Proceedings of the 2022 Winter Simulation Conference
B. Feng, G. Pedrielli, Y.Peng, S. Shashaani, E. Song, C.G. Corlu, L. H. Lee., E. P. Chew, T. Roeder and P. Lendermann, eds.

USING DEEP LEARNING FOR SIMULATION OF REAL TIME VIDEO STREAMING APPLICATIONS

Abdolreza Abhari
Dipak Pudasaini

Distributed Systems & Multimedia Processing
Laboratory (DSMP lab)
Department of Computer Science, Ryerson University
350 Victoria Street
Toronto, Ontario M5B 2K3, CANADA

ABSTRACT

The traditional approaches for simulation of video analytics applications suffer from the lack of real-data generated by employed machine learning techniques. Machine learning methods need huge data that causes network congestion and high latency in cloud-based networks. This paper proposes a novel method for performance measurement and simulation of video analytics applications to evaluate the solutions addressing the cloud congestion problem. The proposed simulation is achieved by building a model prototype called Video Analytic Data Reduction Model (VADRM) that divides video analytic jobs into smaller tasks with fewer processing requirements to run on edge networking. Real data generated from VADRM prototype is characterized and tested by curve fitting to find the distribution models for generating the larger number of artificial data for resource management simulation. Distribution models based on real data of CNN-based VADRM prototype are used to build a queueing model and comprehensive simulation of real-time video analytics applications.

1 INTRODUCTION

The sizes of video files are large and need more processing power for processing. A powerful system is needed to process large volumes of video files. Therefore, the combination of edge computing and cloud computing technology is powerful for the video analytic system. Edge computing is a distributed computing system closer to the data source. It is the emerging technology near the Internet of Things (IoT) devices. It is an advanced technique to minimize high latency and reduce bandwidth consumption.

Fog/edge computing is the extension of cloud computing toward IoT devices. It has small physical resources such as low processing servers or terminals with low storage capacity (Yousefpour, Ishigaki, and Jue 2017). A video analytic application is usually processed by Graphics Processing Unit (GPU)/Central Processing Unit (CPU)-based edge servers close to the security camera to track the moving objects and extract trajectories by analyzing video frames. Cloud computing is then used for further processing, such as pattern recognition of the extracted trajectories of moving objects. The division of the video analytic application into subtasks has been presented in (Pudasaini and Abhari 2020) but did not use simulation techniques in different scenarios. This research will use an edge network to solve common video analytics problems and find a scalable solution for processing data at the edge level by using simulation.

In a common video analytic system, video data is directly transferred to the cloud for detection and recognition (Anjum et al. 2019). The problems of this approach are high latency and more network bandwidth to transfer data into the cloud. Examining alternated methods and finding a solution in a real

system is very expensive. In the video analytics applications, the tasks are required huge processing and that is why they called edge computing killers (Ananthanarayanan 2017). In our previous research (Pudasaini and Abhari, 2021) a prototype was implemented as a proof of concept that shows dividing a big video analytic task into subtasks is possible. In the proposed method, the large data processing of video analytics is addressed by taking advantage of the fact that these applications can be divided into different tasks in a common video analytics application. The process of dividing video analytic tasks into subtasks is called Video Analytic Data Reduction Model (VADRM). In this paper real data generated by VADRM prototype is used in the workload characterization and simulation. The main contributions are given below.

1- Implementing VADRM video analytic pipeline mode and data collection: we implement a VADRM using deep learning based on edge computing that divides video analytic jobs into smaller tasks with fewer processing requirements for edge-enabled video analytic applications.
2- Using the real data in iFogSim Simulator: The real video files are passed into the VADRM to collect data from each module. The data created from the VADRM modules are passed into the iFogSim simulator to measure the system's performance
3- Workload characterization and analysis of the distribution models of VADRM data: The real data created from the modules of VADRM used by curve fitting to analyze the empirical distributions.
4- Designing a queuing model and general simulation with artificial data: The analytical models of video analytic applications are formulated and used by a queuing model, and two scenarios for the simulation of real-time video streaming applications are conducted.

2 RELATED WORKS

There is not much research about the workload characterization of new deep learning-based video analytics applications and their performance measurement. However, there are increasing numbers of solutions for improving edge computing. Edge computing focuses on bringing cloud computing services and utilities closer to the user for fast processing of data-intensive applications (khan et al. 2019). The edge computing approach is used to offload computing applications, data, and services from cloud computing to the network's edge (shi et al. 2016). The emerging technologies in cloud computing ensure to delivery of cloud services for mobile computing, scalability and privacy policy enforcement for the Internet of Things (IoT), and the ability to mask transient cloud outages (Satyanarayana. 2017).

In (Merenda, Porcaro, and Iero 2020), the machine learning models in the edge network have been reviewed. It has presented the architectures and requirements for the solutions in implementing edge machine learning in IoT devices. Cloud-fog-edge computing model for video surveillance based on modular arithmetic has been proposed by (Kuchukov, Nazarov, and Vashchenko 2020). It has presented the distributed computing model in which some parts of video analytics tasks are processed in the fog devices and large sizes of tasks are processed in the cloud devices. The service migration from cloud to multi-tier fog devices has been proposed in (Rosário et al. 2018). The integration of cloud computing with fog networks does not support Quality of Experience (QoE) with high demands for multimedia content. This paper has proposed the operational impacts and benefits related to service migration from cloud to fog. These related works have not used real data generated from machine learning into the simulation.

Job scheduling and resource management are important and edge-enable applications. When a job arrives at the dispatching center, based on the arrived job's size and the processors' condition (in terms of busy or idle), the decision should be made about the processor that the job should be dispatched on it. So, dispatching jobs in a Fog/Edge environment is an online and real-time problem. Sharifi et al. presented a Queueing-based model (Sharifi, Abhari, and Taghipour 2021) and a mixed-integer linear programming (MILP) model (Sharifi, Abhari, and Taghipour 2021) to optimize the dispatching strategy for a non-real-time problem. In this regard, they have considered that the arrival time and size of the jobs are deterministic and known at the beginning of the system's mission horizon. Although this was a non-realistic assumption for the real-time applications that all data about jobs are available, the results of our previous works can help researchers develop better dispatching strategies than the existing ones when jobs are waiting in the queue and their

information can be analyzed. In this paper, the assumption is job scheduling is done in real-time with a marginal waiting time for jobs in the queue.

3 DEEP-LEARNING DATA COLLECTION METHODOLOGY

VADRM prototype is developed as a sample of the video analytic applications that reduce the size of video files in the consecutive phases of video file processing. This prototype divides the video analytic process into different tasks suitable for edge computing architecture. The centralized processing methods are not appropriate for processing scalable video files. In our previous paper, we have explained how VADRM reduces the size of video files for edge or cloud-based network (Abhari and Pudasaini 2022). This paper explains the data collection methodology used in CNN-based modules of the VADRM pipeline. The pipeline of VADRM is shown in Figure 1 where data is passed into the different modules of VADRM (i.e., motion detection, object detection, etc.). VADRM uses Convolutional Neural Network (CNN) to process videos in each module such as object detection and object tracking phases which are explained later.

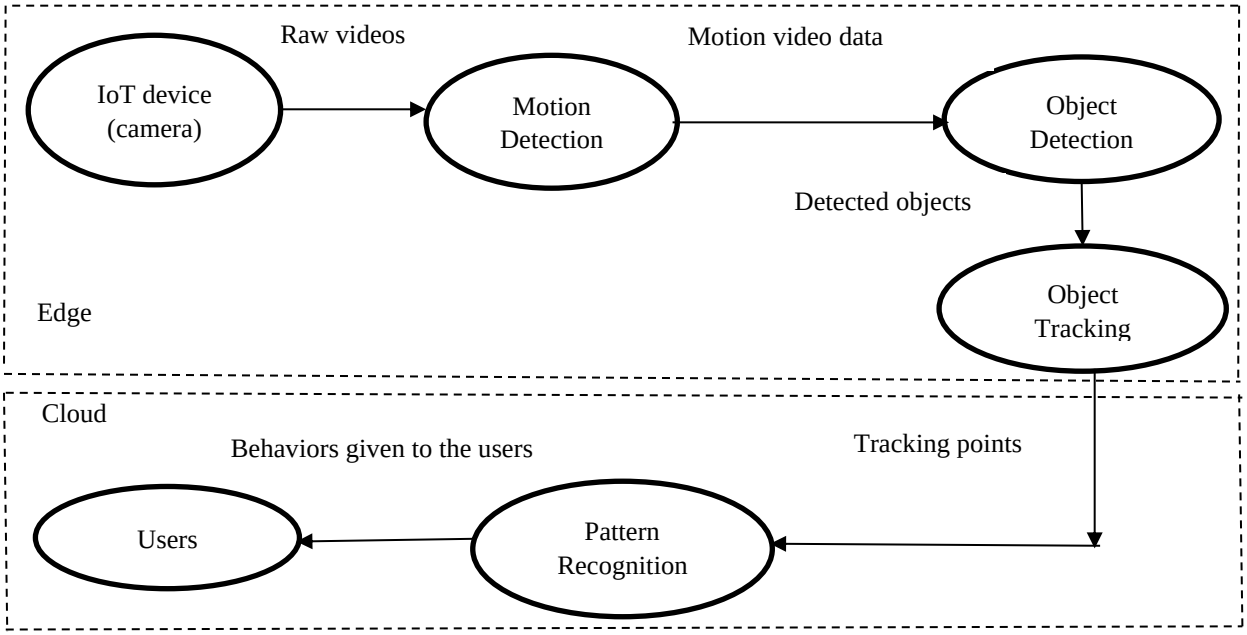

Figure 1: Video analytics model using edge and cloud computing.

Similar to most video analytics applications VADRM splits the big video analytics jobs into four stages: motion detection (s1), object detection (s2), object tracking (s3), and pattern recognition (s4). Figure 1 shows a video camera that captures the motion of the vehicles and passes the data to the modules which are stored in the local servers. There are a number of local servers in different places that act as edge devices to store the motion of the vehicles. Finally, the edge devices detect and track the trajectories and send them to the cloud for pattern recognition. If edge devices are busy, data for the required VADRM is sent to the cloud for the required processing.

3.1 Data Collection in VADRM Modules

The data sent to VADRM modules are collected separately in four stages. These four stages are four modules which are motion detection (s1), object detection (s2), object tracking (s3), and pattern recognition (s4). Next, a brief explanation and the nature of the data in each module are presented.

3.1.1 Motion Detection (s1)

The data from the video camera is continuously sent to this module which captures the motion of the vehicles. Then the captured raw video data (i.e., frames) will be forwarded to an object detection module.

3.1.2 Object Detection (s2)

The motion detection module passes video data to the object detection module. Object detection extracts features of the objects to detect them. Multiple objects are detected in this module. In this work, the deep learning CNN-based technique, You Only Look Once (YOLO) method is used to detect the objects.
YOLO v3 object detector is used for vehicle detection in our experiment. YOLO v3 is a fully convolutional object detection architecture used in many real-time applications (Redmon and Fardhai 2018), (Redmon et al., 2016). This model uses a special backbone network called the darknet. This network is chosen because of its high-speed frame per second (fps) and good accuracy. We can choose any object detection model in place of YOLO v3 for the model. It consists of 24 convolutional layers followed by 2 fully connected layers. The convolutional layers are pre-trained with the ImageNet dataset that has used the Darknet framework.

3.1.3 Object Tracking (s3)

The object tracking module receives the results from the object detection module. Then, the object tracking module tracks the path of moving objects to extract their trajectories. Video frames are fed into a robust object detector model. The object detector outputs the class categories and bounding boxes for the objects of interest in the video frames. The bounding boxes and the regions of interest in the frames are cropped and passed to the appearance model. The appearance model outputs the feature description for each detected object. The Deep SORT method leverages these feature descriptions for the association of the detected objects with tracks in addition to the previous SORT-based association, which uses the Kalman filter to predict the objects' location in the next timestamp. For the CNN architecture for object tracking, we used the residual network architecture (Zagoruyko and Komodakis 2016). In this architecture, first two convolutional layers are used and then pass into six residual blocks with the same patch size and different strides. The samples of screenshots after detection and tracking are shown in Figure 2.

3.1.4 Pattern recognition (s4)

The tracking module generates the tracking points and then forwards them to the pattern recognition module. The patterns of the moving objects are normal or abnormal behaviors that will be sent to the users. The data sent to this module is smaller than the other modules. Our previous research proposed trajectory pattern recognition methods based on supervised and unsupervised learning (Pudasaini and Abhari, 2019).

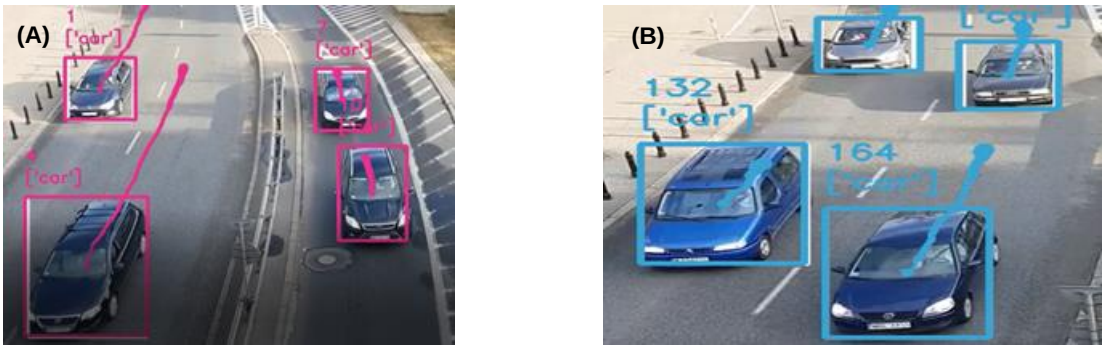

Figure 2: Two samples (A)-(B) of object tracking.

3.2 Collected Dataset

The real data are collected from the VADRM modules described above, and analyzed using a curve fitting models. The unit of data is frames collected in different phases of the video analytic model collected by passing different sizes of video files to VADRM, which are divided as follows:
Motion Detection: 9.21, 8.88, 9.51, 9.26, 9.12, 9.01, 8.95, 9.11, 8.96, 9.03, 8.99, 9.02, 9.30, 9.22, 9.25, 9.14, 9. 17
Object Detection: 7.56, 7.50, 7.65, 7.90, 7.67, 7.66. 7.88, 7.77, 7.85, 7.62, 7.48, 7.69, 7.83, 7.56, 7.49, 7.72, 7.81
Object Tracking: 6.14, 6.0, 6.11, 6.17, 6.09, 6.19, 6.05, 6.13, 6.14, 6.12, 6.23, 6.15, 6.25, 6.29, 6.05, 6.17, 6.22

3.3 VADRM Resource Management

We assume VADRM (the prototype in this work) employed a dispatcher and when the video arrives in the system it splits into smaller tasks and places them in the dispatcher. Then the dispatcher device will distribute the tasks over the available servers. Thus, the optimization is achieved by VDRAM splitting the big jobs into different tasks, each task is sent to the edge server, which is suitable for that specific job. The decision to assign tasks to different servers is based on the common FCFS strategy for the general model and the edge ward (Gupta et al. 2017) strategy when using the iFogSim simulator. When all edge devices get busy, and there are still jobs in the system that need to be served, the central controller (i.e., dispatcher) sends a job to the cloud. The queued jobs at the dispatcher will be separated into subtasks and scheduled to different edge devices with suitable computing abilities. Therefore, it is essential to evaluate the current capacity of the edge devices and later estimate the subtask completion time. The centralized cloud's total cost must be estimated if the job is scheduled to the centralized cloud. The computing resources cost and bandwidth resource cost comprise the total cost measured by simulation. The VADRM is simulated both with real data in by iFogSim simulator and by analytical modeling with random data. The simulation results are explained in sections 4 and 5, respectively. We created two models based on two scenarios of VADRM resource management which are explained in section 5.

In the first scenario, since the system has one dispatcher it is also modeled with G/G/m queueing system. The simulation of two scenarios is done for a simplified network of three edge devices, and one cloud. In both scenarios, the dispatcher sends the jobs to the cloud whenever edge devices are busy. The second scenario is a simulation of the pipeline and real-time video streaming with a hard deadline.

4 VADRM NETWORK SIMULATION USING IFOGSIM WITH REAL DATA

The iFogSim simulator simulates the VADRM model and its real data for the real-time scenario. When the input video is passed into the VADRM, it will be divided that job into a batch of tasks with three different types of jobs in the size of frame/sec. This batch of the jobs will be dispatched among edge and cloud devices each second because of the hard deadline of the real-time video streaming scenario. The capacity of edge devices is based on MIPS. The scalar factor 150 is considered to convert frames into Million Instructions (MI) because the original work of iFogSim has the MI value 0-5000 for edge devices. Based on these real data generated by VADRM, the simulation of sending them to the devices in three scenarios. These three scenarios in the simulator are based on the involvement of edge and cloud devices. The VADRM model is based on edge computing and cloud computing. We will compare this model with edge-only systems, and cloud-only systems. We calculate the bandwidth, latency, and resource utilization which are shown in Figure 3 to Figure 5. Here we placed edge devices in three places with one camera each.

The choice of configuration values is based on the minimum requirement of video surveillance in the real-world scenarios that are referred from the iFogSim Simulator. The tuple CPU length value is based on our VADRM real implementation. Table 2 describes the configuration of application module components in the video analytic application. Table 1 describes the latencies configuration between the source and the destination nodes. Table 3 presents the capacities of fog nodes and the cloud. It presents the size of different devices in our physical topological structure. It also presents the network capacity of fog devices. Based on the values of data achieved by VADRM, we set the values in the iFogSim simulator to find the network performance. The real 50 video data are passed into iFogSim then find the average value as the final results.

Table 1: Network latencies configuration.

Between	**Latency (Milliseconds)**
Cloud to Proxy server (VADRM dispatcher)	100
Proxy server to fog devices	50
Fog devices to camera	1

Table 2: Values of video analytic application.

Tuple types	**Tuple CPU length (MIPS)**	**Network length (Bandwidth)**
Raw video stream	1368	2000
Motion video stream	1152	2000
Detected objects	921	1000
Tracking points	737	800

Table 3: Capacity of fog nodes and cloud.

Device	**CPU (MIPS)**	**RAM (Bytes)**	**Up bandwidth (Mbps)**	**Down bandwidth (Mbps)**
Cloud	40100	40000	1000	10000
Proxy server (VADRM dispatcher)	2700	40000	10100	10100
Fog device	2600	4000	1100	11000
Camera	1300	1000	200	200

The performance metrics measured from the simulator are network bandwidth, resource utilization and latency. The comparison of network bandwidth, resource utilization, and latency in different scenarios are explained in Figure 3 to Figure 5.

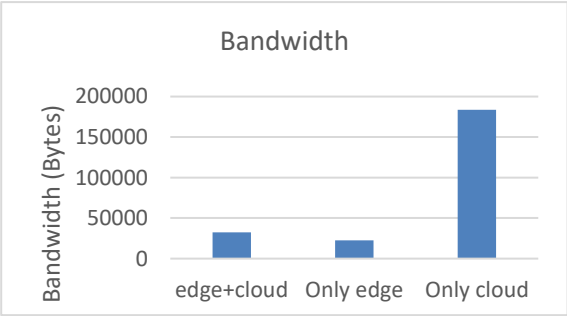

Figure 3: Comparison of bandwidth.

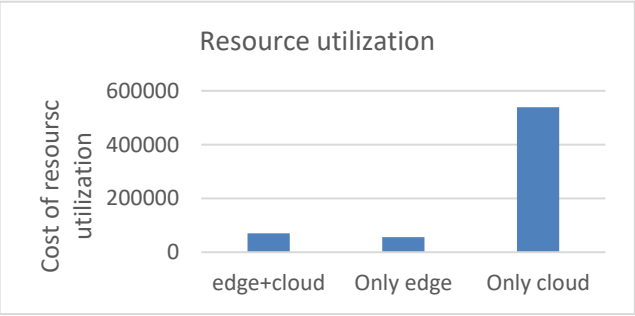

Figure 4: Comparison of resource utilization.

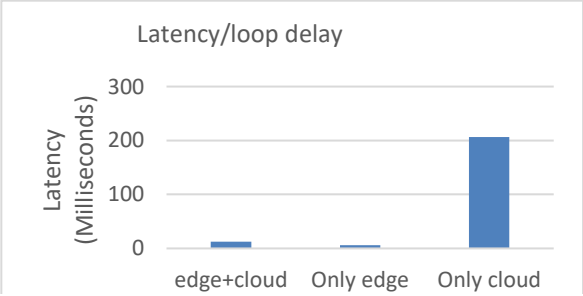

Figure 5: Comparison of latency on real video.

The results show that the network bandwidth, resource utilization, and latency are extremely reduced by using edge and cloud, which is around more than 81% than traditional Cloud-based system.

5 VADRM SIMULATION WITH ARTIFICIAL DATA

The real data collected from VADRM (Video Analytic Data Reduction Model) are defined in section 3 which are used for the ifogsim simulation. In this section, we conduct a simulation when using random data generated by VADRM. The distribution models and simulation results are explained in this section.

5.1 Distribution Models and Workload Characterization

Our developed video analytic prototype (i.e., VADRM) generates the data in each phase. The number of frames in each phase of the video analytic model have been collected as described in the previous section. The distribution of these data is determined by curve fitting and analyzing empirical distribution. The analysis is performed in python with 51 values of the frames that are generated from each module of VADRM when receiving the real video streaming. The empirical and a model are shown in Figure 6.

The empirical distribution from the video analytic model is compared to many candidate distributions. The best-fitting is the Beta distribution which has the shape for the small parameter. The Beta distribution is used to model the continuous random variables, suitable to model the fractions of the data. The Uniform distribution is the second-best distribution of the given data. The confidence scores of Beta and Uniform distribution are 7.35 and 8.65, respectively which shows the errors and the lower is better. Based on these scores, the Beta and Uniform distributions are more appropriate for modeling these data.

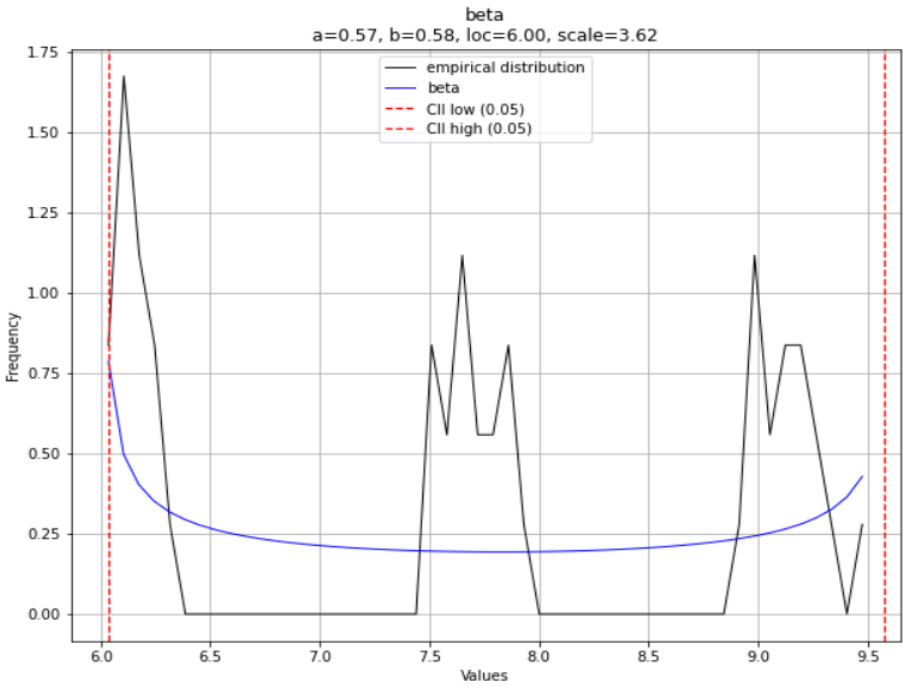

Figure 6: Beta distribution.

5.2 Queuing Modelling for the VADRM with Uniform Distribution of Data

The video analytics applications are complex systems; therefore, using simulation is a better way to analyze these systems. For measuring the performance of VADRM as a sample of video analytics applications, the queuing modeling can be used only by considering the simplest scenario of the systems in which job

interarrival times and service times are independent of each other. In that case, we can use G/G/m queueing system in which job arrival and service times have different distributions than Poisson and Exponential distributions with *m* homogenous servers. From the previous section, considering the job sizes in the unit of frames with the Uniform distribution, we can assume service time is also Uniform distribution. Thus, the service time is proportional to the job sizes in the boundary of *a* and *b* seconds. Service time with a Uniform distribution and the range (a,b) has a mean time of s= $(a+b)/2$ seconds. The Uniform service time and λ average arrival rate (per second) result in the following G/G/m queue parameters:

$$\text{Average arrivale rate (average interarrval time)} = \frac{1}{\lambda} \quad (1)$$

$$\text{For average service time s, the average service rate } \mu = \frac{1}{s} = \frac{2}{a+b} \quad (2)$$

$$\text{Utilization of servers (showing the probability of working servers)} \rho = \frac{\lambda}{m\mu} = \lambda\frac{(a+b)}{2m} \quad (3)$$

$$\text{The average number of jobs in the system } E[n] = m * \rho = \frac{\lambda}{\mu} = \lambda\frac{(a+b)}{2} \quad (4)$$

The assumption is when the number of jobs is more than edge servers, the cloud can serve any incoming jobs in parallel. In this system, the waiting time in the queue is marginal and we don't calculate that. Instead, this queuing model is used to determine how the system behaves in the steady-state and when the number of servers and arrival rate increase. To show scalability of the system by using the above equations (1 to 3), the changes in the utilization factor when the number of servers increased can be shown by Figure 7. The increase in the arrival rate when increasing the number of servers is shown in Figure 8.

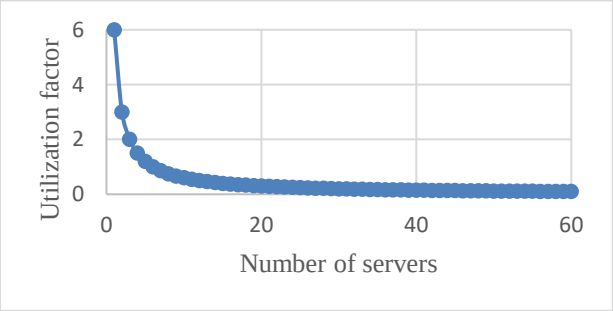

Figure 7: Utilization factor vs. no of servers.

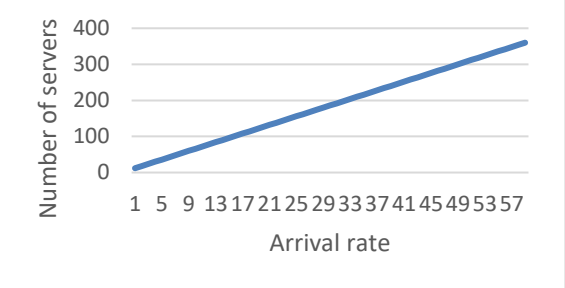

Figure 8: Job arrival rate vs. no of servers.

The service time Uniform distribution parameters are proportional to the number of frames of job sizes. For example, the value of the service rate is 1/6 for considering the range of a + b, equal to 12 seconds for the service time. For this service rate, as shown in Figure 7, the utilization factor ρ only becomes stable (i.e., $\rho \leq 1$) when the number of servers increases to 6 (which is the knee of the curve). Then after having six servers, the ρ decreases more slowly as the number of servers increases. It means for such a system to be stable the minimum number of servers is 6 which is equal to (a+b)/2. By calculating Equation 4, the average number of jobs in the system in the steady state is also 6. The Utilization factor ρ also shows the percentage of time the servers are busy and 1- ρ shows severs are idle. Figure 7 shows in a system with the average arrival rate of one job (in the average of 6 frames size) in each second if the network has three identical edge servers, with the average service time of 6 seconds each, then adding 3 more processors from the cloud provides close to 100% of the times of servers to be busy and adding more servers are not useful. Figure 8 provides information about the system under heavy load. It shows that by increasing the number of servers as the multiplication of the arrival rate to minimum of the required servers ($m = \lambda * (a+b)/2$) when the job arrival rate increases sequentially, the system is stable (i.e., $\rho \leq 1$).

Waiting time in G/G/m queues, can be found by M/M/m approximation (Whitt 1976) (Medhi 2003) and assuming inter-arrival times is exponentially distributed. Considering λ is the mean rate of arrival and σ_a^2 is the variance of the inter-arrival time. And μ is the service rate and σ_s^2 is the variance of service time.

$$C_a^2 = \frac{\sigma_a^2}{(\frac{1}{\lambda})^2} \qquad and \qquad C_s^2 = \frac{\sigma_s^2}{(\frac{1}{\mu})^2} \tag{5}$$

$$W_q(G/G/m) = W_q(M/M/m)\ \frac{C_a^2+C_s^2}{2} \tag{6}$$

The equation (6) is waiting time of the queue. By assuming interarrival time of jobs in this example has exponential distribution with mean of 1 second inter-arrival time and λ =1 average arrival rate, then by calculating waiting time in the M/M/m queue and by using equations (5) and (6) following graphs can be achieved for queue waiting time.

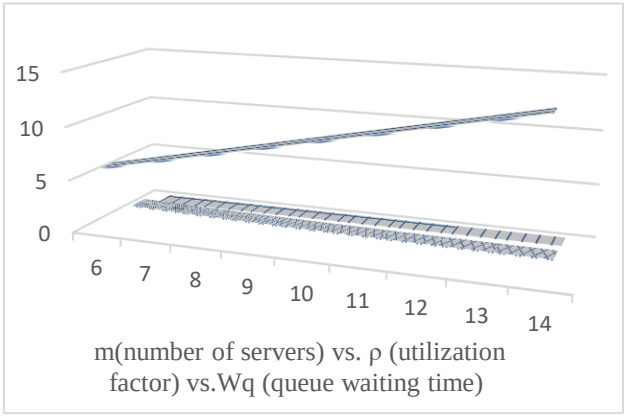

Figure 9: No of servers vs. ρ vs. Wq.

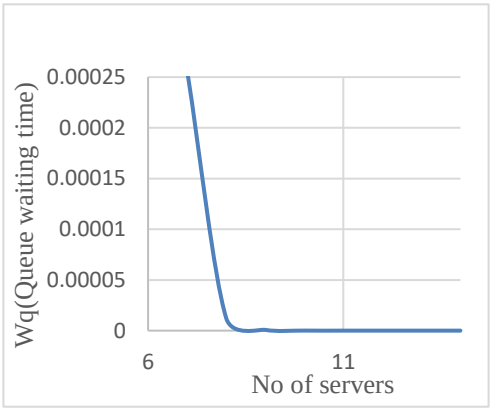

Figure 10: Wq(G/G/s) vs. no of servers.

Figures 9 and 10 show waiting time of jobs in the queue in this system is negligible and with the increase of the number of servers for more than 6 servers (i.e., (a+b)/2), it converges toward zero.

5.3 Simulation of VADRM Real Time Scenarios with Beta Distribution of Data

In this part, a general case for VADRM is simulated. We assume all the jobs are split into smaller tasks that edge devices can process by generating 36000 random job sizes using Beta distribution. In this general model simulation, VADRM sends spiltted jobs to a central controller in every second (i.e., video streaming data coming from the camera) to be allocated to the computing nodes through the edge and cloud network. For simplicity, we assumed it takes one second to dispatch the job to each device. When a job is completed at each node, a 'complete' message will be sent to the controller by that node, and the dispatcher or controller knows the device is available and can receive another job. We consider two scenarios of this problem when we have three weak edge devices (can be IoT devices or simple laptops) called Edge Processing1 (EP1), Edge Processing2 (EP2), Edge Processing3 (EP3), and also a cloud computing facility, all connected in a synchronized clock network with good bandwidth that creates a one second delay for transferring video streaming jobs to each device. We use a simple strategy for a job assignment, we can sort the EPs based on the power of the EP in terms of frames per second (fps) processing power that they have for video streaming to assign the job to them to finish faster. So the first arrived job should be dispatched to the first idle EP, and if there is no Idle EP, the job should be dispatched to the cloud (that is called CP). There is no queue before the three EPs and one cloud device. The only difference between cloud devices and EPs is that the cloud can get any job in parallel, but EDs should finish their allocated job to become free and accept new jobs. If EPs had a queue, the problem is too easy to handle because there was no job to dispatch to the CP, and all EPs could handle all the dispatched jobs with a potentially long queue behind them. But in video analytics it is a real-time system the jobs cannot wait and since Eps don't have a

queue if there is no EP available the incoming jobs should be dispatched to the cloud until one EP becomes free. We simulated two scenarios for this real-time video streaming problem:

A) **First Scenario**: The first scenario is by considering a soft deadline to simulate the real time video streaming on each device. It means every second when one job is coming, we assigned it to an available device. It is simulating a star topology when the dispatcher is in the center of the edge network.

B) **Second Scenario**: In the second scenario, we use a hard deadline. It means every second we assign four jobs to all devices. This scenario simulates the pipeline or ring network topology similar to the VADRM implementation in which we assume the dispatcher allocates the job after receiving the complete signal to the next available node.

For both scenarios, we can use a first-in-first-out strategy for job arrival and sorted the processors based on their processing speed (fps) from high to low. In both scenarios of simulation, we consider the same jobs coming in the max 1-hour time frame for job arrival and assignment. So, we have 3600 jobs in the first scenario, which arrive at the dispatching center by a one-second lag. The size of the jobs has a Beta distribution between five to eight frames/sec. In the first scenario, we have to compute the finishing time of all 3600 jobs. For the second scenario, these 36000 jobs arrive in the batch of four jobs per second and will be finished in 900 seconds. We consider three EPs with the processor capacity powers (shown by fps) of average sizes of randomly generated job sizes a CP that can accept any number of jobs with different sizes and run them in parallel. Tables 4 and Table 5 show the detailed results of the simulation and Figures 11 to 14 show the graphs of resource utilization by the number of sizes of allocated jobs in each scenario.

Table 4: Results of the simulation (Scenario1).

Devices	**Number of allocated jobs**	**Size of allocated jobs (Mbytes)**	**Rate of the idle times in %**
EP1	9169	334.07	12.1
EP2	6684	324.74	14.5
EP3	4548	331.36	12.8
CP	15599	113.66	70.1

Table 5: Results of the simulation (Scenario2).

Devices	**Number of allocated jobs**	**Size of allocated jobs (Mbytes)**	**Rate of the idle times in %**
EP1	8806	87.13	6.6
EP2	7454	68.15	23.1
EP3	4208	36.03	57.8
CP	15532	127.82	26.4

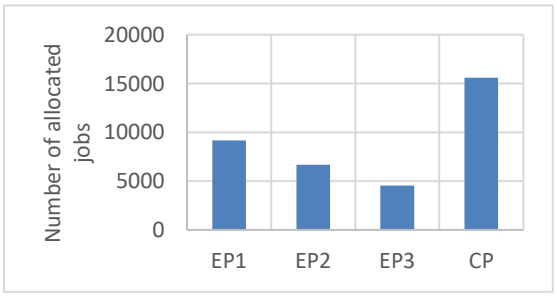

Figure 11: Allocated jobs 1st scenario.

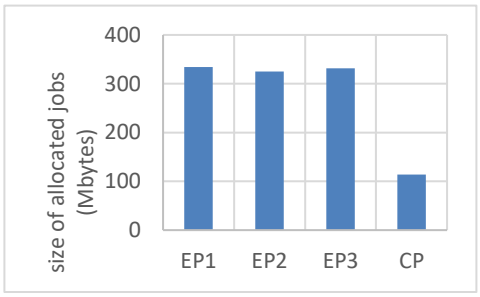

Figure 12: Size of the allocated jobs 1st scenario.

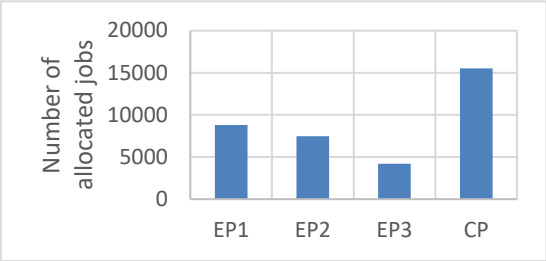

Figure 13: Allocated jobs 2st scenario.

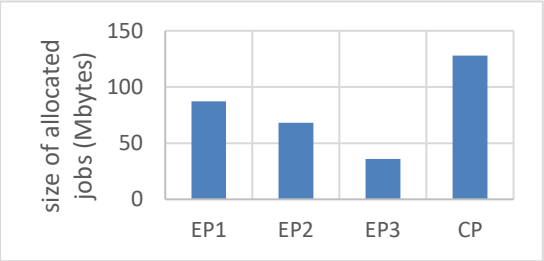

Figure 14: Size of the allocated jobs 2st scenario.

This analytical model shows that the size of allocated jobs in edge devices is 57% and cloud devices are 43%, respectively, as shown in Figure 11 for the first scenario. Figure 13 which is related to the number of allocated jobs in the second scenario is almost similar. Figure 12 shows that the percentage of sizes of jobs allocated in cloud devices is 10%, whereas the edge devices which is 90% in the first scenario. For the second scenario as shown in Figure 14 the size of allocated jobs to the cloud is higher than the first scenario which is 40%. The size of jobs allocated in edge devices is 60%. These results conclude that when there is a hard deadline in real-time applications, bigger jobs will be assigned to the cloud to meet the deadline.

6 CONCLUSION

The novel methodology used in this paper is for the performance measurement of the current video analytics application. It suggests building a system prototype to generate artificial data for a comprehensive simulation. The developed prototype is a sample of deep-learning-based video analytics applications that divides the video analytic application into many small tasks with less processing requirements, called VADRM. The workload characterization used curve fitting based on the real data generated by the implementation of VADRM shows that Beta and Uniform distributions are the best fit for generating the random job sizes when VADRM is used as the video analytics solution. The random job sizes generated by Uniform distribution is used to analyze the system under heavy load by a G/G/m queuing model, showing that when the job arrival rate increases, since the cloud runs the job in parallel, the system is stable.

Generating artificial data by Beta distribution is used to simulate video analytic applications developed based on VADRM. A comprehensive simulation was performed for two real-time video analytic application scenarios to verify job allocation on processing nodes. These results show that edge devices for both scenarios process around 57% of the jobs. In terms of job sizes, the results show that the size of jobs assigned to edge devices for both scenarios is around 60% to 90% of the total sizes of the jobs.

Based on both prototype model simulation with iFogSim with real data, we conclude that the VADRM model can potentially improve network performance (i.e., bandwidth and latency) by 81% and reduce the size of jobs in the cloud by at least 60% for real-time video analytics application. This improvement can be done by employing edge devices by using fog computing in the real system. This work shows that building a prototype model is a helpful technique for evaluating machine learning-based video analytics applications, where data is insufficient to simulate both network and resource management in the cloud and edge-based computing. The data achieved by VADRM that employed CNN-based method was used to generate random data for the realistic simulation and performance measurement.

REFERENCES

Abhari, A., and D. Pudasaini. 2022. “Video Analytic Data Reduction Model for Fog Computing”. In *Proceedings of Annual Modeling and Simulation Conference (ANNSIM)*, July 18th -20th, San Diego, CA, US, 1-12.

Ananthanarayanan, G., P. Bahl, P. Bodik, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha. 2017. “Real-Time Video Analytics: The Killer App for Edge Computing”. *Computer* 50(10):58-67.

Anjum, A., T. Abdullah, M. Tariq, Y. Baltaci, and N. Antonopoulos. 2019. "Video Stream Analysis in Clouds: An Object Detection and Classification Framework for High Performance Video Analytics". *IEEE Transactions on Cloud Computing*

7(4):1152-1167.

Gupta, H., A. V. Dastjerdi, S. K. Ghosh, and R. Buyya. 2016. "iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in the Internet of Things, Edge and Fog Computing Environments*". Software Practice & Experience* 47(9): 1275- 1296.

Khan, W. Z., E. Ahmed, S. Hakak, I. Yagoob, and A. Ahmed. 2019. "Edge Computing: A Survey". *Fourth Generation Computer Systems* 97: 219-235.

Kuchukov, V., A. Nazarov, and I. Vashchenko. 2020. "Cloud-fog-edge Computing Model for Video Surveillance Based on Modular Arithmetic". *IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*. January 27th -30th, St. Petersburg, Russia, 374-376.

Medhi. J. 2003. Stochastic Models in Queueing Theory. *Academic Press*, Amsterdam, Second edition.

Merenda, M.,C. Porcaro, and D. Iero. 2020."Edge Machine Learning for AI-Enabled IoT Devices:A Review". *Sensors* 20(9):1-34.

Pudasaini, D., and A. Abhari. 2020. "Scalable Object Detection, Tracking and Pattern Recognition Model Using Edge Computing". In *Proceedings of Spring Simulation Conference (SpringSim)*. May 18th-21st, Fairfax, VA, US, 1-11.

Pudasaini, D. and A. Abhari. 2021. "Edge-based Video Analytic for Smart Cities". *International Journal of Advanced Computer Science and Applications* 12(7): 1-10.

Pudasaini, D., and A. Abhari. 2019. "Scalable Pattern Recognition and Real Time Tracking of Moving Objects". *Spring Simulation Conference (SpringSim)*. April 29th-May 2nd, Tucson, Arizona, US, 1-11.

Redmon, J., and A. Farhadi. 2018. YOLOv3: An Incremental Improvement. https://pjreddie.com/media/files/papers/YOLOv3.pdf. accessed 8th April 2018.

Redmon, J., S. Divvala, R. Girshick, and A. Farhadi. 2016. You Only Look Once: Unified, Real-Time Object Detection. https://arxiv.org/pdf/1506.02640.pdf. accessed 9th May 2016.

Rosário, D., M. Schimuneck, J. Camargo, J. Nobre, C. Both, J. Rochol, and M. Gerla. 2018. "Service Migration from Cloud to Multi-tier Fog Nodes for Multimedia Dissemination with QoE Support". *Sensors* 18(2):1-17.

Satyanarayanan, M. 2017. "The Emergence of Edge Computing". *Computer* 50(1):30-39.

Sharifi M., A. Abhari, and S. Taghipour. 2021. "A Queueing Model for Video Analytics Applications of Smart Cities". *In Proceedings of the 2021 Winter Simulation Conference*. December 15th -17th, Phoenix, Arizona, US, pp. 1-10.

Sharifi, M., A. Abhari, and S. Taghipour. 2021. "Modeling Real-Time Application Processor Scheduling for Fog Computing". *Annual Modeling and Simulation Conference (ANNSIM)*. July 19th – 22nd, Fairfax, Virginia, US, 1–12.

Shi, W., J. Cao, Q. Zhang, Y. Li, and L. Xu. 2016. "Edge Computing: Vision and Challenges". *IEEE Internet Things*, 3: 637–646.

Whitt, W. 1983. "The queueing network analyzer". *Bell System Technical Journal* 62(9):2779-2815.

Yousefpour, A., G. Ishigaki, and J. P. Jue. 2017. "Fog Computing: Towards Minimizing Delay in the Internet of Things". *IEEE 1st International Conference on Edge Computing*. June 25th – 30th, Honolulu, Hawaii, US, 17-24.

Zagoruyko, S. and N. Komodakis. 2016. "Wide residual networks," *BMVC*, 1–12.

AUTHOR BIOGRAPHIES

ABDOLREZA ABHARI is a Professor in the Department of Computer Science at Ryerson University and director of the DSMP lab (http://dsmp.ryerson.ca). He holds a Ph.D. in Computer Science from Carleton University. His research interests include web social networks, data science, AI and agent systems, network simulation, and distributed systems. His email address is aabhari@ryerson.ca.

DIPAK PUDASAINI is a Ph.D. student in the Department of Computer Science at Ryerson University and an Assistant Professor at Tribhuvan University. He holds an M. Sc. in Computer Science from Ryerson University. His research interests include computer vision, data science, machine learning, web service selection, and simulation. His email address is dpudasaini@ryerson.ca.