A real-time logo detection system using data offloading on mobile devices

Jiacheng Shang & Jie Wu

CYBER-PHYSICAL SYSTEMS
https://doi.org/10.1080/23335777.2018.1499674

ARTICLE

A real-time logo detection system using data offloading on mobile devices

Jiacheng Shang[a] and Jie Wu[a]

[a]Center for Networked Computing, Temple University, Philadelphia, PA, USA

ABSTRACT

In the past few years, mobile augmented reality (AR) has attracted a great deal of attention. It presents us a live, direct or indirect view of a real-world environment whose elements are augmented (or supplemented) by computer-generated sensory inputs such as sound, video, graphics or GPS data. Also, deep learning has the potential to improve the performance of current AR systems. In this paper, we propose a distributed mobile logo detection framework. Our system consists of mobile AR devices and a back-end server. Mobile AR devices can capture real-time videos and locally decide which frame should be sent to the back-end server for logo detection. The server schedules all detection jobs to minimise the maximum latency. We implement our system on the Google Nexus 5 and a desktop with a wireless network interface. Evaluation results show that our system can detect the view change activity with an accuracy of 95.7% and successfully process 40 image processing jobs before deadline.

ARTICLE HISTORY
Received 6 June 2018
Accepted 30 June 2018

KEYWORDS
Augmented reality applications; mobile computing; mobile offloading

1. Introduction

In the past few years, mobile augmented reality (AR) has attracted a great deal of attention. It presents us a live, direct or indirect view of a real-world environment whose elements are augmented (or supplemented) by computer-generated sensory inputs such as sound, video, graphics or GPS data. With the help of advanced AR technology (e.g. computer vision and object detection), information about the user's real-world surroundings becomes interactive and digitally manipulable. Many AR devices and systems, such as Microsoft HoloLens, Sony SmartEyeglass and Google Glass, have recently been released. In this paper, we consider the following AR application: a user goes shopping with an AR device (e.g. Google Glass). The AR device captures real-time video and recognises logos. Based on the recognised logos, the AR device displays corresponding deals and promotion information to the user.

CONTACT Jiacheng Shang jiacheng.shang@temple.edu Center for Networked Computing, Temple University, Philadelphia, PA, USA

To support this application, the system should provide accurate logo detection and recognition using limited resources. For example, the AR device should extract important logo frames and label them correctly with an identifier. However, achieving accurate, real-time logo detection and recognition on mobile AR devices is challenging because of limited computation ability and battery life. Current computer vision algorithms tend to require a lot of computation resources, like graphics processing unit (GPU).

To achieve accurate, real-time logo detection and recognition on mobile AR devices and minimise latency, we need to offload some tasks to the back-end server. Mobile data offloading, often known as Wi-Fi offloading, is the use of complementary network technologies that deliver data originally targeted for cellular networks. Through proper data offloading, we can reduce the amount of data carried on the cellular bands and free bandwidth for other users. However, we must still be careful about what data should be offloaded to the server since wireless network latencies are usually high and the server can only support a limited number of jobs. In our system, users want to receive detection results and related information before losing the logo in the camera. Servers, on the other hand, need to determine the order to execute different clients' tasks based on their urgency. The system must also prevent bad users from pretending that their tasks are urgent.

In this paper, we design a distributed, continuous and real-time logo detection system based on data offloading. At the mobile AR devices side, our system can reduce the latency of the AR devices by offloading computationally intensive jobs (feature extraction and classification) to the back-end server. Our system also reduces bandwidth consumption by only offloading key frames that may have new logos. In order to extract important frames that may contain useful logo information, we design a view change detection model by leveraging embedded mobile sensors. At the server side, we implement a deep learning model for logo detection based on you only look once (YOLO) [1] and design a deadline-driven scheduling algorithm to minimise the maximum lateness. In order to prevent some 'bad users' from continually marking their logo detection requests as urgent, we further design a utility function for all users. Each user initially owns limited 'money' and must pay for each logo request. We implement our system on the Google Nexus 5 and a desktop with a wireless network interface. Evaluation results show that our system can support about 40 offloaded jobs at the same time.

The remainder of this paper is organised as follows: In Section 2, we will introduce the system pipeline, design challenges and proposed solutions. We will discuss our local model in Section 3, including the view change detection, important frame queue and priority assignment. The offloading model will be defined and introduced in Section 4. In Sections 5 and 6, we will introduce our experimental implementation and analyse the evaluation results. We discuss the generality and limitation of our system in Section 8. The final conclusion and future work will be given in Section 9.

2. System design

In this section, we describe the logo detection pipeline, its challenges and our solutions.

2.1. *Logo detection pipeline*

When we capture a video frame from the system, the logo detection pipeline consists of four stages: view change detection, priority assignment, job scheduling and logo detection, as shown in Figure 1 and 2.

2.1.1. *View change detection*

In our experiment, we find that new logos tend to appear when a user frequently changes the camera's view. Based on this observation, we use mobile devices' motion sensors to detect view change activity. Then, frames appearing after a view change are labelled as important frames and are candidates for further classification.

2.1.2. *Priorities assignment*

This stage assigns priorities and deadlines to all important frame candidates. Our observation is that a users' frame should be assigned higher priority if the logos will soon be out of view of the camera.

2.1.3. *Job scheduling*

After receiving frames that contain important logo information from all clients, we assign different deadlines for the buffering of these frames based on their priorities. Then, an earliest deadline scheduling-based algorithm is designed to minimise the maximum lateness.

2.1.4. *Logo detection*

This stage finds the location of object in uploaded frames and assigns labels to them. In our system, we implement YOLO on labelled object detection training data set. Once a detection task is scheduled based on its deadline, the frame is passed to the objected detection model. The detection result is then sent to the mobile devices directly before the user loses the corresponding object.

2.2. *Challenges and solutions*

When we are designing, the naive idea is to implement all the functions on local mobile AR devices. However, because of the limited resources, mobile AR devices cannot guarantee good performances for real-time object detection (e.g. logo detection). Based on previous works [2], running object detection on the mobile device can be 11 times to 21 times slower than running it on a typical server machine, feature extraction can be 18 times slower and detection can be 14 times slower. As a result, we need to offload computationally intensive detection works to the server to reduce the processing time. Energy consumption is another important issue for mobile AR devices. Based on Chen's experimental results [2], the energy consumed by executing the entire pipeline on the device is 12 times to 21 times more than the energy consumed when each frame is offloaded to the server. Since the current mobile AR devices have limited power, it is better to offload as much work as possible to the server to reduce local energy consumption.

At the same time, we want to reduce the bandwidth usage. Ideally, every frame should be sent to the server for further processing and detection. However, this will waste bandwidth and server resources since some frames may not have a logo or multiple frames may have the same logo. We need to determine locally which frame is important so that mobile devices can send only important information to the server. It is also difficult to determine the important frames locally since running new logo detection on mobile devices usually needs at least of hundreds of milliseconds, which is not practical in real AR applications. In our system, we propose a local model to determine which frame is important by monitoring view change events. The key observation is that the camera often captures new logos when user's view changes. After offloading detection jobs to the back-end server, a proper scheduling strategy should be designed to provide good performance for most connected users. In our system, we adopt the earliest deadline first scheduling strategy to minimise the maximum latency for all users.

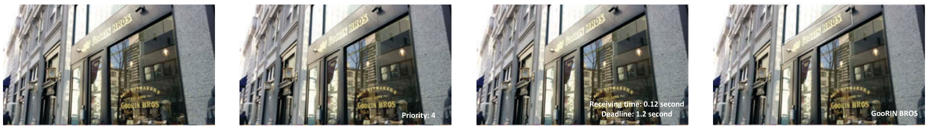
3. Local model

In this section, we will introduce the designs of our view change detection model, the important frame queue and the priority assignment.

3.1. *View change detection*

The goal of view change detection is to find frames that imply that a user's view has changed in a real-time video. Selected frames are candidates to that will be offloaded to the roadside server.

Our approach to detecting view changes is based on the analysis of the data from motion sensors. When a user changes the view by turning around, this activity also influences the motion sensors embedded in the mobile device (headsets or smartphones). In order to understand how to view change events influence motion sensors, we collect data from accelerometers and gyroscope



(a) View change detection (b) Priority assignment (c) Deadline assignment (d) Logo detection and recognition

Figure 1. The four computationally intensive stages of the object detection pipeline: (a) View change detection, (b) priority assignment, (c) deadline assignment and (d) logo detection and recognition.

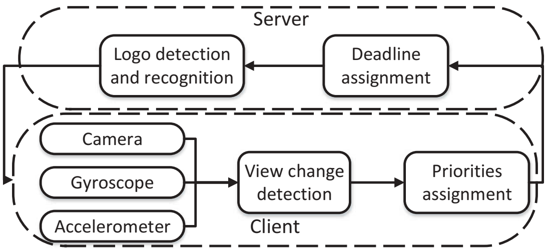
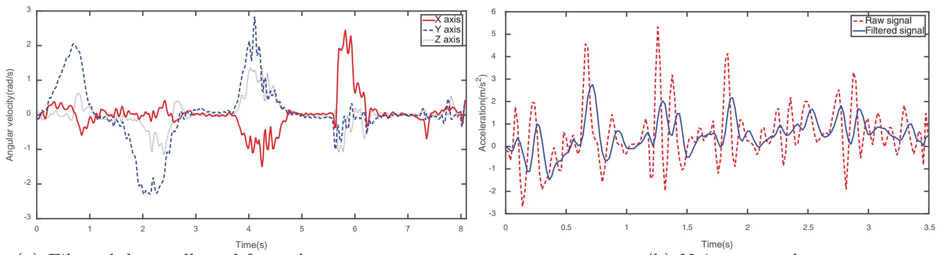


Figure 2. System architecture.



(a) Filtered data collected from the gyroscope sensor (b) Noise removal

Figure 3. Sensor data processing: (a) Filtered data collected from the gyroscope sensor and (b) noise removal.

sensors embedded in smartphones. The influence that turning around has on filtered gyroscope sensor data is illustrated in Figure 3(a). We can observe that the waveforms in three axes are nearly steady if a user does not turn around since there is no significant angle change. When a user tries to turn around, the smartphone rotates a little with the motion of the user's head, leading to the significant peak in the Y and X axes. To detect a real-time view change event caused by turning around, we first apply a moving average window with a size of 30 to the raw gyroscope sensor data on the Y or X axes. Based on our experiments, we find that turning around usually takes about 1 s so we adopt a moving window with a size of 0.6 s in all experiments. Within each window, we use the algorithm described in Algorithm 1 to check for a significant peak with an absolute magnitude higher than 1 rad/s. If we find a peak in a moving window, the latest video frame is marked as an important frame.

In addition to turning around, human views also change due to motions. Based on our experiments, human views totally change about every 3 m in a shopping scenario. We collect the accelerometer sensor data for the Z axis from a walking person, and the results are illustrated in Figure 3(b). Because the walking activity is periodical, collected accelerometer sensor data also vary periodically across time. We calculate the double integral of acceleration data to get the coarse distance estimation. To obtain the distance estimation, we apply a high-pass filter on the raw signal and velocity signal. The low-frequency component of the signal appears as DC components, so generated velocity and displacement signals become decreasing (sometimes increasing) curves. The first and second integration results are shown in Figure 4(a,b). We can see that our distance estimation model can get accurate results based on raw sensor data. Then, we use the estimated distance to determine the important frames. More specifically, if the distance exceeds 3 m, the latest video frame is marked as an important frame.

3.2. *Important frame queue*

Based on the view change detection results, we get multiple important video frames. Ideally, all the important frames detected should be offloaded to the roadside server. However, lots of important frames may be detected in a short period due to abnormal user activities (e.g. when a user is frequently shaking his or her head). To limit the average rate of offloading important frames, we adopt a FIFO (first in, first out) queue with a fixed size. Every 2 s, the system uploads all the frames in the FIFO queue to the server. After that, the FIFO queue is erased. There is a trade-off on how to set the queue size. If the size is large, the system is more robust to noisy data, but the server cannot handle all the requests from all users. If the size is small, the server can support more users, but the system can be easily influenced by noisy sensor data. Based on

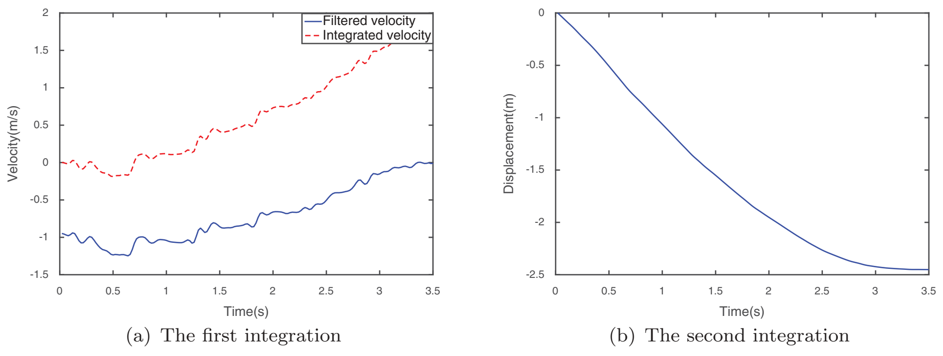


Figure 4. Signal processing of accelerometer data: (a) The first integration and (b) the second integration.

our experiment settings and shopping scenario, we set the queue size to 2 to get the best performance for all users in our experiments.

```
ALGORITHM 1: View change detection on one axis: source code
Input: A moving window $W = (w_1, w_2 \ldots, w_N)$.
Output: Whether there is a view change.
1: while There is a new gyroscope sample $g$ do
2:   if Window $W$ if full. then
3:     Insert $g$ to the end of $W$ and delete $w_1$.
4:     if $(|g - w_1| > 1)$ then
5:       Report there is a view change detected.
6:       remove all samples in $W$.
7:   else
8:     Insert $g$ to the end of $W$.
```

3.3. *Priority assignment*

Though the system detects multiple important video frames, we argue that these frames should have different priorities for scheduling. For example, if a user is walking at a high speed, his or her frames should have higher priorities for scheduling at the server side to ensure that the user can get his or her prediction results before losing the logos. We argue that a user's important frames should be assigned a higher priority if the user is very likely to lose logos in those frames. If a user changes the view frequently or significantly, detected important frames will be assigned high priorities. If not, we will look at user's average speed in a period. The higher the average speed is, the higher the priority that is assigned. Based on data collected from motion sensors, we assign each frame a priority from 1 to 5, where 5 means the highest priority.

Fairness is another issue we need to consider for our priority strategy. Imagine a 'bad' user frequently changes the view in order to always get a high priority for his or her important frames. This is not fair to users who always follow the rules since they cannot win a competition in the job scheduling game on the server side. To address this problem, we propose a credit-based approach. Every 150 s, each user is given 500 credits. The user needs to pay for each priority assignment based on its calculated priority, and the credit paid is the final priority assigned to that frame. The highest priority that can be assigned to a frame F_i is calculated in the following equation:

$$F_i = C \times \frac{2}{150 - t}$$

where t is the time from the beginning of each 150 s and C is the remaining credit. Then, the final priority assigned to frame P_i is

$$P_i = \frac{P_s}{4} \times F_i$$

where P_s is the calculated priority based on sensor data with a range from 1 to 5.

We can see that proposed priority assignment strategy can prevent 'bad' users from assigning all frames a high priority. If a user spends too much, he or she has to assign a low priority to all subsequent frames, which ensures that no one can occupy server resources all the time and achieves fairness among all users.

4. Offloading model

In this section, we will define our scheduling problem. Then, we formulate our optimisation problem to solve for the optimal scheduling strategy.

4.1. *Problem definition*

In this paper, we consider a wireless network with multiple mobile devices and one roadside server. Mobile devices can directly communicate with the roadside server via Wi-Fi. Every 2 s, the server will receive frame detection requests from all connected mobile devices and determine a scheduling sequence to minimise the maximum lateness.

Suppose that there are N video frames to be scheduled on a single machine. The machine can process at most one job at a time, and it must process a job until its completion once it has begun processing. Suppose that each job J_i must be processed for a specified P_i units of time, and the processing of job J_i may begin no earlier than a specified release date R_i, $i = 1, \ldots, N$. We assume that the schedule starts at time 0 and each release date is nonnegative. Furthermore, we assume that each job J_i has a specified due date D_j, and if we complete its processing at time C_i, then its lateness L_i is equal to $(C_i - D_i)$. We are interested in scheduling the jobs so as to minimise the maximum lateness L_m that is defined as:

$$L_m = \max L_i$$

4.2. *Priority and deadline*

When the server receives a video frame from a mobile device, the metadata includes the arrival date and assigned priority. In order to involve all the metadata, we need to translate the assigned priority to the processing deadline of a frame; i.e. we want to assign earlier deadlines to frames with higher priorities. Based on this intuition, we define the deadline of a job J_i as

$$D_i = \frac{(2 - R_i)}{P_i} + R_i$$

The priority of a video frame is from 1 to 5. When a frame has the lowest priority 1, its deadline will be 2 based on the above equation, which is the end of each cycle of the communication between mobile devices and the server. If a frame has the highest priority, 5, its deadline will be set to $0.8R_i + 0.3$. Based on the definition of R_i, $R_i \leq 2$ always holds, so we get that $(0.8R_i + 0.4) \leq D_i \leq 2$ holds all the time.

4.3. *Optimisation problem formulation*

Consider a real-time application where multiple users send tasks to the server and the arrival time and execution time of all tasks are acknowledged. Our goal is to find an execution sequence that minimises the maximum lateness for all users, that is,

$$\text{minimize} \qquad L_{\max} = C_i - \frac{(2-R_i)}{P_i} + R_i \text{subject to} 0 \leq R_i < 2 \tag{1}$$

4.4. *Solution*

```
ALGORITHM 2: Online earliest deadline scheduling: source code
    Input: A set of jobs $j$ with corresponding deadline $d_j$.
    Output: The order of execution.
    1: Construct a min-heap using current tasks based on their deadline.
    2: while The max-heap is not null do
    3:     if (Current time $t < 2$) then
    4:         Pick the task in the root to execute.
    5:         Add newly received tasks into the min-heap.
    6:     else
    7:         Terminate current processing and discard all remaining jobs.
```

Unfortunately, this problem is well known to be NP-hard, and in fact, even deciding if there is a schedule for which $L_{max} \leq 0$ is strongly NP-hard. Since we have limited prior knowledge of the arrival time sequence and the estimated execution time of each offloaded task, we adopt the earliest deadline first scheduling. Our solution is to maintain a priority queen for all currently offloaded tasks based on their deadlines using a min-heap. The server always picks the task with the nearest deadline to run. Each video frame has to be executed before the end of the cycle. If the execution time of a frame exceeds the given time slot, the processing of that frame is terminated, and the frame is also discarded. If the processing time exceeds the end of the cycle (2 s), the execution sequence is not feasible, and all frames that are not processed yet are discarded. The detailed algorithm is described in Algorithm 2.

5. Implementation

We have implemented the client on the Google Nexus 5 smartphone and the server on a desktop with a wireless network interface.

5.1. *Client*

We implement our client on a Google Nexus 5 smartphone with all source codes written in Java. More specifically, we use embedded signals to capture the sensor signals and real-time videos. The linear acceleration signal and gyroscope signal are sampled at about 50 samples per second. In order to filter out the irrelevant impact that remains the same during the distance estimation (like gravity), we adopt a three-order high-pass filter. Since the impact that comes from walking is also of extremely low frequency, we set the cut-off frequency as $\frac{1}{1700}$. For the view change detection model, we use a moving window with a size of 30 samples, which is about 0.6 s based on our observations. We let participants record videos and use extra tools to cut them into frames. The participants are responsible for labelling frames, and these frames are then used for the classification model training on the server.

5.1.1. *Server*

The server implements the object detection model and detection job scheduling pipeline. More specifically, we retrain YOLO [1] on PASCAL Visual Object Classes (VOC) data set on an NVIDIA TITAN X Graphics Card and test it on VOC 2007 test data set. We use multithread socket programming to receive images transmitted from the clients and to calculate corresponding deadlines.

6. Evaluation

In this section, we show the evaluation results of our view change detection, distance estimation, uplink transmission delay and overall system performance under different numbers of users.

6.1. *View change detection*

To evaluate the view change recognition performance of our system, we collect data from three people with a smartphone attached to their heads. Each participant is asked to change their view while they are standing or walking. The ground truth of the view change is recorded based on the video captured using the smartphone's front camera. We collect 100 instances from each user for view change detection with or without movement.

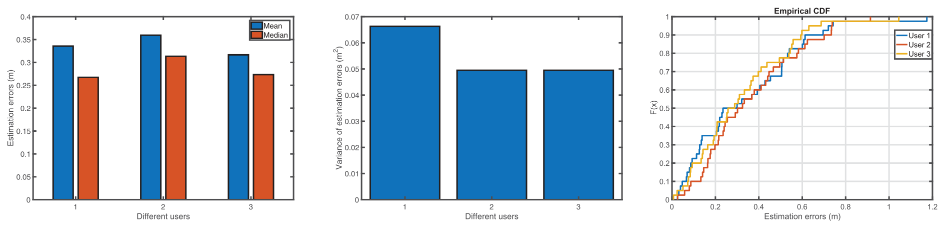
The evaluation results are shown in Table 1. We can observe that the recognition accuracy is almost 100% when the user does not move (walking or running). Even if the user walks at different speeds, the view change detection is still robust enough. The lowest detection accuracy is 93%. Based on the experimental results, we can say that our view change detection model can accurately detect view change events using the motion sensors embedded in mobile devices.

6.2. *Distance estimation*

To better understand the performance of our distance estimation model, we collect 120 measurements from three users and plot the distribution graph. The results are shown in Figure 5(c). We can see it clearly that most distance estimations have errors from under 0.6 m, which accounts for about at least 80% for all users. About 40% of the distance estimations have an error less than 0.26 m. Only about 6% of the total measurements have an error larger than 0.7 m. We further explore the mean, median and variance of the distance estimation collected from our three participants, which is illustrated in Figure 5 (a,b). We can see that the mean error is at most 0.36 m for all participants, and the median is at most 0.31 m. The variance is at most 0.663, which means that most estimation errors are close to the mean error. Since the estimated walking distance in our system is just a reference for important frame selection, we do not need accurate measurements. Considering human foot length

Table 1. View change detection performance.

Different users	Without movement	With movement
User 1	100%	98%
User 2	99%	93%
User 3	100%	96%
Overall	99.7%	95.7%



(a) Mean and Median (b) Variance (c) Error distribution

Figure 5. Distance estimation performance: (a) Mean and median, (b) variance and (c) error distribution ..

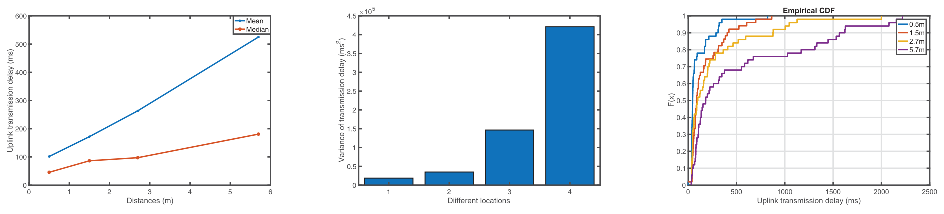
is about 0.27 m, a mean estimation error of about 0.35 m is acceptable. Based on the second rule of view change detection, we argue that the mean estimation error of 0.35 m is sufficient for detecting view change events in our local model.

6.3. *Uplink transmission delay*

Based on the definition of our scheduling problem, different jobs may have different release data. In this subsection, we evaluate the uplink transmission delay between our mobile devices and the back-end server. We let a laptop send a frame with a size of 154KB to the server. In our design, the server is connected to the router via a wired connection. We capture the Transmission Control Protocol (TCP) packets at both sides and compare the differences between their timestamps. The server and the laptop are synchronised to the same Network Time Protocol server in advance. We calculate the mean latency, median of delay and the variance of delay at different distances, and the results are listed in Figure 6(a,b). It is clear that the uplink transmission delay rises with the increase of the distance between the mobile user and the router. When the mobile user is close to the router, the uplink transmission delay is no more than 0.1 s in most cases. Also, the uplink transmission delay is more likely to be influenced by environmental changes with high distances. We further plot the distribution of the uplink transmission delay via Wi-Fi communication, as shown in Figure 6(c). We can observe that most transmission can be done in 0.5 s with a short distance, while only about 70$\%$ of the transmissions can be done in 0.5 s when the distance is 5.7 m.

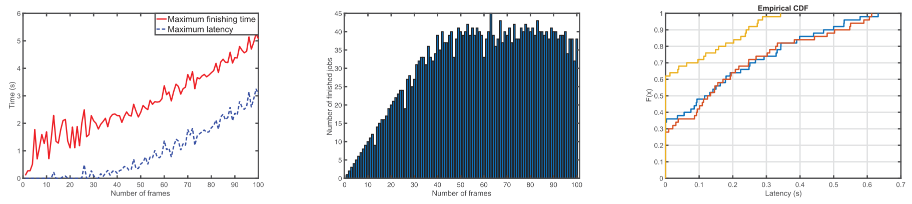
6.4. *Scheduling performance*

In this subsection, we further evaluate our scheduling algorithm. Based on the distributions of the uplink transmission delay and image processing delay, we conduct a simulation with different numbers of mobile devices. Figure 7(a) shows the maximum lateness and finishing time for different numbers of users.



(a) Mean and Median (b) Variance (c) Delay distribution

Figure 6. Uplink transmission delay: (a) Mean and median, (b) variance and (c) delay distribution.



(a)Maximum latency and finishing time (b) Number of finished jobs (c) distribution of latency

Figure 7. Evaluation results of our scheduling algorithm: (a) Maximum latency and finishing time, (b) number of finished jobs and (c) distribution of latency.

We can see clearly that both the maximum lateness and finishing increase with the growth of numbers of frames. This is in line with our expectations since the job executed last may need to wait for more jobs. Figure 7(b) shows the numbers of successfully processed frames for a different number of offloaded frames. It is clear that the maximum number of jobs one server can support is about 40. To support more frames and more users, a distributed system with multiple servers should be included. We also evaluate the distribution of latency for three times when the number of offloaded frames is 50, and the results are shown in Figure 7(c). We can see that about 90% of the total frames have a latency under 0.5 s, and all the frames can be processed no later than 0.7 s after their deadlines. This means that we can ensure a low processing latency for most of the offloaded jobs.

7. Related work

In this section, we introduce some existing works on object detection, mobile edge computing and recent AR frameworks.

7.1. *Object detection*

Object detection is to confirm the existence and location of an object in a video or picture. Object detection plays an important role in various applications such as video surveillance and AR. Many object detection models have been proposed for different applications. In [3], Zhao et al. proposed an approach for motion detection by increasing the weight of object information and also restraining the static background. The system proposed in [4] can detect and recognise objects by leveraging Deep Neural Networks. Ren et al. further reduced the running time of detection networks by sharing convolutional features between Fast R-CNN and Region Proposal Network [5]. SSD proposed in [6] adopt a relatively simple approach that completely eliminates proposal generation and encapsulates all computation in a single network.

YOLO [1] achieves good detection performance by dividing the image into regions and predicting bounding boxes and probabilities for each region.

7.2. *Mobile-edge computing*

Mobile-edge computing (MEC) is a new framework to provide co-locating computing and storage resources at the edge. In MEC systems, mobile end users offload resource-hunger tasks to the back-end server. Many works have been proposed to address computation offloading problem in MEC. In [7], Rudenko et al. found that significant power saving can be achieved by offloading certain tasks of realistic size. Gonzalo et al. in [8] proposed an adaptive offloading mechanism that leverages the execution history. They collect the consumed resources and the state of the device and use this information to perform an offloading. In [9], Huang et al. proposed a dynamic offloading algorithm based on Lyapunov optimisation to save energy and meet given application execution time requirement. Barbera et al. conducted various experiments in [10] and showed that wireless transmission affects the overall offloading performance in a large degree. Wu et al. considered network unavailability in mobile offloading in [11] and proposed a new offloading decision and application partition algorithms to minimise energy consumption and execution time. Wen et al. in [12] proposed a new offloading algorithm by configuring the clock frequency of the mobile devices to minimise the energy consumption. In [13], Xian et al. proposed a new offloading algorithm. By setting a timeout, their work can achieve good offloading performance without estimating the computation time in advance. These approaches only consider the single user computation offloading problem, which may be not practical in real applications.

Many systems are designed for offloading under the setting of multiple mobile users. In [14], Chen et al. formulate the distributed computation offloading decision-making problem under the setting of multiple mobile users as an offloading game. They showed that the offloading game admits a Nash equilibrium and possesses the finite improvement property. Some systems are designed specifically for consumer-oriented services that should put the end users in the first place. In [15], Zhang el al. proposed an algorithm that can find the best code partition and integration points.

7.3. *Recent AR frameworks*

Recently, various AR systems have been proposed for different networks. For example, in [16], Ran et al. propose a distributed framework that ties together front-end devices with more powerful back-end ‘helpers’ that allow deep learning to be executed locally or to be offloaded. A virtual reality (VR) video conferencing system over named data networks is proposed in [17]. The system is composed of three components: the producer, the consumer and the sync manager. To ensure the real-time requirement, it adopts a prefetching approach, in which the consumers request video chunks in advance [18] presents a system over a 5G network where a collection of base stations interconnected via back-haul links are sharing their caching and computing resources to maximise the aggregate reward they earn by wireless clients.

8. Discussion

8.1. *Generality of our system*

First, our view change detection model consists of generic techniques that can be applied to other mobile devices with embedded motion sensors. The image processing and detection model at the server's side can be replaced by any computer vision model for different applications (e.g. face detection). The scheduling problem we define is in line with the actual situation. Our view change model can be used to select key frames and reduce the bandwidth compared with existing approaches.

8.2. *Limitations*

Based on our current settings (one server and multiple mobile devices), only about 20 users and 40 frames are supported in each cycle. In real shopping scenarios, tens of users will exist in an area. To support more users, multiple processing cores should be included for the distributed processing, and the problem should be formulated again on multiple machines. Due to hardware constraints, we do not cover this part in this paper.

Moreover, we do not discuss the coordination among different users. When two users are close to each other, it is very likely that they will see same logos in a short time. Sometimes, a user's line of sight may be blocked by others. In this case, multi-hop communication can be adopted to realise real-time logo detection. Habak et al. proposed Femto clouds [19] that can offload computation to nearby mobile devices. Similarly, framework designed in [20] offloads computation among the peers within the tactical edge. Since mobile devices have limited computation resources, it is hard to implement these two systems for AR application. Currently, all mobile devices in our system have a queue of important frames with a fixed size. In real shopping scenarios, it may be better to set different sizes for different areas based on the distribution and density of users.

9. Conclusion

In this paper, we propose a real-time logo detection system for mobile devices. Our system can capture important frames in real-time videos and recognise logos in them. Since the vision algorithms for object detection entail significant computation, our system offloads these detection jobs to the roadside servers. To reduce the bandwidth, we select important frames by a view change detection model on the mobile devices. To minimise the worst performance for all offloaded jobs, we use a greedy algorithm at the server side to minimise the maximum lateness of all jobs. Various experiments are designed to evaluate our system. The evaluation results show that our system can truly reduce the bandwidth usage and maintain a great object detection performance using the view change detection model. Up to 10 users can be supported using the current settings (one server and multiple mobile devices). In the future, we want to further balance the workloads between the server and the mobile devices. For example, mobile devices can process images locally with battery consumption constraints as long as the processing time is less than the time spent on the offloading.

Acknowledgements

This research was supported in part by NSF grants CNS 1629746, CNS 1564128, CNS 1449860, CNS 1461932, CNS 1460971, CNS 1439672 and CNS 1301774.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work was supported by the Division of Computer and Network Systems [1301774, 1439672, 1449860, 1460971, 1461932, 1564128, 1629746].

References

[1] Redmon J, Farhadi A. Yolo9000: better, faster, stronger. arXiv preprint; 2016. (arXiv; no. 161208242).

[2] Chen TYH, Ravindranath L, Deng S, et al. Glimpse: continuous, real-time object recognition on mobile devices. In: Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems; ACM; 2015. p. 155–168.

[3] Yang K, Cai Z, Zhao L. Algorithm research on moving object detection of surveillance video sequence. Optics and Photonics Journal. 2013;3(2):308.

[4] Szegedy C, Toshev A, Erhan D. Deep neural networks for object detection. In: Advances in neural information processing systems; 2013. Harrahs and Harveys, Lake Tahoe. p. 2553–2561.

[5] Ren S, He K, Girshick R, et al. Faster r-cnn: towards real-time object detection with region proposal networks. In: Advances in neural information processing systems; 2015. Harrahs and Harveys, Lake Tahoe. p. 91–99.

[6] Liu W, Anguelov D, Erhan D, et al. Ssd: single shot multibox detector. In: European conference on computer vision; Springer; 2016. Amsterdam, The Netherlands: Springer, Cham. p. 21–37.

[7] Rudenko A, Reiher P, Popek GJ, et al. Saving portable computer battery power through remote process execution. ACM SIGMOBILE Mobile Computing and Communications Review. 1998;2(1):19–26.

[8] Huerta-Canepa G, Lee D. An adaptable application offloading scheme based on application behavior. In: Advanced information networking and applications-workshops, 2008, AINAW 2008. 22nd International Conference on; IEEE; 2008. GinoWan, Okinawa: IEEE. p. 387–392.

[9] Huang D, Wang P, Niyato D. A dynamic offloading algorithm for mobile computing. IEEE Transactions on Wireless Communications. 2012;11(6):1991–1995.

[10] Barbera MV, Kosta S, Mei A, et al. To offload or not to offload? The bandwidth and energy costs of mobile cloud computing. In: INFOCOM, 2013 Proceedings IEEE; IEEE; 2013. p. 1285–1293.

[11] Wu H, Huang D, Bouzefrane S. Making offloading decisions resistant to network unavailability for mobile cloud collaboration. In: Collaborative computing: networking, applications and worksharing (Collaboratecom), 2013 9th International Conference on; IEEE; 2013. p. 168–177.

[12] Wen Y, Zhang W, Luo H. Energy-optimal mobile application execution: taming resource-poor mobile devices with cloud clones. In: INFOCOM, 2012 Proceedings IEEE; IEEE; 2012. p. 2716–2720.

[13] Xian C, Lu YH, Li Z. Adaptive computation offloading for energy conservation on battery-powered systems. In: Parallel and distributed systems, 2007 International Conference on; Vol. 2; IEEE; 2007. p. 1–8.

[14] Chen X, Jiao L, Li W, et al. Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE/ACM Transactions on Networking. 2016;24(5):2795–2808.

[15] Zhang Y, Liu H, Jiao L, et al. To offload or not to offload: an efficient code partition algorithm for mobile cloud computing. In: Cloud networking (CLOUDNET), 2012 IEEE 1st International Conference on; IEEE; 2012. p. 80–86.

[16] Ran X, Chen H, Liu Z, et al. Delivering deep learning to mobile devices via offloading. In: Proceedings of the Workshop on Virtual Reality and Augmented Reality Network; ACM; 2017. p. 42–47.

[17] Zhang L, Amin SO, Westphal C. Vr video conferencing over named data networks. In: Proceedings of the Workshop on Virtual Reality and Augmented Reality Network; ACM; 2017. p. 7–12.

[18] Chakareski J. Vr/ar immersive communication: caching, edge computing, and transmission trade-offs. In: Proceedings of the Workshop on Virtual Reality and Augmented Reality Network; ACM; 2017. p. 36–41.

[19] Habak K, Ammar M, Harras KA, et al. Femto clouds: leveraging mobile devices to provide cloud service at the edge. In: Cloud computing (CLOUD), 2015 IEEE 8th International Conference on; IEEE; 2015. p. 9–16.

[20] Gao W. Opportunistic peer-to-peer mobile cloud computing at the tactical edge. In: Military communications conference (MILCOM), 2014 IEEE; IEEE; 2014. p. 1614–1620.