Analytics-Aware Storage of Surveillance Videos: Implementation and Optimization

Min-Han Tsai‡, Nalini Venkatasubramanian†, and Cheng-Hsin Hsu‡
‡ National Tsing Hua University, Taiwan † University of California, Irvine

***Abstract*—Increasingly more surveillance cameras in smart environments stream videos to storage servers for on-demand video analytics queries in the future. Unlike on-demand video services, in which maximizing the user-perceived video quality is the design objective, the considered storage servers aim to retain as much information as possible while offering enough space for incoming video clips. In this paper, we design, optimize, and implement an analytics-aware storage server on a smart campus testbed at NTHU, Taiwan, which consists of eight smart street lamps equipped with various sensors, network devices, analytics servers, and a storage server. We focus on the design and implementation of the storage server, and consider two key research problems: (i) how to efficiently determine the information amounts of individual video clips and (ii) how to intelligently downsample individual video clips. More specifically, the first problem is to sample video frames from the stored video clips to analyze for approximations of the information amounts without overloading the storage server. The resulting information amounts are fed into the second problem to decide the video downsampling approaches for retaining as much information amount as possible without consuming excessive storage space. We propose two efficient algorithms to solve these two problems and compare their performance with the current practices via real experiments on our smart campus testbed. Our experiment results reveal the practicality and efficiency of our proposed design and algorithms, e.g., compared to the current practices, our storage server: (i) improves the per-request information amount by up to ~ 4 times, (ii) increases the total information amount by at most $\sim 20\%$, (iii) boosts the number of saved video clips by up to $\sim 35\%$, (iv) runs in real-time, and (v) scales well with larger storage space.**

I. Introduction

More and more cameras in smart spaces enable novel and diverse analytic applications, such as object detection and tracking [1], face recognition [2], health monitoring [3], and traffic management [4]. Typically, the cameras upload surveillance video clips to data centers for storage and analytics [5][6][7]. Doing so however may lead to high operational cost and suffer from network congestion because each surveillance camera produces a traffic stream at several Mbps. A better way to manage the video clips is to store them on a *storage server* in an *edge* network, as illustrated in Fig. 1. The edge network interconnects multiple nearby Internet-of-Things (IoT) devices, including surveillance cameras, and connects to the Internet via a *gateway* through an *access network*. Without uploading all the video clips to the cloud, the traffic load on the access network is reduced. When users need to analyze the surveillance videos, they instruct nearby *analytics servers* to request for corresponding video clips from the storage server. These analytics servers could be stationary serving smartphone or laptop users; they could also be mobile ones installed on, e.g., police cars.

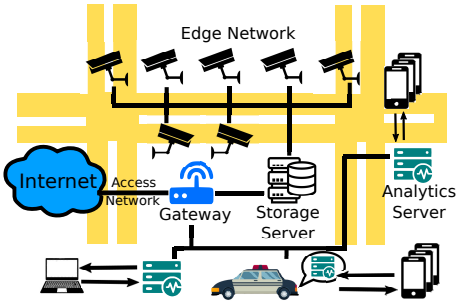


Fig. 1. An illustration of a sample IoT edge network, consisting of cameras (and other sensors), a gateway, a storage server, and several analytics servers.

Keeping the surveillance video clips at the storage server, however, may quickly fill up its disk. For example, storing 2 Mbps video clips from 10 surveillance cameras for merely a week consumes 1.4 TB disk space, but surveillance video clips are typically archived for much longer than a week. Upon the storage space of the edge server is used up, we have to get rid of some video clips to make room for incoming ones. A naive way to do that is to delete the oldest video clips. Doing so, however, may lead to too much information loss, because video clips from different cameras and at different time contain *different* amounts of information. In fact, video clips that contain some information may better be *downsampled* (temporal, spatial, fidelity, or other approaches) instead of being completely deleted. The downsampled video clips can still be analyzed for useful analytics results in the future.

In this paper, we design, implement, and optimize a storage server for saving the surveillance video clips. *The goal is to retain video clips with the highest information amount and selectively downsample the stored video clips to make room for future ones.* This is, however, no easy task for the following reasons:

1) Different video clips contain diverse information amounts, which depend on the dynamic demands of video analytics.
2) Different downsampling approaches lead to diverse amounts of information loss.
3) Quantifying the information amount requires executing video analytics and downsampling video clips requires video transcoding. Both video analytics and downsam-

pling are computationally intensive and thus need to be carefully scheduled.

We address the above three challenges as follows. We first define the information amount to systematically guide the decisions made on our storage server. We consider multiple downsampling approaches for freeing up storage space. We then study two key optimization problems of our analytics-aware storage server. The first problem is to select the sample video frames to analyze, in order to approximate the actual information amount without overloading the storage server. The second problem is to choose the downsample approaches to preserve as much information as possible, while making enough room for incoming video clips. For each of the optimization problems, we discuss its design rationales and propose an efficient algorithm to solve it.

We implement our solution in a real smart campus testbed. Fig. 2 depicts our testbed consisting of eight street lamps close to the EECS building at NTHU, Taiwan. The street lamps, like the one shown in Fig. 2 (left), are equipped with several sensors, including air-quality, temperature, humidity, wind speed, and motion sensors. Four of the street lamps come with IP cameras: three fixed bullet camera and one PTZ (Pan-Tilt-Zoom) camera. The street lamps are interconnected by a mixture of Gigabit Ethernet and WiFi mesh networks. Some of the street lamps hold analytics servers, which can be compact PCs, like Intel NUCs, or single-board computers, like Raspberry Pis. Fig. 2 (right) reveals two sample analytics analyzing the surveillance video clips. We have also deployed a compact PC with an NVIDIA RTX-2060 GPU as the storage server. Using this storage server, we conduct extensive experiments to evaluate our proposed solution. Our experiments reveal that our proposed solution: (i) improves the per-request information amount by up to ~ 4 times, (ii) increases the total information amount by at most $\sim 20\%$, and (iii) boosts the number of saved video clips by up to $\sim 35\%$.

Fig. 2. The smart street lamp testbed at NTHU, Taiwan: (left) a sample street lamp with cases for sensors, network devices, analytics servers, and a storage server; (right) sample video analytics.

II. System Overview

Fig. 3 gives an overview on our storage server in an edge network, which is detailed in this section.

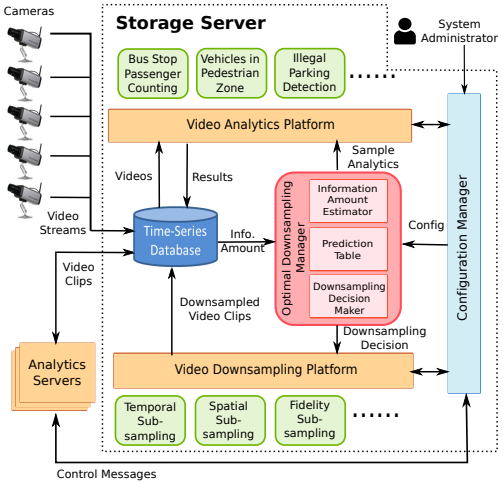


Fig. 3. System overview of our storage server in the edge network.

A. Workflow

The surveillance cameras continuously stream coded videos to the storage server. The video streams are saved on the storage server at the maximal quality, i.e., with the full information amount. The stored video clips can be *requested* by *analytics servers*, which host multiple video analytics. We add a *video analytics platform* to the storage server, which runs video analytics that may be deployed on the analytics servers. We employ an *information amount estimator* to select some surveillance video clips (or frames) for analysis, in order to approximate the information amount without incurring long running time. When the storage server needs to free some space, the video clips (or frames) with the lowest information amount are downsampled. We also add a *video downsampling platform* to the storage server, which hosts various downsampling approaches. We employ a *downsampling decision maker*, which carefully considers the tradeoff among the remaining information amount, freed storage space, and complexity of different downsampling approaches before making the decisions.

Furthermore, we employ *prediction tables* to keep track of available storage and computation resources for timely executions of information amount analysis and video clip downsampling. This is to ensure that enough storage space is freed for new video clips before their arrival time. In summary, the three key components of our storage server are: information amount estimator, downsampling decision maker, and prediction tables. These three components are executed in parallel. We collectively refer to them as *optimal downsampling manager*.

B. Components

We introduce the components of Fig. 3 in the following.

- *Information amount estimator*. This algorithm approximates the information amount of individual video clips (or frames) to address challenge 1 in Sec. I.

- *Downsampling decision maker*. This algorithm controls the information loss due to video clip (or frame) downsampling to address challenge 2.
- *Prediction table*. It supports online predictions of resource consumptions and information amounts models, which are needed by the above two components to ensure ontime completion of the video analytics and downsampling approaches. This is to address challenge 3.
- *Video analytics platform*. It hosts multiple video analytics, which are also run by analytics servers.
- *Video downsampling platform*. It hosts multiple downsampling approaches, which free up storage space.
- *Configuration manager*. It is an interface for system administrators to control the storage server. For example, system administrators may add/remove video analytics and downsampling approaches, or visualize the video clips in the time-series database.

III. Key Research Problems

We formally define the two research problems in this section.

A. Information Amount Estimator

We consider A video analytics. Each analytics a ($a \in [1, A]$) analyzes surveillance video clips to detect certain *events*. We let x_a be the output of analytics a, where x_a can be either a discrete or a continuous value. Examples of discrete outputs include illegal parking (boolean) and queue length at a bus stop (integer); examples of continuous outputs include flood monitoring (depth) and fog detection (visibility). For analytics a with discrete outputs, we let n_a be the *normal* output. That is, if $x_a = n_a$, analytics a detects *no* event. For analytics a with continuous outputs, we define a *tolerance* level δ_a and consider no event is detected if $|x_a - n_a| \leq \delta_a$. Because the outputs of different analytics have diverse scales, x_a needs to be normalized. We first let $\tilde{x}_a$ be the *maximal* absolute value of the output from analytics a[1]. We then define the normalized information amount as:

$$e_a = \begin{cases} 0 & |x_a - n_a| \leq \delta_a; \\ |x_a/\tilde{x}_a| & \text{otherwise}, \end{cases} \quad (1)$$

where δ_a is set to zero for analytics with discrete outputs. Generally, larger e_a values indicate that more information is detected by analytics a.

The outputs of individual video analytics depend on the inputs, which are the surveillance video clips. We consider C video clips stored in a time-series database on the storage server. Each video c ($c \in [1, C]$) has f_c frames in length. Moreover, we let $\mathbf{W}$ be a $C \times A$ matrix, where $\mathbf{W}_{c,a}$ ($c \in [1, C]$ and $a \in [1, A]$) represents the weight of analytics a on clip c, which indicates the importance. $\mathbf{W}$ is configurable by the system administrators, for prioritizing different analytics and clips. With the symbols defined above, we formally write the information amount of all video frames ($\mathbf{F}$) of all video clips ($\mathbf{C}$) with all video analytics ($\mathbf{A}$) as:

$$E(\mathbf{C}, \mathbf{F}, \mathbf{A}) = \sum_{c=1}^{C} \sum_{a=1}^{A} \mathbf{W}_{c,a} e_{c,f_c,a} \cdot f_c \Big/ \sum_{c=1}^{C} \sum_{a=1}^{A} \mathbf{W}_{c,a}. \quad (2)$$

Notice that $e_{c,f_c,a}$ denotes the per-frame information amount across all frames of clip c without any sampling. The summations iterate through all analytics and clips, and are normalized with weight $\mathbf{W}_{c,a}$.

For a given $\mathbf{C}$, $\mathbf{F}$, and $\mathbf{A}$, computing $E(\mathbf{C}, \mathbf{F}, \mathbf{A})$ using Eq. (2) is extremely time consuming as it *dictates* executing all analytics on all frames of all clips. The storage server, unfortunately, is unlikely to be able to perform analysis in real-time, and an *approximation* of $\mathbf{E}(\mathbf{C}, \mathbf{F}, \mathbf{A})$ with a shorter execution time is required. The sample frames need to be carefully chosen, e.g., selecting a few *consecutive* frames may lead to biased results due to the temporal locality across the frames. To cope with this issue, we equally divide each clip c into l video chunks where each chunk contains up to $\lceil \frac{f_c}{l} \rceil$ consecutive frames. We then form a segment of l frames by selecting the middle frame from each chunk[2]. With the concept of segment, we may adjust the l value to trade off the computational complexity (slower with larger l) and information amount accuracy (more accurate with larger l). The definition of per-frame information amount is written as:

$$e_{c,l,a} = \begin{cases} 0 & |x_{c,l,a} - n_{c,l,a}| \leq \delta_a; \\ \left| \frac{x_{c,l,a}/l}{\tilde{x}_a/\tilde{l}} \right| & \text{otherwise}, \end{cases} \quad (3)$$

where $\tilde{l}$ represents the sampling length when we get the maximal absolute value $\tilde{x}_a$. To select the l values, we write a $C \times A$ matrix $\mathbf{L}$, where $\mathbf{L}_{c,a}$ denotes the *sampling* length l of analytics a on clip c. Note that $\mathbf{L}_{c,a} = 0$ implies that analytics a is not applied on clip c at all. The approximated information amount is therefore written as:

$$E'(\mathbf{L}) = \sum_{\mathbf{W}_{c,a} \neq 0} \mathbf{W}_{c,a} e_{c,\mathbf{L}_{c,a},a} \cdot f_c \Big/ \sum_{c \in \mathbf{C}, a \in \mathbf{A}} \mathbf{W}_{c,a}. \quad (4)$$

Computing $E'(\mathbf{L})$ with Eq. (4) is less computationally intensive compared to computing $E(\mathbf{C}, \mathbf{F}, \mathbf{A})$ with Eq. (2). The challenge is to maximize the $E'(\mathbf{L})$ by carefully selecting the best $\mathbf{L}$. *This is the job of our information amount estimator.*

B. Downsampling Decision Maker

We next introduce how we save the storage space via different downsampling approaches. For concrete discussion, we introduce temporal and fidelity approaches, while spatial or others can be readily adopted by our storage server. The first approach is *temporal* subsampling, which keeps the first frames of recurring non-overlapping time windows. For instance, temporal subsampling with a *frame-skip* of 4 means keeping 1 out of every 4 frames (i.e., frames $4k + 1$ $\forall k \in \mathbb{N}$).

[1]Without loss of generality, we assume $\tilde{x}_a \neq 0$. Otherwise, analytics a is not worthy of being executed.

[2]Other sampling approaches, e.g., taking the first or the last frame of each chunk, would also work.

Any future video analytics on *deleted* video frame f are approximated with the immediately preceding frame that is kept, which is $4\lfloor\frac{f-1}{4}\rfloor+1$ in this example. Second, *fidelity* downsampling approach essentially transcodes the video clips with a lower bitrate. The resulting video clips have the same number of video frames, although their analytics outputs may be different from the ones from the original video clips.

We let $\mathbf{P}_0$ be all possible downsampling decisions, where each approach specifies a frame rate and a bitrate. We let $\mathbf{P}$ be a 1 dimensional downsampling decision matrix, with a size of C. $\mathbf{P}_c$ indicates the downsampling approach (e.g., 12 frame-per-second, or fps, at 500 kbps) selected for clip c. $\mathbf{P}_c = 0$ and $\mathbf{P}_c = -1$ indicate deleting clip c and keeping c as it is, respectively. The approximation of information amount after taking downsampling decision $\mathbf{P}$ is written as:

$$E'(\mathbf{P}) = \sum_{\mathbf{P}_c \neq 0} \left(\sum_{a=1}^{A} \mathbf{W}_{c,a} \hat{e}(c, a, \mathbf{P}_c) \cdot f_c \Big/ \sum_{a=1}^{A} \mathbf{W}_{c,a} \right). \tag{5}$$

We assume that each video clip can be downsampled more than once with different decisions. The challenge is to maximize $E'(\mathbf{P})$ by carefully selecting the best $\mathbf{P}$. *This is the job of our downsampling decision maker.*

IV. Optimal Downsampling Manager

In this section, we present our design of optimal downsampling manager. We start from the prediction tables, which are followed by the two algorithms solving the research problems described above.

A. Prediction Tables

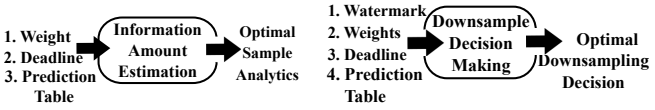
The resource consumption of video analytics and downsampling approaches depends on: (i) analytics/downsampling types, (ii) analytics/downsampling parameters (i.e., $\mathbf{L}$ and $\mathbf{P}$), and (iii) *context information*. Context information refers to external information that serves as *hint* for more accurate predictions. Examples of context information include weather (fewer people taking buses in rainy days), day-of-the-week (campus buses are busier on weekdays), time-of-the-day (fewer pedestrians crossing an intersection in late evenings), and parking lot occupancy (illegal parking are more likely when the lots are more full). We let function $t(c, a, \mathbf{L}_{c,a})$ be the per-frame execution time when analyzing sampling frames with length $\mathbf{L}_{c,a}$ of video clip c using analytics a. The context information of c is stored in the same time-series database along with c itself. Similarly, we use $\hat{t}(c, \mathbf{P}_c)$ to represent the downsampling time of approach d with decision $\mathbf{P}_c$ on video clip c. In addition, the storage space saved by downsampling can be quantified using the *compression rate* $r(c, \mathbf{P}_c) = \hat{o}_{c,\mathbf{P}_c}/o_c$, where $\hat{o}_{c,\mathbf{P}_c}$ represents the compressed video clip size and o_c is the size of clip c in the database.

There are several ways of predicting the resource consumptions under different parameters and context. One possibility is to adopt *general* regression models with *arbitrarily chosen* parameters under *diverse* context. Doing so, however, requires too many samples to train the regression models, which is fairly expensive. We argue that the parameters need *not* be arbitrarily chosen because they are determined by the administrators of the storage server. In other words, as long as our predictions are accurate with a few *pre-selected* parameters values, the storage server will work as good as, if not better than, having general regression models. Hence, we propose to employ lookup tables indexed by analytics/downsampling decisions and context. The tables are built and updated online with a *sliding window* of λ samples to accommodate the environmental dynamics. Whenever there is a ground-truth sample coming from the video downsampling platform or analytics servers, we update the sliding window and the tables. When a prediction is needed, the values in the lookup tables are returned. When a cell is not populated (is empty), we use the value in the closest cell (in the sense of context) for prediction.

TABLE I
Sample Look-up values of Prediction Tables

Index	Sample Values
Analytics	{People counting, Illegal parking, ...}
Downsampling Decision	{(12 fps, 500 kbps), (6 fps, 125 kbps), ...}
Day-of-the-week	{Weekday, Weekend}
Time-of-the-day	{0, 2, ..., 22}

We also need to predict the per-frame information amount $e_{c,L_{c,a},a}$. Following the same design rationale of the resource prediction, we build an information amount table $\hat{e}(\cdot)$ indexed by analytics, downsampling decision, and context. We give a sample lookup table in Table I. We estimate $e_{c,L_{c,a},a}$ by $\hat{e}(\cdot)$ and update $\hat{e}(\cdot)$ with the moving averages, which in turns allows us to derive $E'(\mathbf{L})$, and $E'(\mathbf{P})$ using Eqs. (4) and (5), respectively.



(a) (b)

Fig. 4. Inputs/outputs of: (a) the information amount estimation and (b) the downsample decision making problems.

B. Information Amount Estimator

Design Rationale. Given that the computing power is limited, we aim to invest more computing power on the clips and analytics with higher information amounts, so that we have more accurate estimation on them. We consider a storage server with storage space O. The deadline for completing the sample analytics is δ_i mins, while the problem is solved once every δ_i' hours. Both δ_i and δ_i' are configurable parameters. The goal is to find the sample analytics matrix $\mathbf{L}^*$, so that: (i) the analytics can be completed by δ_i and (ii) the estimated information amount $E'(\mathbf{L}^*)$ is as close to $E(\mathbf{C}, \mathbf{F}, \mathbf{A})$ as possible. Fig. 4(a) summarizes the inputs of the problem, which are: (i) weights $\mathbf{W}$, (ii) deadline δ_i, and (iii) prediction tables. The output is the optimal analytics samples $\mathbf{L}^*$.

Algorithm 1 Greedy Estimation (GE) Algorithm

Inputs: Weight $\mathbf{W}$, Deadline δ_i, Sampling Lengths $\mathbf{L}_0$, and Prediction Tables $\hat{e}(\cdot)$.
Output: Optimal Sampling Matrix $\mathbf{L}^*$.

1: Let $\mathbf{L}_{c,a} = max(\mathbf{L}_0), \forall c \in [1, C], a \in [1, A]$
2: **while** $\sum_{\forall c,a} t(c, a, \mathbf{L}_{c,a}) \cdot \mathbf{L}_{c,a} > \delta_i$ **do**
3: Find $(c, a) = \underset{(c,a)}{\text{argmin}} \frac{\mathbf{W}_{c,a}\hat{e}(c,a,\mathbf{L}_{c,a})\cdot f_c}{t(c,a,\mathbf{L}_{c,a})}$, where $\mathbf{L}_{c,a} \neq \min(\mathbf{L}_0)$
4: Let $\mathbf{L}_{c,a}$ be the next smaller length in $\mathbf{L}_0$
5: **return** $\mathbf{L}$ as $\mathbf{L}^*$

Greedy Estimation (GE) Algorithm. We propose an algorithm called Greedy Estimation (GE). This algorithm is based on an intuition: *the execution time and accuracy of information amount are both reduced once the sampling length is shorter*. Algorithm 1 gives the pseudocode of the GE algorithm. In line 1, we initialize the sampling lengths $\mathbf{L}_{c,a}$ of all clip c and analytics a to be the largest value in $\mathbf{L}_0$. If doing so can fit into deadline δ_i, we take that $\mathbf{L}$ as $\mathbf{L}^*$, because it gives the most accurate information amount. The while-loop starts from line 2 checks if the total execution time of $\mathbf{L}$ exceeds the deadline. If yes, line 3 chooses the (c, a) pair that contains the least information amount, among all pairs whose sampling length can still be decreased. We decrease the sampling length of (c, a) in line 4 because it doesn't contain too much information anyway. Upon $\mathbf{L}$ meets δ_i, we return $\mathbf{L}$ as the optimal sampling matrix $\mathbf{L}^*$ in line 5. It is not hard to see that GE has a time complexity of $\mathcal{O}(\delta_i)$, and a space complexity of $\mathcal{O}(CA)$.

C. Downsampling Decision Making Problem

Design Rationale. Given that the storage space is limited, we aim to downsample the video clip with the smallest per-unit-size information amount until we free up enough storage space. We set watermarks to indicate determine when to make decisions. The downsampling decision making problem is solved once the used space of the storage server reaches a high watermark $O_{v'}$. The video downsampling must be done by a deadline δ_d, and the resulting used storage space should be lower than a low watermark O_v. $O_{v'}$, O_v, and δ_d are user-configurable. Fig. 4(b) summarizes the inputs of the problem, which are: (i) low watermark O_v, (ii) weights $\mathbf{W}$, (iii) deadline δ_d, and (iv) prediction tables. The output is the optimal downsampling decision $\mathbf{P}^*$.

Greedy Decision (GD) Algorithm. We propose an algorithm called Greedy Decision (GD). The algorithm is based on an intuition: *the video clip with the smallest per-unit-size information amount should be scarified first, while the degree of its downsampling approach should be kept as small as possible.* Algorithm 2 gives the pseudocode of the GD algorithm. Line 1 initializes the downsampling decisions of all clips c to be -1, which means keeping clip c as is. The same line also initializes S and T as the used storage space and total execution time, respectively. The while-loop starting from line 2 iterates as long as the used storage space is higher than the low watermark *or* the total execution time does not exceed

Algorithm 2 Greedy Decision (GD) Algorithm

Inputs: Weight $\mathbf{W}$, Deadline δ_d, Watermark O_v, and Prediction Tables $\hat{e}(\cdot)$.
Output: Optimal Downsampling Matrix P^*.

1: Let $\mathbf{P}_c = -1, \forall c \in C$; $S = \sum_{\forall c \in C} o_c$; $T = 0$;
2: **while** $S > O_v$ **or** $T > \delta_d$ **do**
3: $c = \underset{\forall c \in C, \mathbf{P}_c \neq 0}{\text{argmin}} \frac{\sum_{a=1}^{A} \mathbf{W}_{c,a}\hat{e}(c,a,\mathbf{L}^*_{c,a})\cdot f_c}{\hat{o}_{c,\mathbf{P}_c}\cdot\sum_{a=1}^{A}\mathbf{W}_{c,a}}$
4: $d = \underset{\forall \hat{o}_{c,\mathbf{P}_c} \geq \hat{o}_{c,d}}{\text{argmin}} (\hat{o}_{c,\mathbf{P}_c} - \hat{o}_{c,d})$
5: $S = S - \hat{o}_{c,\mathbf{P}_c} + \hat{o}_{c,d}$;
6: $T = T - \hat{t}(c, \mathbf{P}_c) + \hat{t}(c, d)$;
7: $\mathbf{P}_c = d$
8: **return** $\mathbf{P}$ as $\mathbf{P}^*$

the deadline. The numerator in line 3 accounts for the overall information amount of clip c, and the denominator accounts for the overall size. That is, line 3 selects clip c with the smallest per-unit-size information amount to be downsampled. Line 4 chooses the decision d with the smallest downsampling step, and hope applying d on c is sufficient to bring the used storage space to the low watermark O_v. Lines 5–7 apply the changes on $\mathbf{P}$, S, and T, before going back to line 2. We note that line 4 may end up with $d = 0$, which indicates that clip c is going to be deleted. Upon the low watermark and deadline are met, line 8 returns $\mathbf{P}$ as the optimal decision matrix $\mathbf{P}^*$. It is not hard to see that the time complexity of the GD algorithm is $\mathcal{O}(O_v + \delta_d)$, while its space complexity is $\mathcal{O}(C)$.

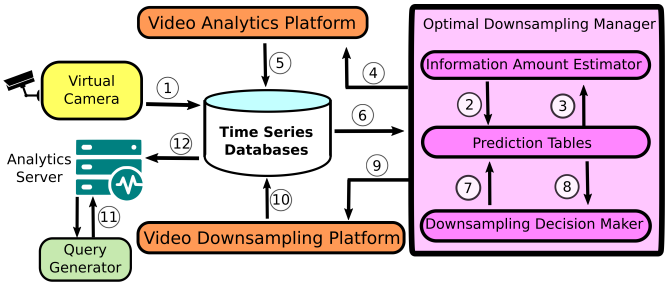


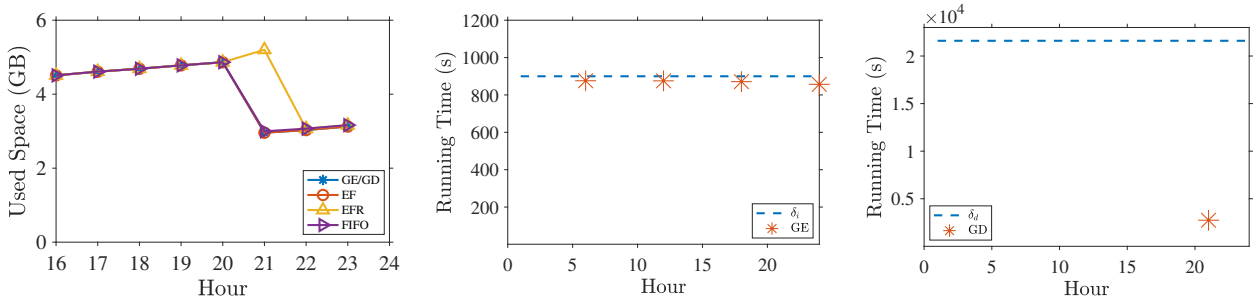
Fig. 5. Testbed consisting of our implemented storage server, an analytics server, a virtual camera, and a query generator. The circled numbers indicate the order of the operations.

V. Evaluations

We conduct real experiments to evaluate the performance of our storage server.

A. Implementations

We have implemented the proposed storage server on our smart street lamp testbed at NTHU, Taiwan, which is shown in Fig. 2. Our testbed implementation is summarized in Fig. 5, which complies with the design in Fig. 3. Most implemented components belong to the storage server, which is written in Python. We leverage: (i) InfluxDB [8] to realize the time-series database, (ii) Darknet [9] to realize the video analytics platform and analytics servers, and (iii) FFmpeg [10] to realize the video downsampling platform. We implement the



(a) (b) (c)

Fig. 6. Our storage server works effectively: (a) the used storage space, (b) the running time of the sample video analytics, and (c) the running time of the video downsampling decision. Sample results from day 2 (weekend) are shown.

optimal downsampling manager in a *modularized* manner, where different components communicate using the socket API. This allows us to readily replace various algorithms in our storage server for comparisons. While our storage server *can*: (i) receive surveillance video streams from the four IP cameras on our smart street lamps and (ii) serve queries through analytics servers from actual users, doing so results in unpredictable *workload*, which may prevent us from fairly comparing different storage server designs. To cope with this limitation, we also implement: (i) a *virtual camera*, which emulates a camera in our testbed by replaying the surveillance videos and (ii) a *query generator*, which emulates the requests from analytic servers. The two components allow us to impose *exactly* the same workload on different storage server designs.

B. Setup

For comparisons, we have also implemented the following storage server designs, which are the current practices:

- **Equal-Fidelity (EF)** downsamples the old video clips to a pre-defined bitrate (100 kbps if not otherwise specified) when more storage space is needed. EF only downsamples each video clip once, i.e., a previously downsampled video clip is deleted when EF is invoked again.
- **Equal-Frame-Rate (EFR)** is similar to EF, except that it downsamples the old video clips to a pre-defined frame rate (6 fps if not otherwise specified) instead of bitrate.
- **First-In-First-Out (FIFO)** always deletes the oldest video clips to free disk space for incoming video clips.

The considered system parameters are as follows, where the bold font (if any) indicates the default value.

- **Sampling lengths**: $\mathbf{L}_0 \in \{5, 10, 25, 50, 100\}$.
- **Downsampling decisions**: $\mathbf{P}_0 \in \{(\text{fr}, \text{br})$, where $\text{fr} \in \{24, 12, 6, 1\}$ fps and $\text{br} \in \{1000, 500, 100, 10\}$ kbps$\}$.
- **Storage space** $O \in \{\mathbf{5}, 20, 100\}$ GB.

Moreover the high/low watermarks are set to be 100% and 60% of the storage space in our experiments. We let $\delta_i = 15$ mins, $\delta_i' = 6$ hrs, and $\delta_d = 6$ hrs. We report the sample results from unit weight matrix due to space limitations. We consider two sample analytics (which are illegal parking and people counting) and let $\lambda = 12$ throughout the experiments. During the peak hours, the illegal parking events occur at 60% of the time, while the average people count is 5.91. We consider the following performance metrics: (i) information amount, (ii) used storage space, (iii) number of the stored video clips, and (iv) running time of the algorithms.

To drive our virtual camera, we select video clips recorded in November 2019 from an IP camera facing a major intersection. The video clips are encoded in HEVC [11] at 1 Mbps in 2048×1536 resolution. Each video clip lasts for 1 min at 24 fps. For each experiment, we run the storage server for five days using the video clips starting from the 9-th (Saturday). After replaying the video clips of the first five days, we generate random queries on the day 6, where the queried video clips uniformly span over the 1st–5th days. For each of the two considered analytics applications, we employ a Poisson process to generate random queries, so that there are 10 queries per hour on average. We repeat the same experiments with our proposed GE/GD algorithms, as well as FIFO, EF, and EFR storage server designs.

C. Results

Our storage server and GE/GD algorithms are effective. Fig. 6 plots the sample results on used storage space and the running time of sample video analytics and downsampling decision from day 2. Results from other days are similar. Fig. 6(a) confirms that: (i) the downsampling decision maker is invoked once the used storage space reaches the high watermark and (ii) the used storage space drops to the low watermark as designed. Figs. 6(b) and 6(c) report the total running time of the video analytics and downsampling approaches. These two figures demonstrate that both deadlines (δ_i and δ_d) are met in real experiments. In fact, the downsampling approaches are done with merely 9% of the deadline. Fig. 6 shows the effectiveness of our design and implementations.

Our GE/GD algorithms preserve more information amount. Fig. 7 plots the sample results on the preserved information amount from days 2 and 3. Results from other days are similar. Figs. 7(a) and 7(d) give the per-query information amount. We first observe that EFR and FIFO lead to zero information amount for almost all queries (some samples are overlapping and harder to see). This is because the queries are

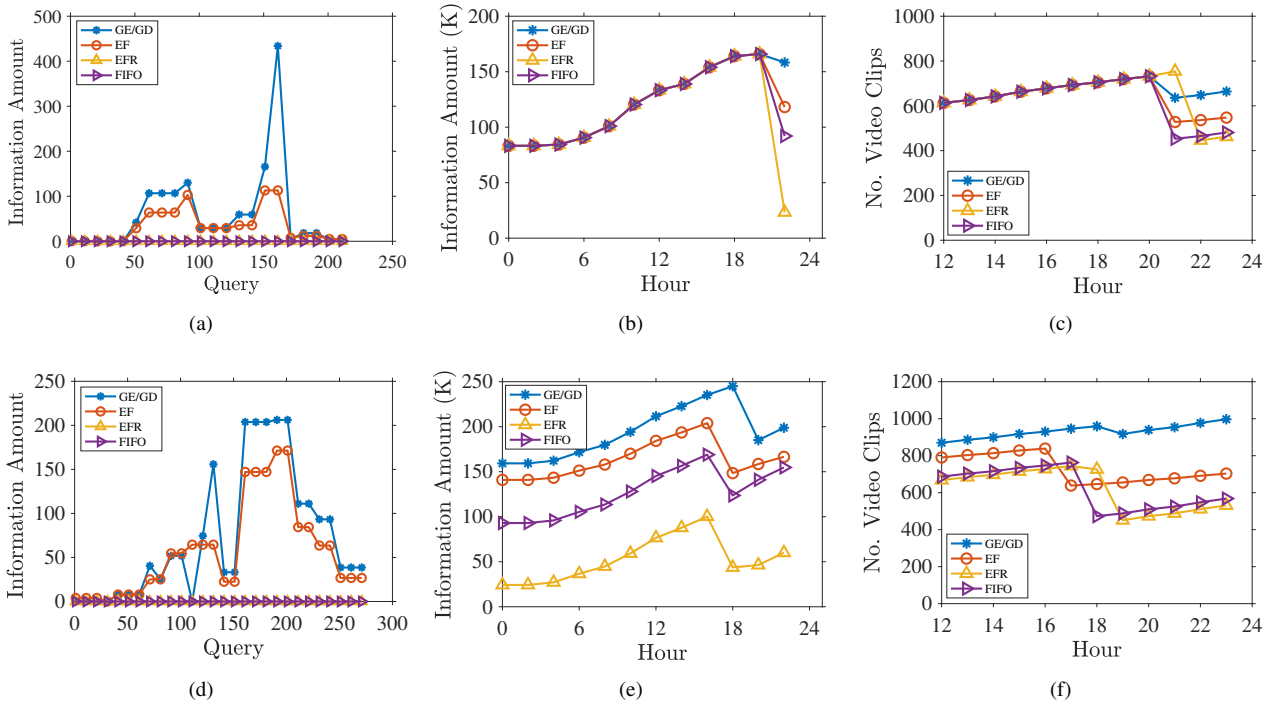


Fig. 7. Our algorithms preserve more information amounts: (a), (d) per-query information amount; (b), (e) total information amount of stored video clips; (c), (f) number of stored video clips. Sample results from day 2 (weekend): (a), (b), (c); sample results from day 3 (weekday): (d), (e), (f).

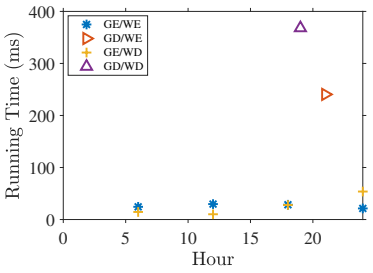


Fig. 8. Running time of GE and GD algorithms. Sample results from day 2 (WE) and day 3 (WD).

generated on day 6; at that time, video clips of earlier days are already removed. EF works slightly better than EFR and FIFO. Our GE/GD algorithms significantly outperforms EF by up to ~ 4 times on days 2 (weekend) and ~ 3 times on day 3 (weekday). Figs. 7(b) and 7(e) show the total information amount of stored video clips. We make two observations on these two figures. First, the four lines overlap with one another in Fig. 7(b) until about 20:00. This is because the used storage space has not reached the high watermark. Second, after video downsampling is needed, our GE/GD algorithms significantly outperform all other designs. Compared to the runner-up, we outperform EF by at most $\sim 30\%$ on day 2 and $\sim 20\%$ on day 3. Figs. 7(c) and 7(f) depict the number of stored video clips. These figures show that our GE/GD algorithms save at most $\sim 13\%$ more clips on day 2 and $\sim 35\%$ more clips on day 3, compared to EF. Fig. 7 shows that our GE/GD algorithms indeed preserve more information amount compared to the current practices.

Our GE/GD algorithms run in real-time. Fig. 8 reports the sample running time of our algorithms from days 2 and 3. Results from other days are similar. The figure confirms that our algorithms run in real-time, at most 53 ms for GE and 368 ms for GD. The running time is indeed negligible compared to the deadlines, which are in the order of mins if not hrs.

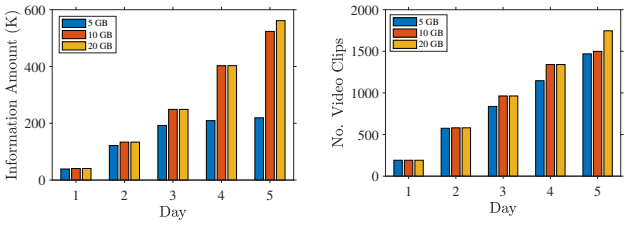


Fig. 9. Our GE/GD algorithms scale well with larger storage spaces: (a) total information amount of stored video clips and (b) number of stored video clips.

Our GE/GD algorithms scale well with larger storage spaces. Fig. 9 reports the per-day performance of our GE/GD algorithms under different the storage spaces. We observe that GE/GD perform well across the five days. Moreover, GE/GD also perform well with larger storage spaces: both in terms of total information amount and number of video clips.

VI. Related Work

Video analytics applications are considered as the *killer app* of edge computing [12], which have been studied in the literature. For example, Liu et al. [13] present EdgeEye, which is an edge computing framework for real-time video analytics applications. *Such studies are however orthogonal to our proposed storage server, as they did not consider the information amount.* Video downsampling is usually achieved by transcoding, which has been studied in the literature. For example, Li et al. [14] propose to transcode videos in an on-demand manner to reduce the cloud resource consumptions. *Such studies are also orthogonal to our proposed storage server, as they did not consider the information loss.* There have been a few recent studies on the designs of video storage servers in smart environments, which are closer to our work. For example, Shao et al. [15] study the correlation among the cameras at different locations to detect the abnormal behaviors, which is achieved by building a risk table. Each clip is then determined to be deleted, partially deleted, or kept. Unfortunately, the strategy of making such decisions are not detailed in their paper. Usman et al. [16] propose an intrusion-driven model, which encodes videos clips with different encoding parameters. They assume the video clips can only be downsampled once. *Different from our work, these two studies [15,16] did not propose systematic approaches to: (i) quantify the information amounts, nor (ii) decide the downsampling approaches and parameters.*

VII. Conclusion

In this paper, we detailed the design, optimization, and implementation of an analytic-aware storage server for surveillance videos from smart environments. The design goal of the storage server is to retain as much information amounts as possible under the constraints on storage space and computational power. We achieved the design goal in three steps. We first carefully defined the information amount and proposed an efficient table design for predicting resource consumptions and information amount. The prediction is then capitalized to solve two key research problems: (i) information amount estimation, which computes a sampling length matrix to approximate the information amounts without overloading the storage server and (ii) downsampling decision maker, which selects a downsampling decision matrix to retain the most information amount without consuming excessive resources (both computation and storage). We conducted extensive experiments to compare the performance of our proposed algorithms and system against the current practices. The experiment results demonstrate the merits of our solution, which:

1) improves the per-request information amount by up to ~ 4 times;
2) increases the total information amount by at most $\sim 20\%$;
3) boosts the number of saved video clips by up to $\sim 35\%$.

Our proposed storage server can be extended in several directions. First, more comprehensive information prediction, such as those built upon Reinforcement-Learning (RL) approaches may be adopted; and the implication of more accurate predictions on the performance storage server can be quantified. Second, a wider array of analytics applications may be considered, so that the information overlap among them can be investigated and potentially leveraged in the storage server design. Last, approximation algorithms can be developed to solve the two research problems with performance guarantees

References

[1] N. Funde, P. Paranjape, K. Ram, P. Magde, and M. Dhabu, "Object detection and tracking approaches for video surveillance over camera network," in *Proc. of IEEE International Conference on Advanced Computing & Communication Systems (ICACCS)*, Coimbatore, India, 2019, pp. 1171–1176.

[2] P. Li, L. Prieto, D. Mery, and P. Flynn, "On low-resolution face recognition in the wild: Comparisons and new techniques," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 8, pp. 2000–2012, 2019.

[3] A. Prati, C. Shan, and K. Wang, "Sensors, vision and networks: From video surveillance to activity recognition and health monitoring," *Journal of Ambient Intelligence and Smart Environments*, vol. 11, no. 1, pp. 5–22, 2019.

[4] F. Mehboob and M. Abbas, "Intelligent video surveillance," in *Video Surveillance-Based Intelligent Traffic Management in Smart Cities*, A. Neves, Ed. IntechOpen, November 2018, ch. 2, pp. 1–9.

[5] S. Dey, A. Chakraborty, S. Naskar, and P. Misra, "Smart city surveillance: Leveraging benefits of cloud data stores," in *Proc. of IEEE Conference on Local Computer Networks Workshops (LCNW)*, Clearwater, Florida, 2012, pp. 868–876.

[6] D. Rodríguez-Silva, L. Adkinson-Orellana, F. Gonz'lez-Castano, I. Armino-Franco, and D. Gonz'lez-Martinez, "Video surveillance based on cloud storage," in *Proc. of IEEE International Conference on Cloud Computing (CLOUD)*, Honolulu, Hawaii, 2012, pp. 991–992.

[7] S. Hossain, M. Hassan, M. Al, and A. Alghamdi, "Resource allocation for service composition in cloud-based video surveillance platform," in *Proc. of IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, Melbourne, Australia, 2012, pp. 408–412.

[8] InfluxDB. (2019) InfluxDB official website. [Online]. Available: https://www.influxdata.com/

[9] Alexey. (2019) YOLO-v3 and YOLO-v2 for Windows and Linux. [Online]. Available: https://github.com/AlexeyAB/darknet

[10] FFmpeg Developers. (2019) FFmpeg documentation. [Online]. Available: http://ffmpeg.org/

[11] G. Sullivan, J. Ohm, W. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[12] G. Ananthanarayanan, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *Computer*, vol. 50, no. 10, pp. 58–67, 2017.

[13] P. Liu, B. Qi, and S. Banerjee, "Edgeeye: An edge service framework for real-time intelligent video analytics," in *Proc. of International Workshop on Edge Systems, Analytics and Networking (EdgeSys)*, Munich, Germany, 2018, pp. 1–6.

[14] X. Li, M. Salehi, and M. Bayoumi, "High performance on-demand video transcoding using cloud services," in *Proc. of IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, Cartagena, Colombia, 2016, pp. 600–603.

[15] Z. Shao, J. Cai, and Z. Wang, "Smart monitoring cameras driven intelligent processing to big surveillance video data," *IEEE Transactions on Big Data*, vol. 4, no. 1, pp. 105–116, 2017.

[16] A. Usman, R. Usman, and S. Shin, "An intrusion oriented heuristic for efficient resource management in end-to-end wireless video surveillance systems," in *Proc. of IEEE Annual Consumer Communications & Networking Conference (CCNC)*, Las Vegas, Nevada, 2018, pp. 1–6.