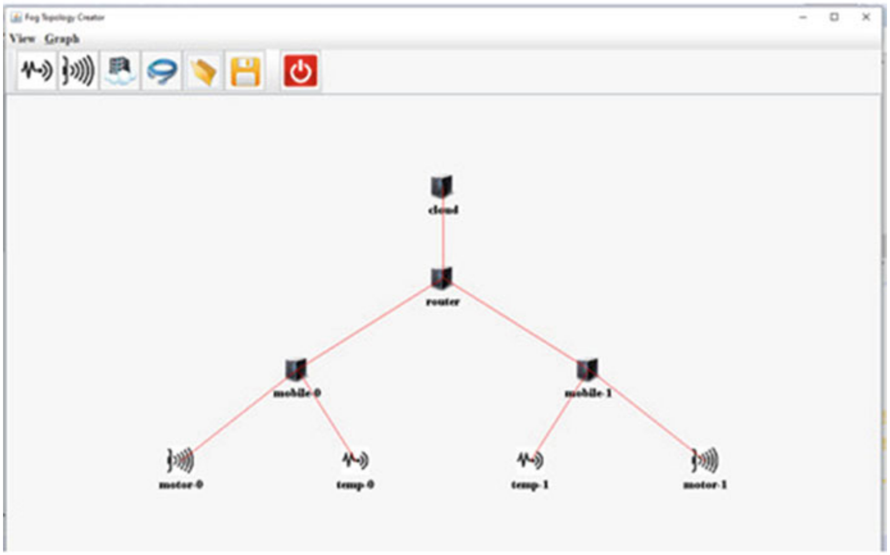

Fig. 3.1 Fog computing application model topology

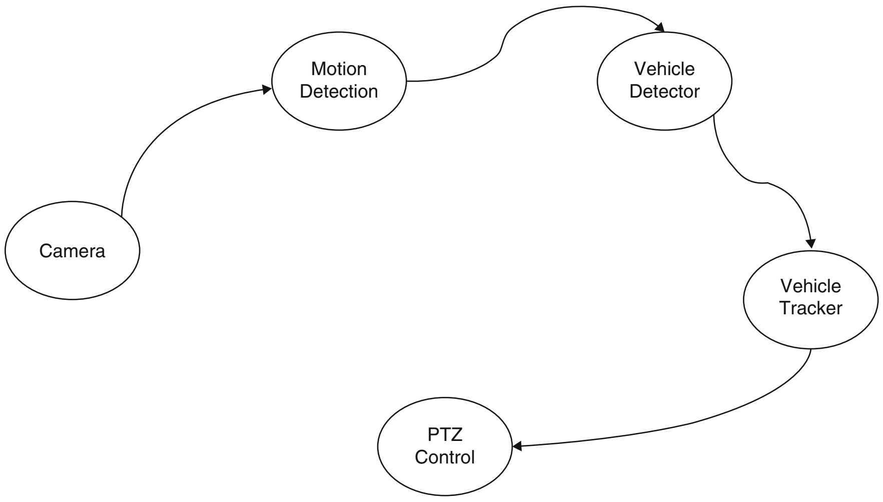

Fig. 3.2 Proposed fog-based application model

As mentioned earlier, the Traffic Surveillance application model relies on surveillance cameras that can detect moving vehicles. This application requires sufficient processing power for a good experience. Our surveillance application model consists of five modules as shown in Fig. 3.2. “These modules are motion detector, vehicle detector, vehicle tracker, user interface and pan–tilt–zoom (PTZ) controller. The motion detector module retrieves the raw video streams captured by a smart camera and sends the video to the vehicle detector module. The vehicle detector module detects the moving vehicles and sends vehicle identification and the current position of the vehicle to the vehicle tracker. The vehicle tracker obtains the coordinates of the tracked vehicles, computes an ideal PTZ configuration of all the cameras surveilling the area and then forwards the command to the PTZ control module. The PTZ module, placed in each smart camera, rotate the smart camera based on the PTZ parameters received from the vehicle tracker module. Lastly, the user interface module forwards a portion of the video streams containing each tracked vehicle to the user’s device. Since it is assumed that each smart camera is equipped with a PTZ control module, our application model of the Traffic Surveillance application is composed of four modules.”

3.5.3 Sequence Diagram of the Proposed Application Model

Figure 3.3 demonstrates a sequence diagram of our proposed model and how objects interact in the arranged time sequence.

The sequence diagram in Fig. 3.3 depicts the interaction of the main objects of the Traffic Surveillance Model and how operations are carried out. This interaction

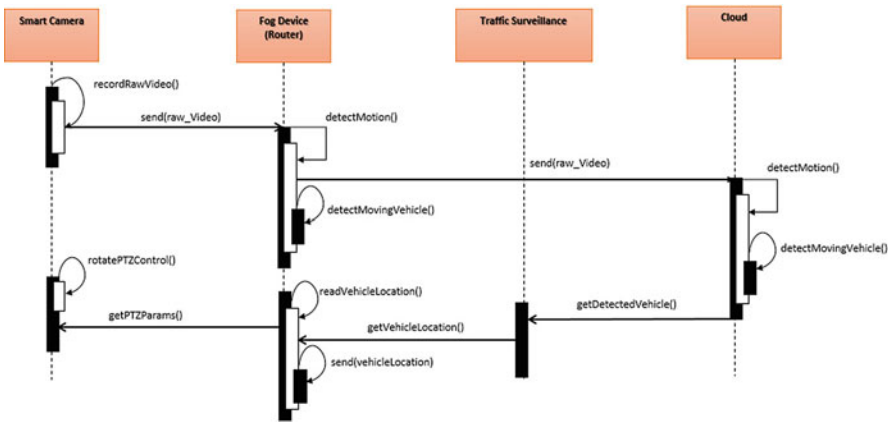

Fig. 3.3 UML sequence diagram of the proposed application model

can take place between objects, systems and subsystems. The figure also shows how messages are sent and processed using a vertical axis of the diagram. The Smart Camera records a raw video stream, and this is represented using *recordRawVideo()* method and then send the message using *send(rawVideo)* method which carries out this operation to the next object which fog device. Fog devices can be server, router, or controllers which are deployed throughout the network. When IoT sensors generate data, it is analysed via one of these nodes without having to be sent to the cloud. In this case, motion detection can be carried out in fog device using *detectMotion()* or a message can be sent to the cloud alternatively. Return messages are then sent back to the smart camera which acts as an IoT sensor and actuator of the system. These tasks are carried out in fog device and return vehicle location using *getVehicleLocation()* method and send PTZ params with *getPTZParams()* method. Alternatively, the *getDetectedVehicle()* method returns a message from the cloud to the Traffic Surveillance application when the current task was sent to the cloud, and to successfully carry out this task, a background image should be extracted from the original video image.

3.5.4 Activity Diagram of the Proposed Application Model

Figure 3.4 shows an activity diagram of our proposed model and how the tasks are being carried out for each instance of the Vehicle recognition.

The UML Activity Diagram demonstrated in Fig. 3.4 depicts the workflow of stepwise activities and the control flow of the proposed application model. It also defines how activities are coordinated to offer a service that can be at various levels of abstraction. Typically, each event of the Traffic Surveillance Application

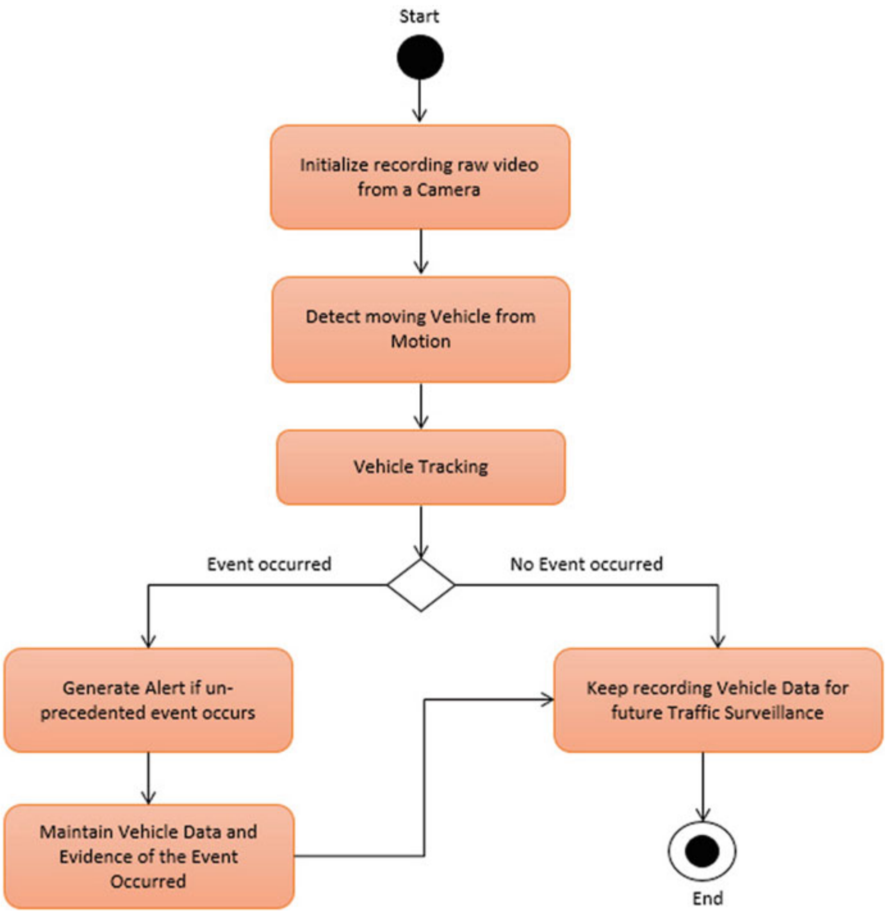
Start
Initialize recording raw video from a Camera
Detect moving Vehicle from Motion
Vehicle Tracking
Event occurred
No Event occurred
Generate Alert if un-precedented event occurs
Keep recording Vehicle Data for future Traffic Surveillance
Maintain Vehicle Data and Evidence of the Event Occurred
End

Fig. 3.4 UML activity diagram of proposed application model

needs to be accomplished by some operations, mainly where the operation needs to accomplish some different tasks. For the initial node which starts the beginning of a set of actions or activities, a black dot is used. The first object node of the activity initializes the recording of a raw video stream which is done using the Smart Video Camera. The following node represents the recognition of a vehicle from a video that contains a detected motion and then commences vehicle tracking by the last node. To control the flow of operation, a decision node is used to represent a test condition in a case where an unusual event is detected from a vehicle. If no event is occurred, a video camera continues to record video data for future surveillance, then the final node is used to stop all the control flow inactivity.

3.6 Performance Evaluation

3.6.1 Why Using iFogSim?

iFogSim is a toolkit to model, simulate and evaluate networks of fog computing, edge computing and Internet of Things (IoT). This framework offers a platform for analysing and evaluating the performance of applications. To model IoT applications, various simulator tools used in the fog computing environment are available in the literature. Some of these tools are: EmuFog [16], FogNetSim++ [17], FogTorchII [18], to name a few. Each of these simulation tools could have been used to test our proposed model, but we have selected iFogSim [19–21] for the following reason: iFogSim extends the CloudSim simulator by adding new functionalities [2], and as such, it is a preferred simulation toolkit for fog computing widely used by most researchers. Another reason is that iFogSim is written in the form of a Java-based open-source software package, which employs the JSON file format to represent the physical topologies, hence making it a package of choice to model real-time applications following the object-oriented paradigm. It also supports the simulation of entities and services. In iFogSim, the communication between entities/modules and the sharing of information between entities rely on a message passing mechanism. As such, iFogSim is a good platform to develop a model and test it with different metrics such as energy consumption, network usage and latency. On the other hand, the architecture of iFogSim offers the physical, logical and management components. The physical component includes the fog devices, the actuators and sensors, whereas the logical component includes the processing modules which are used to run tasks. The user of the system can draw, define the model, and build their topologies using a user-friendly graphical user interface (GUI). Alternatively, the user can define and write the topologies using a Java API which comes with the simulator. The management component can be used to schedule and monitor the application.

In our proposed model, we have chosen iFogSim because we have designed a tree-like topology that consists of a cloud data centre, gateways and smart cameras. We have designed the physical topology and defined it programmatically using Java APIs. The characteristics of each device and server have been defined using Java classes inherited from CloudSim.

3.6.2 Configuration Setup

The simulation of the Traffic Surveillance application was conducted using Window 10 Pro with Intel Core i5 processor, 8GB. The simulation was conducted for three sets of network topology configurations which are referred to as Configuration 1, Configuration 2 and Configuration 3, respectively. In Config 1, one area with one surveillance camera, Config 2, one area with two smart cameras, and lastly Config 3, with one area with three smart cameras. Each topology configuration was evaluated by using the four scheduling algorithms mentioned in Sect. 3.3, namely FCFS, SJF, GP and RR separately. The performance of these scheduling algorithms on vehicle tracking-task scheduling is evaluated based on average latency, energy consumption, execution time and network usage.

3.6.3 Simulation Results

Average Loop Delay

Figure 3.5 shows the performance of the studied four scheduling algorithms on the average latency for different configurations.

In Fig. 3.5, it can be observed that the latency increases as the number of areas surveilled also increases. In Config 1, all algorithms show a low latency below 100 milliseconds, and as the workload increases, the average latency also increases. This indicates that more fog devices are needed for effective processing, which in turn will minimize the time taken in the control loop to search for an efficient node. According to these results, one needs to add more fog devices to the network to minimize the latency as the workload increases. FCFS yields a very low latency in all three configurations when compared to other scheduling algorithms.

Energy Consumption

Figure 3.6 shows the performance of the studied scheduling algorithms on the energy consumption for different configurations. It is observed that as the number of smart cameras used in each configuration increases, the energy consumed also

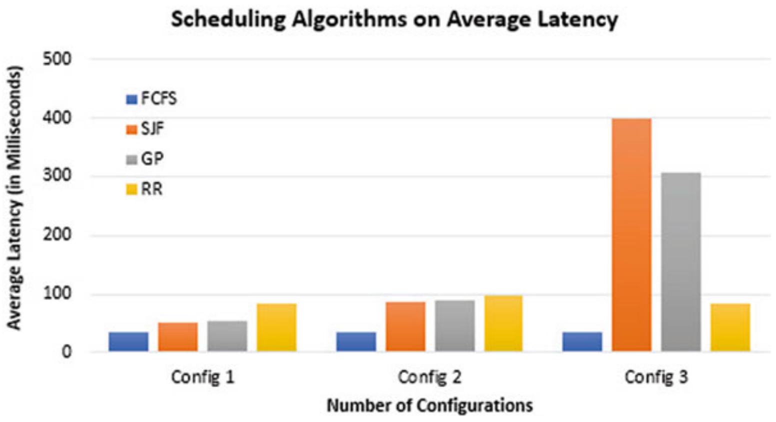

Fig. 3.5 Scheduling algorithms on average latency

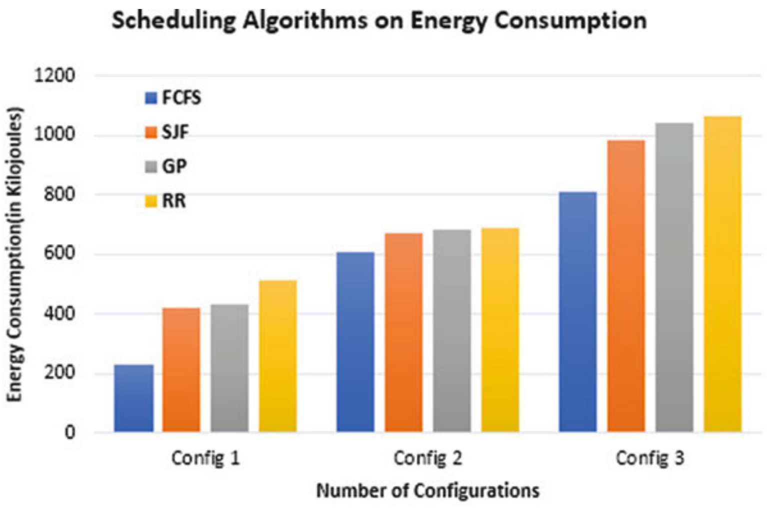

Fig. 3.6 Scheduling algorithms on energy consumption

increases. It also shows that to minimize the energy consumption in the network, fog devices must be added. Also, it is found that FCFS outperforms the other algorithms. In addition, RR appears to consume more energy while SJF and GP use almost equal energy in all setup configurations. These results also show that the best way to minimize the energy usage of our Traffic Surveillance application model is to execute the tasks in a queue and process them in the order of their arrivals.

Execution Time
Figure 3.7 shows the performance of the studied four scheduling algorithms on the execution time. It is observed that FCFS executes faster when compared to SJF and GP for the first two configurations. However, for Config 3, RR appears to outperform SJF and GP. It is also found that FCFS outperforms the other scheduling algorithms. RR appears to be less optimal in all configurations, this could be attributed to the fact that each ready task runs turn by turn only in a cyclic queue for a limited time, which may result in longer execution time compared to that obtained using other scheduling algorithms.

Network Usage
Figure 3.8 shows the performance of the studied four scheduling algorithms on network usage.

It is observed that as the number of surveillance cameras increases, several fog devices need to be added, which yields an increase in network usage. Also, the RR and GP algorithms appear to be superior to the other scheduling algorithms while SJF outperforms FCFS. Moreover, it is found that the tuple size and delay have a massive influence on network usage.

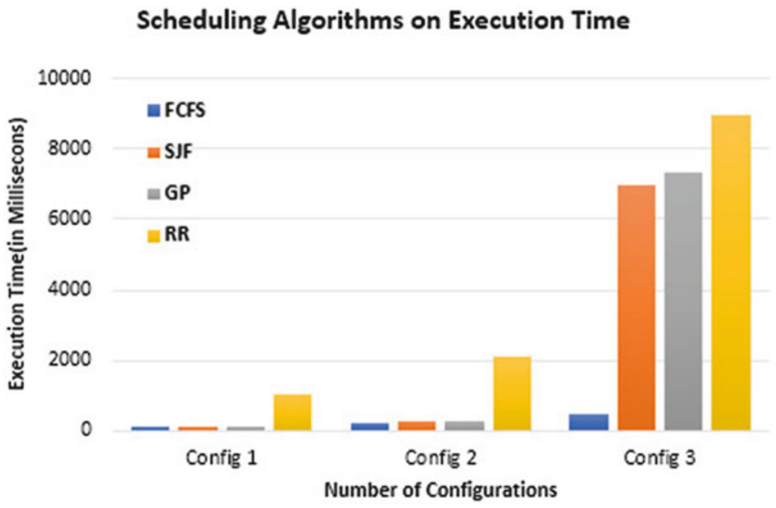

Fig. 3.7 Scheduling algorithms on execution time

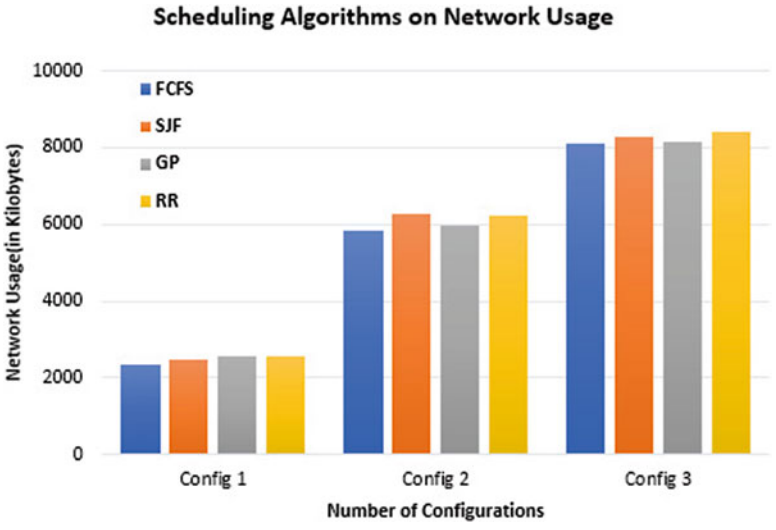

Fig. 3.8 Scheduling algorithms on network usage

3.7 Conclusion

In this chapter, a traffic surveillance model for detection and tracking of vehicles was proposed and simulations were conducted using iFogSim to evaluate its performance. The simulation results have shown that the FCFS algorithm outperforms the three other algorithms in terms of average latency, network usage, energy consumption and execution time, making it a suitable candidate for use in real-time IoT applications which require emergency response. In future work, we plan to simulate the proposed Traffic Surveillance application model in terms of resource usage and energy consumption in the scenario of multiple object tracking. We also plan to validate the effectiveness of our proposed Traffic Surveillance application model by assessing it against other available Traffic Surveillance application models, chosen as benchmark models.

References

1. Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems, 29*(7), 1645–1660. https://doi.org/10.1016/j.future.2013.01.010
2. Calheiros, R. N., Ranjan, R., Beloglazov, A., & De Rose, A. F. (2011). CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience, 41*, 23–50.
3. Hong, K., Lillethun, D., Ottenwälder, B., & Koldehofe, B. (2013). Mobile Fog: A programming model for large – Scale applications on the Internet of Things. In *MCC '13: Proceedings of the second ACM SIGCOMM workshop on mobile cloud computing* (pp. 15–20).
4. Chiu, C.-C., Ku, M.-Y., & Wang, C.-Y. (2010). Automatic traffic surveillance system for vision-based vehicle recognition and tracking. https://doi.org/10.6688/JISE.2010.26.2.17.
5. Sinqadu, M., & Shibeshi, Z. S. (2020). Performance evaluation of a traffic surveillance application using iFogSim. In *Lecture notes on data engineering and communications technologies* (Vol. 51, pp. 51–64). Springer Science and Business Media Deutschland GmbH.
6. Guevara, J. C., & Da Fonseca, N. L. S. (2021). Task scheduling in cloud-fog computing systems. *Peer-to-Peer Networking and Applications, 14*, 962–977. https://doi.org/10.1007/s12083-020-01051-9
7. Intharawijitr, K., Iida, K., & Koga, H. (2016). Analysis of fog model considering computing and communication latency in 5G cellular networks. https://doi.org/10.1109/PERCOMW.2016.7457059.
8. Oueis, J., Strinati, E. C., & Barbarossa, S. The fog balancing: Load distribution for small cell cloud computing. In *IEEE vehicular technology conference* (Vol. 2015). https://doi.org/10.1109/VTCSpring.2015.7146129
9. Aazam, M., & Huh, E. (2015). Dynamic resource provisioning through fog micro datacenter. https://doi.org/10.1109/PERCOMW.2015.7134002
10. Deng, R., Lu, R., Lai, C., & Luan, T. H. Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing. In *IEEE international conference on communications* (pp. 3909–3914). https://doi.org/10.1109/ICC.2015.7248934
11. Pham, X. Q., & Huh, E. N. (2016). Towards task scheduling in a cloud-fog computing system. https://doi.org/10.1109/APNOMS.2016.7737240
12. Zeng, D., Gu, L., Guo, S., Cheng, Z., & Yu, S. (2016). Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system. *IEEE Transactions on Computers, 65*(12), 3702–3712. https://doi.org/10.1109/TC.2016.2536019
13. Gibet Tani, H., & El Amrani, C. (2018). *Smarter round robin scheduling algorithm for cloud computing and big data*. https://hal.archives-ouvertes.fr/hal-01443713
14. Mtshali, M., Africa, S., Adigun, M., Dlamini, S., & Mudali, P. Multi-objective optimization approach for task scheduling in fog computing. https://doi.org/10.1109/ICABCD.2019.8851038
15. Rahbari, D., & Nickray, M. Low-latency and energy-efficient scheduling in fog-based IoT applications. https://doi.org/10.3906/elk-1810-47

16. Mayer, R., Graser, L., Gupta, H., Saurez, E., & Ramachandran, U. *EmuFog: Extensible and scalable emulation of large-scale fog computing infrastructures*. https://github.com/emufog/emufog
17. Qayyum, T., Malik, A. W., Khattak, M. A. K., Khalid, O., & Khan, S. U. (2018). FogNetSim++: A toolkit for modeling and simulation of distributed fog environment. *IEEE Access, 6*, 63570–63583. https://doi.org/10.1109/ACCESS.2018.2877696
18. Brogi, A., Forti, S., & Ibrahim, A. (2017). How to best deploy your fog applications, probably. In *Proceedings - 2017 IEEE 1st international conference Fog Edge computing ICFEC 2017* (pp. 105–114). https://doi.org/10.1109/ICFEC.2017.8
19. Gupta, H., Vahid Dastjerdi, A., Ghosh, S. K., & Buyya, R. (2017). iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Software: Practice and Experience, 47*(9), 1275–1296. https://doi.org/10.1002/spe.2509
20. Naas, M. I., Boukhobza, J., Raipin Parvedy, P., & Lemarchand, L. (2018). An extension to iFogSim to enable the design of data placement strategies. In *2018 IEEE 2nd international conference on fog and edge computing, ICFEC 2018 - In conjunction with 18th IEEE/ACM international symposium on cluster, cloud and grid computing* (pp. 1–8). IEEE/ACM CCGrid. https://doi.org/10.1109/CFEC.2018.8358724
21. Mahmud, R., Narayana Srirama, S., Ramamohanarao, K., & Buyya, R. (2019). Quality of Experience (QoE)-aware placement of applications in Fog computing environments. *Journal of Parallel and Distributed Computing, 132*, 190–203. https://doi.org/10.1016/j.jpdc.2018.03.004