VideoEdge: Processing Camera Streams using Hierarchical Clusters

Chien-Chun Hung
Microsoft Research / University of Southern California

Ganesh Ananthanarayanan
Microsoft Research

Peter Bodik
Microsoft Research

Leana Golubchik
University of Southern California

Minlan Yu
Harvard University

Paramvir Bahl
Microsoft Research

Matthai Philipose
Microsoft Research

ABSTRACT

Organizations deploy a hierarchy of clusters – cameras, private clusters, public clouds – for analyzing *live* video feeds from their cameras. Video analytics queries have many implementation options which impact their resource demands and accuracy of outputs. Our objective is to select the "query plan" – implementations (and their knobs) – and place it across the hierarchy of clusters, and merge common components across queries to maximize the average query accuracy. This is a challenging task, because we have to consider multi-resource (network and compute) demands and constraints in the hierarchical cluster and search in an exponentially large search space for plans, placements, and merging. We propose VideoEdge, a system that introduces *dominant demand* to identify the best tradeoff between multiple resources and accuracy, and narrows the search space by identifying a "Pareto band" of promising configurations. VideoEdge also balances the resource benefits and accuracy penalty of merging queries. Deployment results show that VideoEdge improves accuracy by $25.4\times$ and $5.4\times$ compared to fair allocation of resources and a recent solution for video query planning (VideoStorm [69]), respectively, and is within 6% of optimum.

I. Introduction

Major cities like New York and Beijing are deploying thousands of cameras. Analyzing *live video streams* from these cameras is of considerable importance to organizations. Traffic departments analyze video streams from cameras at intersections for traffic control, and police departments analyze city-wide cameras for surveillance.

Organizations commonly deploy a *hierarchy of clusters* to analyze their video streams. Every organization, e.g., a traffic department, runs its *private* cluster to pull in the video feeds from its cameras (with dedicated bandwidths). The private cluster contains compute for analytics while also tapping into public clouds (like Amazon EC2) for overflow compute. The uplink bandwidth between the private cluster and the public cloud, however, is not sufficient to stream all the camera feeds to the cloud for analytics, and the available network capacity could vary [1]. Some newer cameras also have compute capacity on them. Major video analytics providers like Genetec [8] and Avigilon [4], market studies [15], as well as cloud providers like Amazon and Microsoft [13] indicate that the hierarchical architecture – camera, private cluster, cloud – is indeed common and *the only feasible approach* for large scale analytics of video streams. The suitability of a hierarchical architecture is also reflected in our partnership with the City of Bellevue, WA, and City of Washington, DC that are deploying our traffic video analytics solution [42].

Video analytics *queries* are a pipeline of computer vision *components*. For example, the object tracking query consists of a "decoder" component, followed by an object "detector", and an "associator" component. Each component has many *implementation* choices that provide the same abstraction. For example, object detectors take a frame and output a list of detected objects. Detectors can use background subtraction to identify moving objects against a static background or a deep neural network (DNN) to detect objects based on visual features. Components and their implementations are analogical to logical and physical operators in SQL queries.

Different implementations have different *resource demands* and produce outputs of varying *accuracies*. To detect objects, background subtraction requires fewer resources than a DNN but is also less accurate because it misses stationary objects. Core vision components have tens of different implementations but no single one that is cheapest and most accurate across all scenarios. Components can also have many *knobs* (like frame resolution or frame rate) to set. Higher resolution or frame rate leads to higher accuracy due to more information provided, but requires more resources to process, which further impacts resource-accuracy trade-off. Video queries have thousands of combinations of implementations and knob values that impact their "resource-accuracy" relationship. We define *query planning* as selecting the best combination of implementations and knob values for a query.

While the accuracy of a query depends only on its plan, its network and CPU resource demands are determined by component *placement* across the hierarchy of clusters. For example, placing the tracker's detector on camera and associator in the private cluster uses compute and network of the camera and private cluster, but not the uplink to the cloud and CPU in the cloud. However, only some of the placements are feasible, depending on the available resource capacities.

Finally, multiple queries analyzing video *from the same camera* often have common components – e.g., queries to count cars and monitor pedestrians both need an object detector and associator [42]. The common components are typically the core vision building blocks, and such commonality is especially prevalent among video queries because extracting visual features of objects is central to all video analytics. *Merging* these common components among queries by running single instance of the common components and sharing it allows for significant savings in resources, but would also constrain them to the same choice of plan and placement, among the queries.

Objective: Our objective is to build a *video query optimizer* that takes as input: a) pipeline of components along with the different implementation options and knobs, b) cluster hierarchy with resource capacities, and c) representative video that can be used to estimate component cost and accuracy. The optimizer then automatically, (i) determines the *query plan* for each video query, (ii) *places its components* across the hierarchy of clusters, and (iii) *merges common components* across queries, to *maximize the average query accuracy*.

Although query optimization has been extensively studied in traditional database community [36], [37], [52], [54], they cannot be directly applied to optimizing for video queries for the following reasons: (i) cost of video processing components cannot be easily estimated as it depends both on their configuration and video content, (ii) both accuracy and resources are important in video query optimization, and (iii) while merging is similar to multi-query optimization, it does not jointly solve merging with placement and accuracy of the queries.

Recent work on video processing, VideoStorm [69], selects video query knobs to maximize accuracy, but ignores three important issues that we tackle. First, VideoStorm assumes that all the videos are streamed into a single cluster and hence ignores component placement in hierarchical settings. As a result, it does not deal with multiple resources (network and compute) or dynamic network bandwidths. Second, it does not identify the opportunity to merge common components across queries. Finally, it does not consider implementation choices of vision components (only knobs).

Achieving high accuracy when optimizing video analytics queries requires addressing two challenges: 1) exponentially large search space, and 2) conflicts in merging that save resources but also lowers accuracy.

Challenge 1: exponentially large search space. To maximize accuracy, we need to *jointly* plan, place, and merge across all the queries. If we separately determine a query's plan, we might not have enough resources to place its components. Even if we plan and place queries together, we might not be able to merge common components because they may end up with different plans or placements. However, the joint optimization leads to an *exponentially large search space*.

Our solution, VideoEdge, efficiently navigates the search space of query planning, placement, and merging using an efficient heuristic. We identify the most promising "configurations" – combinations of a query plan and placement – and filter out those that have low accuracy *and* large resource demands. We call the promising configurations the *Pareto band* of configurations since we extend the classic economic concept of Pareto efficiency [61]. The spread in accuracy and resource demands of configurations of video analytics queries allows for the Pareto band to drastically reduce the search space. Each configuration has specific resource demands across different resources and clusters, making it non-trivial to compare configurations. For each configuration, we define its *dominant demand*: the maximum ratio of demand to capacity across all resources and clusters in the hierarchy. This allows us to directly compare configurations across demand and accuracy and avoids lopsided drain of any single resource. Our heuristic searches through the configurations within the Pareto band and greedily switches to configurations that increase accuracy with little increase in dominant demand.

Challenge 2: merging conflicts. When merging common query components, we have to consider potential *merging conflicts*. While merging two components reduces resource usage, the implementation choice and placement of the merged component might not be the best for both queries, which lowers the accuracy. For example, DNN-based detector is better for pedestrian monitoring while background subtractor is better for car counting. We resolve merging conflicts by carefully considering the change in aggregate accuracy against reduction in resource usage of different merging options.

VideoEdge includes an efficient profiler that generates the resource-accuracy profile by using $100\times$ fewer CPU cycles than an exhaustive exploration. VideoEdge accommodates user-specified cost budgets (§V-E) and is robust in handling dynamic bandwidths over time (§VI-A). We compare VideoEdge with VideoStorm [69], and *fair sharing* of resources among queries [19], [34] that is commonly employed by production systems [3], [25], [68]. Evaluation using realistic video queries show that we outperform fair sharing and VideoStorm by $25.4\times$ and $5.4\times$ respectively in accuracy and are within 6% of optimum.

II. Video Analytics Clusters & Queries

We describe hierarchical clusters for video processing in §II-A and the query plans to run various implementations of video processing in §II-B. Finally, we take *object tracker* as an example to show the diverse trade-offs between CPU and network resources, and accuracy in §II-C.

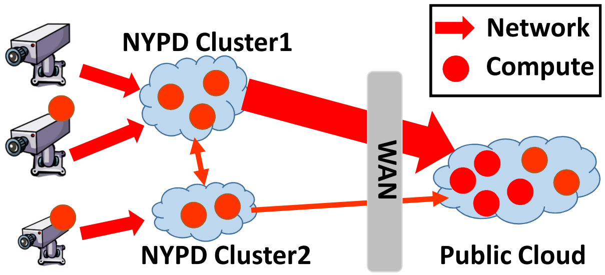


Figure 1: Hierarchical Video Analytics Architecture. Network links between cameras, private clusters, and the public cloud have diverse bandwidths (represented by the width of the arrow). The compute at the locations also vary.

A. *Hierarchical Clusters*

Organizations with large deployment of cameras – e.g., cities, police departments, or agriculture farms – typically use a *hierarchy of clusters* (or *locations*, interchangeably) to process video streams [4], [8], [14], [62]. Figure 1 shows that each organization (e.g., NYPD) runs private clusters that pull video from their cameras.

Compute hierarchy: Compute capacities at the private clusters vary significantly from just a handful of cores (as in a small farm) to hundreds of cores (as in New York City [14]), and can include GPUs or other hardware DNN accelerators. Newer cameras themselves contain compute capacity [5] for video analytics and organizations may also tap into the public cloud like Amazon EC2 and Microsoft Azure for compute.

Network: Connectivity between the cameras and private clusters (via cellular, wireless [70] or fiber links) is a crucial resource. The bandwidth required to support a single camera stream ranges from a few hundred Kb/s to many Mb/s for multi-megapixel cameras. We can control the bitrate of the video stream by configuring the resolution and frame rate directly on the camera. Typically, the uplink from private clusters to the public cloud is a few tens of Mb/s [1] and supports streaming only a small fraction of camera streams to the cloud (per our conversations with Avigilon [4] and Genetec [8], leaders in video analytics solutions). In rural deployments, such as cameras for monitoring crops in farms [62], uplink Internet connectivity is even more restricted and expensive.

Trends of increased compute availability on the “edge” (on cameras and private clusters) and bandwidth scarcity in reaching the cloud [53] has made the “intelligent edge” model a strong focus for many large industry players like Microsoft and Amazon in their IoT product offerings [13]. Because of high bandwidth requirements, utilizing the compute on the hierarchy of edges is the only feasible approach for processing live videos at scale [15]. Our discussions with many traffic jurisdictions in the USA that we partner with, also indicate their keenness on utilizing compute on their cameras to reduce the cloud expenditure.

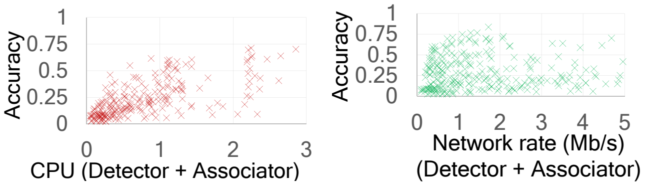
B. *Implementations for Vision Primitives*

Video processing often involves core vision primitives – object *detection*, objects *association* across frames, and *recognition* of object class – each with many *implementation choices*. A common approach to object *detection* is to extract moving objects using background subtraction or using DNNs such as YOLO [51]. There are over 40 different algorithms in the background subtraction library [7]. To *associate* objects across frames, one can use different metrics such as color histograms, or the SIFT [11] or SURF [12] features. The VOT 2016 object tracking challenge has 70 different associators [16]. The ImageNet object recognition challenge [10] has up to 80 different recognizers. Finally, for the primitives with DNN implementations, compression techniques [33] can create tens of efficient variants from any baseline DNN.

Each implementation has different resource demands and accuracy because it targets different conditions (e.g., lighting, camera angle, object sizes) and based on different statistical assumptions. The YOLO [51] object detector is accurate in scenes with just a few big objects but not otherwise, while background subtraction based detectors, a much cheaper technique, only works for moving objects in a fixed camera. In choosing associator implementations, the expensive SIFT features work well even in the presence of shadows, while the much cheaper color histogram is well-suited to track turning objects because the colors will likely be the same no matter their angle to the camera. Further, tracking a single rigid object (e.g., a car) on a fixed camera is simple, while tracking people in a dense crowd requires more sophistication. For these reasons, there is *no implementation that is universally accurate and cheapest* for all conditions.

Query plans: A video processing query typically includes multiple core vision primitives. Given that each primitive has tens of implementations to choose from and can also be configured with various knobs (e.g., different video resolutions or frame rates) [69], there are often hundreds or thousands of *query plans* – choice of implementations and knobs.

Common vision components across queries: Organizations often run tens of queries on *each camera stream* such as counting of various object types (cars, buses, trucks, SUVs, pedestrians), traffic violations like jay walking, and collision analysis between vehicles and bicycles, with our traffic video analytics partners [42]. These queries reuse the same core primitives like detectors, associators, and recognizers. Thus, we need to optimize across queries and allow *merging* these common components to substantially reduce resource demands. However, merging also constrains the common component to have the same plan (implementation choice) and placement across the many queries.



(a) Accuracy vs. Cores **(b)** Accuracy vs. Network rates

Figure 2: Resource-accuracy profile of the tracker.

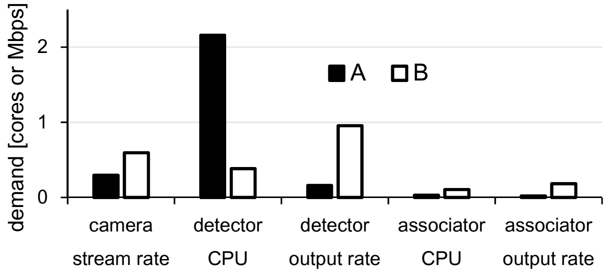


Figure 3: Network and CPU demands of the camera and the two modules in the tracker pipeline for two different plans, both with accuracy $0.73 - 0.75$**.**

C. Resource-accuracy Trade-off

Different query plans have different CPU and network demands, and accuracy. We illustrate this using an *object tracker*, which is a key building block for many video queries. An object tracker consists of two components: a *detector* detects objects in each frame of the video, while an *associator* associates those objects to existing tracks or starts new tracks. We use a representative traffic camera stream from a large US city with an original bitrate of 3 Mb/s at 30 frames/second to compare 300 query plans that vary the resolution, frame rate and different implementations of the detector and associator.

Resource demand vs. accuracy: To compute accuracy, we compare our output tracks against the ground truth obtained via crowd-sourcing. An object's track is a time-ordered sequence of boxes across frames, and in each frame we calculate the F1 score $\in [0, 1]$ (the harmonic mean of precision and recall [60]) between the box in the ground truth and the track generated by the tracker. When computing F1 score for a query result with lower frame rate, i.e., sampling, we compare the results on the sampled frame with its corresponding frame in the ground truth. We define accuracy of the track as the average of the F1 scores across all the frames. Figure 2 reports the trade-off between the resource demands – CPU and network demand (sum of output data rate of the detector and associator but exclude the input stream rate) – against accuracy. The wide range of demands and accuracies is caused by differences in implementations (§II-B). Factors like the light (shadows or overcast) and direction of the car (straight or making a turn) affect the accuracy.

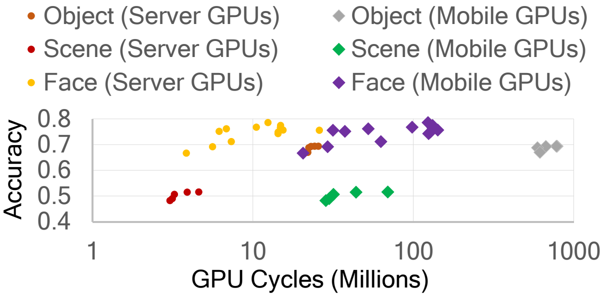


Figure 4: Profiling DNN recognizers – object, scene, face – on server-class and mobile GPUs.

CPU demand vs. network demand: Figure 3 shows that the compute demands and data rates for two example plans with similar accuracy. Plan A is based on a DNN detector, while plan B uses background subtraction. A's CPU demand is much higher than B's, but A's detector output and camera input is much lower. Depending on the resource constraints, both plans might be useful in optimizing for accuracy.

Object Recognizer DNNs: Resource-accuracy profiles are integral to video queries including license plate readers, DNN recognizers, etc. Figure 4 plots the resource-accuracy profile for different DNN recognizer implementations on GPUs: each for scene (AlexNet/MITPlaces [71]), face (DeepFace [58]), and object (the VGG16 model [56]) recognizers. We ran each model on a server-class GPU (NVIDIA K20) as well as a mobile GPU (NVIDIA Tegra K1) that is likely to be available in cameras. We generate less accurate but faster models using lossy techniques as in [33].

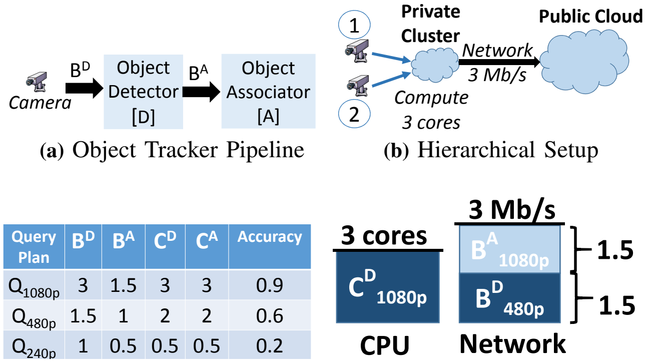
To select the best plan, ideally, we would like to establish an analytical model to estimate the accuracy and demand of a given plan, similar to SQL query operators [27]. However, this is infeasible because the demand and accuracy depend not only on the implementations, but also on the specific characteristics of the camera feed and even time of day. Instead, to estimate both accuracy and demand, we use an efficient profiler which is $100\times$ cheaper than a naive exhaustive exploration of the multi-dimensional space of query plans (§VI).

III. Motivating Example

In this section, we motivate the need for careful query planning, placement, and merging across a hierarchy of clusters using an illustrative example. Our objective is to *maximize the average accuracy* of video queries.

A. Queries and Cluster Setup

Consider the tracking query from §II-C with detector D and associator A; Figure 5a. The components have CPU demands C^D and C^A and input data rates B^D and B^A. Figure 5b presents the setup with two cameras connected to a private cluster. The private cluster has 3 cores and is connected to the public cloud with practically unlimited



(a) Object Tracker Pipeline

(b) Hierarchical Setup

Query Plan	B^D	B^A	C^D	C^A	Accuracy
Q_{1080p}	3	1.5	3	3	0.9
Q_{480p}	1.5	1	2	2	0.6
Q_{240p}	1	0.5	0.5	0.5	0.2

(c) Query plans for the tracker

(d) Utilization at private cluster for best plans & placement

Figure 5: Illustrative Example with two tracker queries (Fig. (a)) running on a hierarchical setup (Fig. (b)). Both queries Q_1 and Q_2 have the same profile of plans (Fig. (c)).

compute capacity; for ease of illustration, we assume no compute on the camera. The private cluster has a 3 Mb/s link to the public cloud and each camera has a dedicated 3 Mb/s link to the private cluster.

We consider three queries to execute on *each* of the camera streams – *car counting*, *jay walking*, and *collision analysis*. All the three queries build atop a tracker (detector → associator, Figure 5a). To simplify the example, we assume that the query components that consume the associator's output to count cars, identify jay walkers and analyze collisions consume negligible resources, and hence all the resource consumption is by the detector and associator components. The accuracy of the tracker's output directly translates to the accuracy of the outputs of each of these queries.

Assume that the only knob we control in the query plans is the frame resolution (which is configurable on the camera). Figure 5c shows the profile of the query plans. Both the accuracy of the tracker's outputs as well as its resource demands (data rates B^A and B^D, and CPU demands C^D and C^A) drop with the frame resolution.

B. Planning, Placement, and Merging

Each query has three query plan options (1080p, 480p, or 240p) and three placement options: (a) both components in the private cluster, (b) detector in the private cluster and associator in the cloud, (c) both in the cloud.[1] Hence, in our example, each query has 9 configurations (combinations of query plans and placements).

Merging: The only choice to run the six queries (three queries off each camera stream) is picking the 240p resolution for each query, Q_{240p}, leading to an average accuracy of only 0.2. Any other combination of query plans makes it infeasible to place the components due to insufficient compute or network capacity. However, in contrast to the current practice of treating these queries independently, we can *merge* their common components and run only one instance of the detector and associator for each camera's video stream, see Figure 6. *Merging saves us network and compute*; we avoid redundant streaming from the camera and execution of the components.

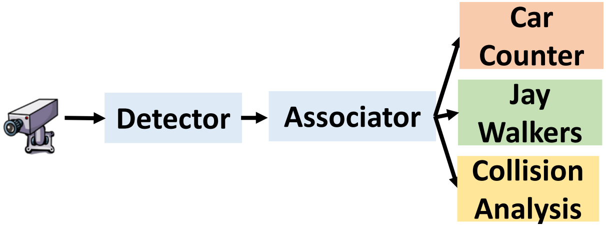


Figure 6: Merging the detector and associator of the "car counter", "jay walker", and "collision analysis" queries on the same camera stream.

Planning and placement: For each of the two merged tracking pipelines, Q^1 and Q^2, picking 1080p resolution maximizes the accuracy (Q^1_{1080p} and Q^2_{1080p}), but we cannot pick 1080p for both simultaneously because it is *infeasible* to place the components. Placing all the components in the cloud requires a bandwidth of 6 Mb/s ($B^D + B^D$) against the available 3 Mb/s. If one tracker's detector (needing 3 cores) is placed in the private cluster (capacity of 3 cores), the network link of 3 Mb/s between the private cluster and the cloud is still insufficient to support the aggregate data rate of 4.5 Mb/s ($B^D + B^A$ for 1080p). At the same time, the private cluster's compute is insufficient to support all the components locally.

Picking Q^1_{480p} and Q^2_{1080p} (or Q^1_{1080p} and Q^2_{480p}) is feasible and leads to the best average accuracy of $(0.9 * 3 + 0.6 * 3)/6 = 0.75$. However, this is feasible *only if* we place the detector of Q^2_{1080p} in the private cluster and its associator in the cloud, while forwarding Q^1_{480p}'s camera stream up to the cloud for executing both its detector and associator. Figure 5d shows the resulting utilizations at the private cluster. This illustrates the need to determine query plans and placements jointly, across all queries, and *consider multiple resources* (unlike current database optimizers [2], [20]).

Note that while the above example for merging was simplified, the decision is non-trivial in practice. This is because we need to select the *same plan* for the merged components which may often lead to *conflicting* accuracies. Car counting works better with background subtraction based object detector, while we need a DNN detector to identify pedestrians for jay walking. We have to resolve such *merging conflicts* and ensure that the best merged plan is not overly resource intensive.

[1] Placing the detector in the cloud and associator in the private cluster is clearly wasteful and we do not consider this placement option.

$A_{i,j}$	accuracy of plan j of query i
M_i	minimum accuracy of query i
C_l	capacity of resource l
$D^l_{i,j,k}$	demand on resource l of query i when using plan j and placement k
$S_{i,j,k}$	dominant resource demand of query i when using plan j and placement k
$x_{i,j,k}$	binary variable equal to 1 iff query i is using plan j and placement k

Table I: Notations for query i.

C. Desirable Features

To summarize, these are the desirable properties of a video query planner towards maximizing query accuracy: (i) jointly plan for multiple queries using their resource-accuracy profiles, (ii) consider component placement when selecting query plans to identify resource constraints, and (iii) account for multiple resources at the hierarchy of locations, (iv) merge common components across queries that process the same video stream. Achieving these properties is computationally complex owing to the combinatorial number of options.

IV. Problem Formulation

We begin with formulating our problem with an optimization model as well as an approximate, highlighting the challenges of applying them to solving our problem, and motivating the need for an efficient heuristic solution (explained in §V).

A. Notations and Definitions

Let $A_{i,j}$ represent the accuracy of plan j for query i. Our profiler provides us with the accuracy and resource demands for each plan and placement (covered in §VI-B). Let M_i be the minimum accuracy required for query i.

We model each cluster (e.g., camera or private cluster; Figure 1) as an aggregate bin of resources (CPU and network uplink and downlink) and only consider placement of query components *across* the clusters. We refer to each combination of resource type (e.g., uplink) *and* cluster (e.g., the camera) as a "resource" l. Let C_l be the capacity of resource l and $D^l_{i,j,k}$ be the demand on resource l from query i when running with plan j and placement k. We refer to each (plan, placement) pair as a *configuration*. Table I lists the relevant notations.

For the example in §III (Figure 5), placing the detector at the private cluster and the associator in the cloud uses the following resources: uplink of the camera and downlink of the private cluster (for the video), CPU and uplink of the private cluster (running the detector and shipping its output), downlink and CPU of the cloud (ingesting the detector's output and running the associator). [2] We deal with fluctuations in bandwidths in §VI-A.

B. Binary Integer Program

Without modeling the merging of queries, this problem can be considered as a Binary Integer Program (BIP):

$$\max \quad \sum_{i,j,k} A_{i,j} \cdot x_{i,j,k} \tag{1}$$

$$s.t., \quad \forall l : \sum_{i,j,k} D^l_{i,j,k} \cdot x_{i,j,k} \le C_l \tag{2}$$

$$\forall i : \sum_{j,k} A_{i,j} \cdot x_{i,j,k} \ge M_i \tag{3}$$

$$\forall i : \sum_{j,k} x_{i,j,k} = 1 \tag{4}$$

$$x_{i,j,k} \in \{0,1\} \tag{5}$$

where $x_{i,j,k}$ is a binary variable equal to 1 iff query i runs using plan j and placement k. The optimization maximizes the sum (equivalently, average) of query accuracies (Eq. 1), while meeting the capacity constraint for all resources l (Eq. 2) and ensuring a specified minimum accuracy M_i for each query (Eq. 3). Eq. 4 ensures that exactly one configuration is selected for each query. Despite providing optimal solution to the problem, solving BIP requires exponential time complexity, and is therefore not ideal for online solution.

Our formulation maps directly to the *multiple-choice multi-dimensional knapsack problem* (MMK) [46]. In the regular multi-dimensional knapsack problem, we are given n items, each with a specific size in D dimensions and a weight. The goal is to maximize the total weight of items that fit into a D-dimensional cube. In MMK, additionally, each item has up to m *incarnations*. In selecting the items, we can pick any of the incarnations of each item. [46] shows a polynomial 2-approximation algorithm in m and n with complexity of $\mathcal{O}((m{\cdot}n)^D)$, where n is the number of queries and m is the number of configurations each query could have.

We could apply this algorithm to our setting with each each query being an item and each configuration of a query being its incarnation. Each resource corresponds to a dimension of the bin and the size of each incarnation (configuration) in dimension l is the demand on resource l. If we set the weight of an incarnation as the accuracy of the corresponding query configuration, solving MMK will maximize total accuracy within the resource capacities – exactly our BIP formulation. Note that we can remove all configurations which have accuracy below M_i to ensure that each query achieves its minimum accuracy.However, the algorithm described in [46] is impractical in our scenario for two reasons. 1) First, the number of resources D is very high in our setting, often proportional to the number of queries n because there are a only handful of queries processing each camera stream; $D = O(n)$. Recall that we define the combination of resource type (e.g., cores) and cluster (e.g., camera) as a resource. Hence, the uplink bandwidth and compute of *each* camera is a unique resource. As the number of cameras is $O(n)$, so is the number of resource types. The complexity of the approximation algorithm in

[2] Downlink bandwidths usually *far exceed* the uplink bandwidths.

[46] thus becomes $O((m{\cdot}n)^n)$, which is exponential with the number of queries and is too slow in practice. 2) Second, the description so far ignores *merging of queries*. We can extend the formulation to handle query merging as follows. Only the queries that process video from the same video stream can merge; we thus logically group all queries on the same stream into *super-queries* and enumerate all the different configurations (or incarnations) of each super-query as all combinations of possible ways to merge these queries, their configurations and placement. However, this would make m exponential in the number of queries per stream and number of modules in the query DAGs, again making the algorithm too slow to use in practice.

We are unaware of other (approximate) algorithms with polynomial complexity that apply in this setting. Instead, we develop an efficient heuristic next in §V.

V. VideoEdge's Video Query Optimization

VideoEdge is a video query optimization framework that *jointly optimizes all queries to maximize the average query accuracy* within the available resources. Specifically, we *plan* for each query (pick its implementations and knobs), *place* components of the query across the hierarchy of clusters, and *merge* identical components of queries that process the same stream to save resources.

A. Dominant Resource Demand

VideoEdge's profiler (§VI-B) estimates demands ($D^l_{i,j,k}$) and accuracies ($A_{i,j}$) for each query configuration. To decide between two configurations c_0 and c_1, we need to compare their accuracies and resource demands. Is c_1 improving the accuracy *enough* for the amount of additional resources it consumes? However, because the queries utilize multiple resources across many clusters (see §IV-A), it is tricky to compare resource demands.

Therefore, we define a *dominant demand* which converts demand for multiple resources into a single value; specifically, the dominant demand of placement k of plan j for query i $S_{i,j,k} = \max_l D^l_{i,j,k}/C_l$. S is a scalar that measures the highest fraction of resources l needed by the query across resource types (CPU, network) and clusters (camera, private cluster, cloud).

A nice property of the dominant demand metric is that, by normalizing the demand D relative to the capacity C of the clusters, it avoids lopsided drain of any single resource at any cluster. As a critical insight, if the system runs out of network bandwidth between two clusters, no more data can go through them and their remaining CPU resources are wasted. Also, by being dimensionless, it easily extends to multiple resources, akin to DRF [31]. While we also considered defining $S_{i,j,k}$ using sum of the resource utilizations ($\sum_l$ instead of $\max_l$) or just the absolute demands, they performed worse in our evaluations.

```
1:  $U$          ▷ Set of all $(i,j,k)$ tuples of all queries $i$ and the
    available plans $j$ and placements $k$
2:  $p_i$                                  ▷ Plan assigned to query $i$
3:  $t_i$                             ▷ Placement assigned to query $i$
4:  for all query $i$ do
5:      $(p_i, t_i) = \arg\min_{(j,k)} S_{i,j,k}$     ▷ Agg. demand $S$, §V-A
6:  for each resource $l$: update $R_l$
7:  while U $\neq \emptyset$ do
8:      U' $\leftarrow$ U $-\{(i,j,k)$where $\exists l : R_l < D^l_{i,j,k}\}$
9:      remove $(i,j,k)$ from $U$ if $A_{i,j,k} \leq A_{i,p_i,t_i}$
10:     $(i^\star, j^\star, k^\star) = \arg\max_{i,j,k \in U'} E_i(j,k)$
11:     $p_{i^\star} \leftarrow j^\star$, $t_{i^\star} \leftarrow k^\star$
12:     for each resource $l$: update $R_l$ based on $D^l_{i^\star,p_{i^\star},t_{i^\star}}$
```

Figure 7: Pseudocode for VideoEdge's heuristic.

B. Greedy Heuristic

We first describe our heuristic without considering merging and then incorporate it in §V-C. To maximize average accuracy, it is crucial to efficiently utilize the available resources. We employ the intuitive principle of allocating more resources to queries that can achieve higher accuracy per unit resource allocated compared to other queries. To achieve this, we use an *efficiency* metric that relates the accuracy to the dominant demand of the query.

Our heuristic starts with assigning the configuration with the lowest dominant resource demand to each query and greedily considers incremental improvements to the queries. [3] When considering switching query i from its current plan j and placement k to another plan j' and placement k', we define the efficiency of this change as the *improvement in accuracy* normalized by the required *additional demand*. Specifically:

$$E_i(j', k') = \frac{A_{i,j'} - A_{i,j}}{S_{i,j',k'} - S_{i,j,k}}$$

Defining $E_i(j', k')$ in terms of the "delta" in both accuracy and demand turns out to be the most suited to our gradient-based search heuristic. It outperformed alternate definitions that just used only the new values, e.g., only $A_{i,j'}$ and/or $S_{i,j',k'}$ in our evaluations.

Figure 7 shows the pseudocode. U represents the set of all (i,j,k) tuples of all queries i, the available plans j, and placements k. The objective is to assign to each query i, a plan p_i and placement t_i; lines $1-3$. It first assigns each query i the plan j and placement k with the lowest dominant demand $S_{i,j,k}$ (lines $4-5$). After that, it iteratively searches across all plans and placements of all queries and selects the query $i^\star$ (and the corresponding plan $j^\star$ and placement $k^\star$) with the highest efficiency (lines $7-12$). Line 9 ensures that we only consider configurations that increase accuracy. It switches query $i^\star$ to new plan $j^\star$ and placement $k^\star$, and repeats until no query can be upgraded (either due to

[3] We only consider plans j with accuracy $A_{i,j} \geq M_i$.

insufficient remaining resources, or no available plans with higher accuracy).

In each iteration, we only consider configurations that fit in the remaining resources R_l by constructing U' (line 8). Note that we cannot remove such infeasible configurations from U completely because they might become feasible later as the heuristic moves components across clusters by changing configurations of queries.

A subtle aspect is that in each iteration, we remove those options from U that reduce a query's accuracy relative to its currently assigned plan and placement (line 9). Such an explicit removal helps because even though the change in accuracy of the removed options would be negative, those options may also have negative difference in dominant utilization ($S_{i,j,k}$), thus making the efficiency positive and potentially high.

Note that our heuristic can also work with other performance goals. For example, if each query has the requirement of minimum accuracy to be useful, our proposed heuristic can first focus on getting all queries to a configuration that meets minimum accuracy before further improving overall accuracy by selecting other configurations with higher accuracy. Our heuristic can also work to achieve fairness across queries: each iteration improves for the query with minimum accuracy, until the resources are depleted.

The computational complexity of our heuristic without merging is $O((m \cdot n)^2)$. There are at most $n \cdot m$ iterations before the proposed heuristic terminates since each iteration removes at least one configuration from consideration and there are total $n \cdot m$ configurations. In each iteration, at most $n \cdot m$ configurations are explored, therefore the overall time complexity is bounded by $(m \cdot n)^2$. This is much more efficient than [46] (§IV-B).

Note that we do not change the plan and placement of the running queries in the system while the heuristic is running but *only when all its iterations complete*.

C. Merging Peer Queries

When there are multiple queries processing the same camera feed with common prefix in their pipeline, we have the opportunity to eliminate running redundant components. We refer to such queries as a *peer set*. Figure 6 shows an example of peer set of queries. The directional car counting query in traffic video uses a detector component for identifying the vehicle, then a mapper component for keeping track of the same vehicle, and finally a counter component to record the number of vehicles per direction. Anomaly detections also rely on an object detector component and a mapper component to understand trajectories and to identify anomalous behaviors. Since both queries require the detector and mapper components, running only one copy of these components can save resources (Figure 6) which in turn can be used to upgrade other queries' plans. We call this as *merging*.

Challenges: Despite the obvious resource benefits of merging, the decision is non-trivial. Merging two peer queries processing the same camera feed with common components is not always beneficial to their accuracies. This is because, the merged components have to be assigned common implementations and knob values. However, a query plan that achieves good accuracy for one query may not do so for the other query. When counting cars and humans in the same camera stream, different implementations of detectors and mappers are better suited. A background subtraction based detector and distance based mapper are better suited for cars. On the other hand, humans, owing to their smaller size are often missed by a background subtracter and require a richer SIFT metric for mapping. Hence, the decisions on merging need to contend with such conflicts. Even when there are no conflicts on accuracy, picking the plan with the common *maximum* accuracy might not be optimal as it might be too resource-intensive. Furthermore, whether to merge the peer queries or not depend not just on the overall accuracy they can achieve, but also depend on the available resources and, most importantly, the trade-off between them. While merging peer queries could reduce resource usage, given sufficient resources, some of the peer queries may end up having its own isolated module pipeline to gain better aggregate quality. A key question in merging peer queries is deciding the implementation and knobs for the merged components. In addition, the decision of merging is not just for the peer queries involved, but should also consider the aggregate quality for all queries in the system as the planning and placement of other queries would also be affected. Finally, the possible merging combinations grow exponentially for a peer set of z queries (any subset of queries in a peer set can be merged). Jointly performing planning, placement and merging is markedly different than multi-query optimizers in databases [30], [54].

To reduce the search space, we make two simplifying assumptions when considering merging: (a) we either merge *all the common components* or nothing at all. For the example in Figure 6, we either merge both the detector and associator or neither of them; we do not consider merging only the detector.Consider partial merging provides more flexibility, yet it would increase the complexity significantly. Exploring more merging potions at reasonable overheads is part of our future work. (b) we avoid searching through all possible plans for the components that are not common ("counter", "jay walker" and "collision" components in Figure 6) and use their plans from the previous iteration of the heuristic. The distinct components in vision queries usually tend to not have many choices in their plans, and hence assumption (b) does not hurt our solution.

To realize merging peer queries, we make the following change to line 10 of Figure 7. When considering switching to configuration $(p_i^\star, t_i^\star)$ of query $i^\star$, we also consider merging this query with *all subsets* of its peer queries. Let R be one

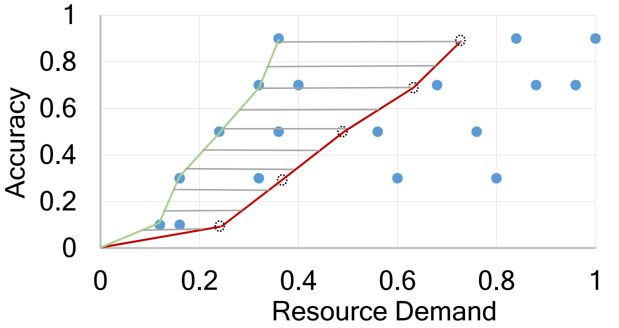


Figure 8: Illustration of Pareto band (shaded) for a single query. Note that for each accuracy (plan), there is a horizontal stripe of placement options with different demands.

of the subsets of $i^\star$'s peer queries. We merge all queries in R with i and apply the $(p_i^\star, t_i^\star)$ configuration to all components in $i^\star$. Any remaining components in the merged query (that are not in $i^\star$) remain in their current plan and placement. For each such merged query, we compute the efficiency metric E relative to *all peer queries of* $i^\star$, i.e., ratio of the aggregate increase in accuracy to the aggregate increase in the resource demand. The computational complexity of our heuristic, after incorporating merging queries, becomes $O((n{\cdot}m)^2{\cdot}2^z)$, in which z is the number of queries in a peer set: for each configuration considered in each iteration, the exploration is extended to consider all 2^z possible subsets of a query's peer queries. [4]

D. Pareto Band

To speed up the heuristic, we significantly reduce the size of the exponentially-large set U by explicitly filtering out query configurations that have low accuracy and high resource demand. For example, the configurations in the bottom-right corners of the tracker query in Figures 2a and 2b are unlikely be selected by the heuristic.

We build upon the classic economic concept of Pareto efficiency [61] to first identify the Pareto boundary of query configurations. Figure 8 plots an illustrative resource-accuracy space for a query with the left green line being the Pareto boundary. For a particular query i, a configuration c is on the Pareto boundary if there does not exist any other configuration with lower demand and higher accuracy. For every point *not* on the Pareto boundary, there is at least one point on the boundary that beats it in *both* accuracy (higher) and demand (lower).

However, limiting our search to only the configurations on the Pareto boundary can be problematic when optimizing for multiple queries. Note that the dominant demand S in §V-A is defined in terms of the resource capacities and not *availabilities*. Thus, when VideoEdge tries to select a better configuration for a query, all the configurations on the Pareto boundary may be infeasible due to insufficient resource capacity (line 8 in Figure 7).

Therefore, to reduce the size of set U but not restrict the heuristic too much, we define a "band" relative to the boundary, which we refer to as *Pareto band*. We first define a δ-boundary to consist of points $(\delta d, a)$ for all points (d, a) on the Pareto boundary. The Pareto band thus consists of configurations between the Pareto and the δ-Pareto boundaries. See an illustration with $\delta = 2$ in Figure 8 (red line is the δ-Pareto boundary). Making the band's width relative to the Pareto boundary provides a cluster of placements with comparable demands. We *only search among the query configurations within the band* using our heuristic (set U in §V-B). The big variation of the configurations in the resource-accuracy space, endemic to video queries, typically results in considerable reduction in the search space.

We see in our evaluations that $\delta = 2$ provides over 90% of the accuracy as with using all the configurations ($\delta \rightarrow \infty$). While we considered specializing δ values per query, a single uniform value suffices for our workload.

E. Resource Pricing

Using certain resources like cellular bandwidths out of the cameras and compute cores in the public cloud incurs monetary costs proportional to their usage. We allow the user to specify total budget in terms of \$/month, after which we convert it to budget B *per unit time*, such as per minute. We then run VideoEdge's heuristic in each small time unit with the following modification. [5] Having a fast heuristic allows such frequent runs.

Assume that the resources $Q_1, \ldots, Q_v$ have prices $W_1, \ldots, W_v$ per unit demand and unit time. A configuration for query i with plan j and placement k has demand $D_{i,j,k}^l$ on resource l. Thus, the total cost of this configuration is $W_{i,j,k}^T = \sum_l (W_l \cdot D_{i,j,k}^l)$. Thus, before running VideoEdge, we create a *new virtual resource* with capacity B, i.e., the total budget. The demand on this new resource will be $W_{i,j,k}^T$. After this, we run the regular VideoEdge heuristic, thereby ensuring that the budget will be met.

VI. System Design

We now describe VideoEdge's systemic design details.

A. Handling Bandwidth Changes

Video queries usually run continuously and the available bandwidths are likely to fluctuate while the queries are running. For example, if the up-link capacity reduces, we might not be able to stream the video from the camera at the chosen resolution and frame rate.

We handle bandwidth fluctuations by constantly monitoring the available bandwidths (using techniques used in prior work [39], [44]); if any of the bandwidths changes

[4] In practice z's value is usually a small constant and far less than n.

[5] While we will never overshoot the budget but may under-utilize it.

significantly for a certain amount of time, VideoEdge recomputes its query optimizations. To minimize the disruption, we only recompute the query plan selection, while retaining the current placement and merging decisions. In addition, the complete version of VideoEdge is periodically re-run to update the queries' plans, placements and merges to ensure high resource utilization.

Also, we constantly run golden queries to monitor and calibrate the accuracy of the queries. Our system can be configured to re-run the scheduling heuristic periodically (e.g., every 3 hours) to account for any dynamics and therefore maintain high accuracy.

B. Resource-Accuracy Profiler

The profiler is a key component of VideoEdge. For each query, it estimates accuracy and per-component resource demands (CPU and bandwidths). Note that the profiler does not consider *placement*; the optimizer in §V jointly incorporates placement with planning and merging.

The profiler estimates the query accuracy by running the query on a labeled video dataset obtained via crowd-sourcing or by labeling the dataset using a "golden" query plan which might be resource-intensive but is known to produce highly accurate outputs. When a user submits a new query, we start profiling it while submitting it to the scheduler with the default query plan.

Since a query can have thousands of plans which we have to execute on the labeled videos, the main goal in profiling is to *minimize the CPU demand of the profiler*. We use two simple tricks: (1) eliminating common sub-expressions by *merging multiple query plans* and (2) *caching intermediate results* of query components.

Assume that both components in the tracking query $D \rightarrow A$ have two implementations; D_1, D_2 and A_1, A_2. We thus have to profile four query plans: D_1A_1, D_1A_2, D_2A_1, and D_2A_2. If we run each plan separately, implementations D_1 and D_2 would run twice on the same video data. Instead, we *merge* plans of profiled queries, similar to §V-C, to avoid redundant runs.

While we could merge all the plans into one, executing this would require large number of concurrent compute slots which might not be available. In such cases, we resort to *caching* of intermediate results. Note that while caching alone will eliminate redundant executions, it has dramatically high requirement on storage space. In profiling the tracker on a 5-minute traffic video, the storage space for caching is 78× the size of the original video.

Hence, we assign a *caching budget* per query. We preferentially cache the outputs of those components that take longer to generate. In addition, we also like to cache the outputs of those components which are to be used more. These are components with many downstream components each with many implementations and knob choices. We encode these in a metric for each intermediate result, $M = n \times \frac{T}{S}$ where n is the number of times this output will be accessed, T is the time taken to generate the output, and S is the size of the output. Our profiler uses the caching budget for intermediate outputs with higher value of the M metric.

Benefit: For the tracking query, our profiler uses 100× fewer CPU cycles compared to exhaustive exploration of all plans. We use cache budget of 900MB per machine, which we believe is practical in modern machines. On a 16-core machine, our profiler took 10 minutes to complete, which could be parallelized across more machines. Such improvement in efficiency, allows us to run the profiler periodically and use the most recent resource-accuracy profile in our optimizer in §V.

VII. Evaluation

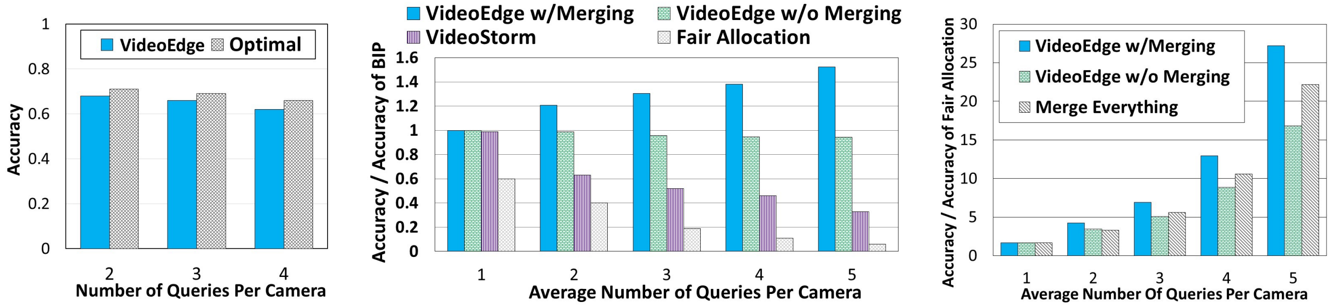
We evaluate VideoEdge with an Azure deployment emulating a hierarchy of clusters using representative video queries, and complement it using large-scale simulations.

1) VideoEdge outperforms state-of-the-art solutions by up to $25.4\times$ better average accuracy, while being within 6% of the optimal accuracy.
2) Merging queries with common component accounts for an additional $1.6\times$ better accuracy.
3) Searching only the configurations in the Pareto band drops the heuristic's running time by 80% while still achieving $\geq 90\%$ of the original accuracy.

A. Setup

Azure deployment. We use a 24 node Azure cluster to emulate a hierarchical setup; each node is a D3v2 instance with 4 cores and 14GB of memory. Ten of the nodes are the "camera compute"; two cameras per node. The 20 cameras "play" feeds from 20 recorded streams from many cities in the USA at their original resolution and frame rate. Two nodes act as a private cluster. Each camera has a 600Kb/s link to the private cluster, resembling the bandwidths available today. The cloud consists of 12 nodes with a 5Mb/s uplink from the private cluster. These bandwidths are based on measurements at our partner traffic jurisdictions at the City of Bellevue, WA and elsewhere. They are also consistent with recent studies from Akamai [1]. We also use simulations to evaluate larger settings under varying resource capacities.

Video queries. We profile and evaluate using the following queries: tracker, DNN object classifier (trained in advance), car counter, and license plate reader – a typical combination of queries. The queries have 300, 20, 10, and 30 query plans, respectively, from different implementation and knob choices. These queries are pre-defined in our experiments so that we can profile them to obtain the resource-accuracy profiles with various configurations. Each query has two components and among the three clusters in the hierarchy there are six placement options per query: both components in the same cluster or each in a different cluster



(a) Small-scale Setting.

(b) Comparison with Baselines.

(c) Impacts of Merging.

Figure 9: Improvement in Accuracy.

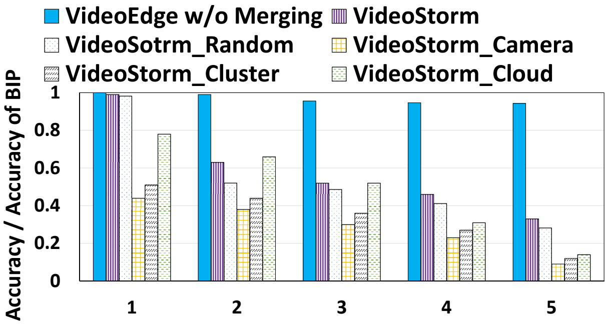


Figure 10: Comparing VideoEdge to VideoStorm with different placement options

(we avoid placements that go "down" from the cloud). We use 200 video clips (each 5-minutes long) from many locations and times of day, and thus generate 200 resource-accuracy profiles for our experiments. The groundtruth for these video queries are obtained by manually labeling. Each query has 300 configurations; 5 resolution and 5 sampling rate values, 3 object detector implementations (two based on background subtraction and one on DNN) and 4 different tracking metrics. Since there are 6 different placements for the tracker query components, our heuristic considers 1800 total configurations. Each experiment is run 5 times, and the median result reported.

Baselines. We compare against four approaches.

(1) Fair Allocation, since it is widely used in production clusters [19], [34]. We extend the definition of fairness used within a cluster ($\frac{1}{n}$ of the resources in a cluster given n queries) to multiple clusters by allocating $\frac{1}{n}$ of *each resource in each cluster*. Within this fair allocation, each query picks the configuration (query plan and placement) that achieves the highest accuracy. [6]

(2) Recent work on video analytics, VideoStorm [69], a single-cluster, single-resource video query planner. We use the aggregate CPU of all the clusters for its planning. Being a single-cluster solution, VideoStorm does not include placements and may selects query configurations that are supported by bandwidth capacity. We place its query components starting in the cameras, and move on to place them at private clusters if the cameras do not have sufficient compute CPU resources, and eventually move to the cloud if both the cameras and the private clusters run out of CPU resources. Such placement method for VideoStorm may not be ideal, but provides fair insights in comparison without deviating from its design spirits. We also ensure that it avoids infeasible placements (e.g., insufficient network or compute resource). In our experiments we also compare VideoEdge to VideoStorm with additional placement options.

(3) To compute the optimal plan and placement *without* merging, we solve the BIP optimization (§IV-B) with the Gurobi solver [9]. We compute the optimal results *with* merging using brute-force computation.

Performance metric. Our performance metric is the *average accuracy across all queries.* We report the relative improvement over the baseline; given accuracies of VideoEdge and a baseline are A_c and A_b, we report $\frac{A_c}{A_b}$.

B. Improvement in Accuracy

We begin with presenting the improvement in accuracy using our Azure deployment. We run the tracking queries and assign them uniformly to the cameras; each camera runs a randomly chosen video from the 200 options. We compare against the two optimal strategies: 1) brute-force, which considers merging but only scales to small deployments, and 2) BIP, which does not consider merging but scales to our default deployment.

First, we use the brute-force optimum in a small-scale setting with just one camera, one private cluster, and cloud and vary the number of queries from two to four.[7] Even at this small scale with four queries, brute force takes 400 CPU days to complete, compared to VideoEdge completing in 1.5 CPU seconds. Figure 9a shows that, although the average

[6] While we considered using DRF [31] as our fair allocation baseline, the DRF algorithm is intertwined with component placement unlike a simple fair allocation. This is because, DRF requires the *demand at each cluster* even to define the dominant fair share, which in turn is a function of placement and not known beforehand.

[7] To quantify gains from merging, we only include queries with the same components, e.g., a detector and an associator.

accuracy decreases with more queries sharing the resources, VideoEdge achieves within 93% – 96% of optimum.

Next, we compare against the BIP optimum that does not consider merging in our default-scale setting, see Figure 9b. We also include VideoEdge without merging since the baselines do not merge queries. We make several observations. First, VideoEdge w/Merging constantly outperforms BIP when merging queries is possible (i.e., more than 1 query per camera), and the gap increases as there are more queries in the system. This suggests that VideoEdge's consideration of merging queries plays a key role in VideoEdge's effectiveness. Second, VideoEdge significantly outperforms VideoStorm and Fair Allocation and the gains increase as the number of queries increases: its accuracy is 5.4× and 25.4× better than VideoStorm and Fair Allocation, respectively, with 5 queries executing on every camera stream. Finally, without the consideration of merging queries, the accuracy of VideoEdge w/o Merging is within 94% of BIP even with the increasing number of queries in the system, and is 3.3× and 15.7× better than VideoStorm and Fair Allocation, respectively, with 5 queries executing on every camera stream.

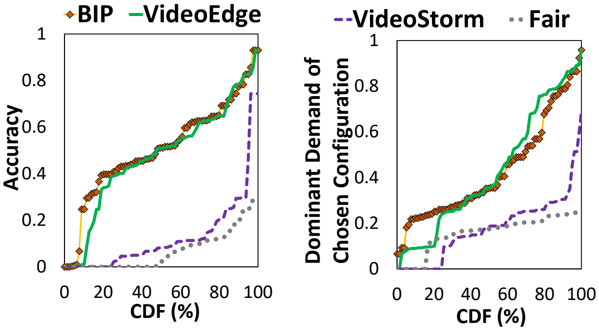
Figure 10 further compares VideoEdge to VideoStorm with different placement options. The additional placement options for VideoStorm in this experiment include: (a) placing all query components in the cameras (VideoStorm-Camera), (b) placing all query components in the private clusters (VideoSotrm-Cluster), (c) placing all query components in the cloud (VideoStorm-Cloud), and (d) placing query components randomly across the locations, while making sure it does not exceed bandwidth capacity. The results suggest that VideoEdge outperforms VideoStorm in all the cases, including the hierarchical placement option we described earlier. We use the hierarchical placement option for VideoStorm in our experiments since it provides the best accuracy for VideoStorm. From the experiments in Figure 10 we learn that "retrofitting" query placement after query plans are selected is less beneficial as compared to VideoEdge's approach of making *joint* decision on the two factors.

Merging conflicts. While merging queries can reduce resource usage and improve accuracy by 1.6× (see Figure 9b), recall that it can lead to "conflicts" between queries when they have different resource-accuracy profiles, thus needing us to weigh the resource gains from merging against any loss in accuracy. To highlight this, we compare against a "Merge Everything" heuristic that merges all possible components without considering conflicts, see Figure 9b; it achieves much smaller gains.

C. Characterizing the Gains

Next, we characterize VideoEdge's gains against the baselines in the scenario of no-merging consideration.

Resource efficiency. Figure 11a presents the distribution of the absolute query accuracies with the "2-queries-per-camera" setting. VideoEdge's CDF closely matches BIP, which shows that VideoEdge's greedy heuristic search in the Pareto band is near-optimal even at a per-query level, not just in aggregate. The key to VideoEdge's performance is effective utilization of resources. The CPU and network utilizations are all above 85%, thus showing the effectiveness of VideoEdge in balancing the load across clusters and avoiding bottlenecks. This is mainly due to our metric of dominant demand (§V-A) that prevents any single resource from being disproportionately utilized. This is supported by Figure 11b that shows that the queries' dominant demands achieved by VideoEdge are significantly higher than with fair allocation and VideoStorm, and similar to BIP, thus leading to higher utilizations and accuracies.



(a) Accuracy. **(b)** Dominant Demand.

Figure 11: CDF of Accuracy And Dominant Demand.

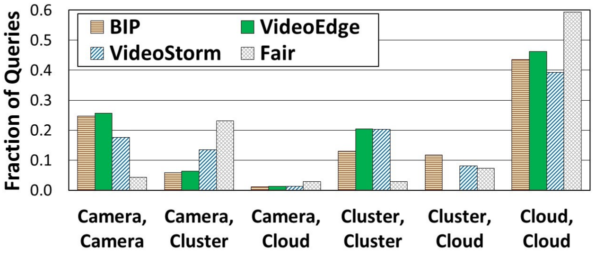


Figure 12: Choice of placements for components.

Placement decisions. Figure 12 presents the distribution of the six options for placing the detector and associator. For 93% of queries, VideoEdge places both components of a query at the same location, i.e., "Camera-Camera", "Cluster-Cluster", and "Cloud-Cloud". As a result, the intermediate data between the components does not use the network between clusters, thus avoiding contention. VideoStorm's random placement strategy places 23% of the query components across clusters which adversely impacts its query plans and accuracies.

Next, we evaluate VideoEdge with constrained query placement – like in many production video analytics deployments – to one level in the hierarchy. All the queries run (i) on their corresponding camera, (ii) in the private cluster, or (iii) in the cloud. Figure 13 shows the ratio of VideoEdge's accuracies (without merging) over each of

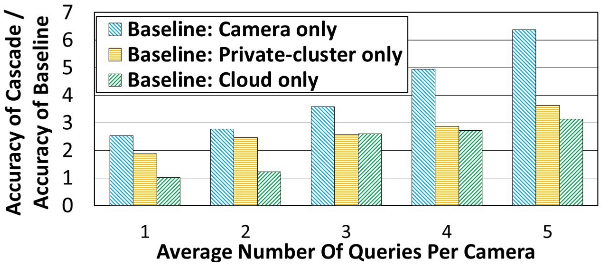


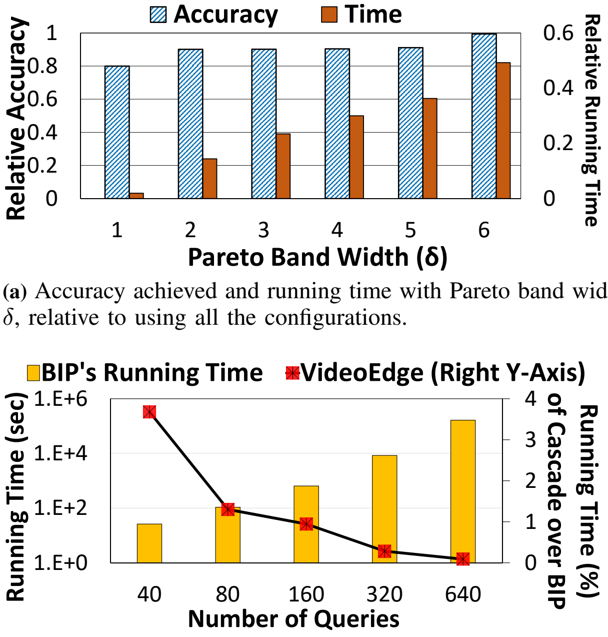
Figure 13: VideoEdge**'s gains over restricted placements.**

the three constrained approaches. Note that each query configuration could have different knob values, e.g., video resolution, which requires decoding differently at the cameras. Hence the same video source could be transmitted with multiple streams each with different decoding scheme, and this depends on the number of queries running over the same camera. As the number of queries (per camera) increases, the compute or network resources become saturated and they cannot support the queries at high accuracies. For example, with just one query per camera, we can stream all video to the cloud and process it there; however, as the load increases, this becomes harder and VideoEdge can run the queries with up to 3× higher accuracy. We achieve the largest gains against the camera-only constraint because the cameras have the least compute available. These results highlight the value in jointly utilizing the hierarchy of clusters for video analytics.

Gains by query type. We also evaluate a wider (equal) mix of query types – object tracker, DNN classifier, car counter, license plate reader – in our simulator. We profile these queries using our profiler (§VI-B) and feed these to the simulator. The cluster settings are similar to the deployment. We measure that the license plate reader and car counter are farther from the optimal (87% of BIP) unlike the other two that are near-optimal. This is explained by the resource-accuracy profiles of the queries. The license plate reader and car counter, beyond a certain accuracy, have *inefficient* profiles (see §V-B). The additional resources needed to improve their accuracy are higher compared to the object tracker and DNN queries which have many more efficient configurations. As a result, our heuristic assigns more resources to the latter two.

D. Scalability with Pareto Band

In §V, we narrow down our search space by using a Pareto band of promising configurations. The smaller the width of the band (δ), the faster the running time of the heuristic at the expense of lower accuracy. Figure 14a presents the impact of δ on the accuracy and running time normalized to considering all the configurations (i.e., $\delta \to \infty$). Even with $\delta = 1$, of the Pareto *boundary*, we see that the relative accuracy is 80% of searching through the entire space. With $\delta = 2$, we achieve a relative accuracy of over 90% in under a fifth of the time. Thus, we use the Pareto band with $\delta = 2$ in our system.



(a) Accuracy achieved and running time with Pareto band width δ, relative to using all the configurations.

(b) Running Time.

Figure 14: Scheduling Overhead And Scalability.

We also compare the running time of the heuristic. Figure 14b compares VideoEdge's running time to the BIP (§IV-B). The left Y-axis stands for the BIP's time while the right Y-axis stands for VideoEdge's relative running time. The BIP optimization's complexity grows considerably faster than our heuristic ($O(n^2 \cdot m^2)$ for n queries each with m configurations. VideoEdge takes only $0.09\% - 3.7\%$ of the time taken to solve the BIP.

E. Resource Pricing Budget

To evaluate how VideoEdge incorporates cost constraints (§V-E), we use cloud CPU as the paid-resource. We first run VideoEdge without any cost constraint, and obtain its cost based on its cloud CPU usage and the Azure pricing info [6]; we consider this cost as the maximum cost. We then apply a cost budget on VideoEdge ranging from 0% to 100% of this maximum cost, and measure the change in accuracies.Two main characteristics stand out. First, as the budget shrinks all the way to one-fifth of the maximum cost, the license plate and DNN classifier queries see far more drop in their accuracies (by 4×) because they are much more compute-intensive while the accuracy of the tracker and counter queries drops by only 26%. The latter two queries switch from the expensive DNN object detector to cheaper background subtractor. Second, the placement of the license plate and DNN classifiers shift more to the private cluster instead of using the public cloud. Overall, we observe that even in the face of shrinking budgets, VideoEdge smartly adapts with alternate choices on query plans and placements.

F. Adapting to Resource Capacity Changes

Recall from §VI-A that VideoEdge adapts to changing bandwidths in the hierarchy of clusters. Our experiments in monitoring uplink bandwidths out of private clusters (from city jurisdictions) as well as between VM instances on Azure showed that the bandwidth can vary by up to a factor of $2\times$. Based on these measurements, we evaluate VideoEdge's reaction to changes in bandwidth from its normal value of X to between $0.5X$ and $1.5X$.

We notice that increase in bandwidth capacity does not lead to much increase in accuracy (by 10%) of the queries, while decrease in bandwidth capacity drastically drops accuracy by 41%. Since the selection of query plans depends on the multi-resource allocation, increasing only the network resources may not significantly improve the overall accuracy due to the bottleneck at compute resources. On the other hand, decreasing network capacity creates bottlenecks in the network, which directly forces the selection of query plans with lower accuracy. Also, as the bandwidth becomes a constraint, VideoEdge selects the DNN object detector which outputs fewer object boxes and thus uses less bandwidth, instead of using background subtractor.

VIII. Related Work

Big data jobs: Placement of VMs (e.g., Oktopus [23], FairCloud [48]) or tasks of big data jobs (e.g., Yarn [19], Mesos [34], Apollo [24], Borg [64]) has been an important research direction. In this line of work, a set of tasks make exact resource requests and is placed in the cluster to maximize utilization. However, such schedulers are typically deployed in a single cluster. Recent work on wide-area analytics ([37], [49], [65], [66]) propose query optimization across geo-distributed clusters. Recent work on wide-area analytics – such as Geode [66], Iridium [49], Clarinet [65], Pixida [37] – propose query optimization across geo-distributed clusters. They, however optimize *batch* queries, not stream-processing queries. Specifically, none of the above mentioned works address the joint decision of query planning, placement and merging as in our problem setting.

Databases: Streaming databases [17], [18], [38], [45], [63] considered the resource-accuracy tradeoff but did not deal with multiple plans (only sampling rate), multiple resources (only memory), or a hierarchy of clusters. There exist many works on multi-query optimization in database systems [26], [29], [30], [52], [54], in which concurrent queries are jointly considered for join order selection or placement across distributed machines in order to either optimize for system utilization or minimize query response time. However, none of them addresses joint planning, placement and merging of multiple queries, especially merging queries is a new challenge brought by running streaming video queries. Works such as [47], [57] do joint placement and merging of components to optimize network utilization, but ignore CPU and do not consider a large number of query plans.

Video analytics has been increasing in popularity: MCDNN [33] uses different versions of DNNs to trade off resource usage and accuracy but does not consider placement across a hierarchy. Optasia [43] writes video queries in SQL and uses SQL optimizers but ignores the resource-accuracy profiles in selecting the query plans; it does not address placing query components across distributed machines either. VideoStorm [69] optimizes query knobs and resource allocation to improve both query accuracy and delay, however only considers the CPU resource in a single cluster. Therefore, VideoStorm cannot be trivially applied to our problem setting (i.e., hierarchical clusters) as we showed in Section §VII. Chameleon [35] is the recent video analytics work for continuously adjusting DNN configurations to optimize accuracy or reduce resources costs based on the temporal and spatial correlation among the video frames. Such techniques could also be applied to our work, while Chameleon does not address query merging opportunity, which contributes to significant gains in accuracy as shown in our work.

Mobile Offloading: Offloading expensive operations from a resource-constrained mobile device to the cloud has been a popular area [21], [22], [32], [40]. Previous works [28], [32], [50] automatically provide a runtime to off-load methods to the cloud and adjust execution parallelism to improve responsiveness and accuracy. Compared to offloading, VideoEdge considers many queries together and optimizes over its query plans and placements to resolve conflicts. Starfish [41] eliminates redundant computation in vision applications on a mobile device but our context also requires jointly planning and placing all the queries while resolving any conflicts.

Sensor networks is an area where multiple queries execute in a hierarchical resource constraint environment [55], [59], [67]. TTMQO [67] proposes a two-tier query optimization framework, where multiple queries are first merged offline and their execution is further optimized in the network by minimizing the number of messages sent (sampling rate). Our context, however, differs in that we have to optimize over many query plans and placement of query components.

IX. Conclusion

Analyzing live video streams over hierarchical clusters has become an important problem. Video analytics queries have multiple implementations and knobs that decide their accuracy and resource demand. We devise VideoEdge to decide these choices, place the queries across the hierarchy, and merge queries with common processing. To navigate the exponentially large search space, we identify the most promising options in a "Pareto band" and search only within the band. We also devise an aggregate multi-resource multi-cluster metric to compare configurations within the band. Our evaluations with real-world video queries show promising results of being within 6% of optimal planning.

References

[1] Akamai's state of the internet report. https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q1-2017-state-of-the-internet-connectivity-report.pdf.

[2] Apache Calcite - a dynamic data management framework. http://calcite.incubator.apache.org. Accessed 04-27-2015.

[3] Apache Storm. https://storm.apache.org/.

[4] Avigilon. http://avigilon.com/products/.

[5] AXIS camera application platform. https://goo.gl/tqmBEy. Accessed 01-25-2016.

[6] Azure pricing. https://azure.microsoft.com/en-us/pricing/. Accessed 08-30-2017.

[7] bgslibrary. https://github.com/andrewssobral/bgslibrary. Accessed 08-23-2017.

[8] Genetec. https://www.genetec.com/.

[9] Gurobi Optimization. http://www.gurobi.com/.

[10] imagenet. www.image-net.org/challenges/LSVRC/. Accessed 09-19-2017.

[11] Introduction to SIFT (Scale-Invariant Feature Transform). http://docs.opencv.org/3.1.0/da/df5/tutorial_py_sift_intro.html.

[12] Introduction to SURF (Speeded-Up Robust Features). http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html.

[13] Meet the intelligent edge. https://www.microsoft.com/en-us/internet-of-things/intelligentedge.

[14] NYPD expands surveillance net to fight crime as well as terrorism. https://goo.gl/Y9OKh0. Accessed 01-25-2016.

[15] Top video surveillance trends for 2016. https://technology.ihs.com/api/binary/572252/.

[16] Visual object tracking challenge 2016. http://www.votchallenge.net/vot2016. Accessed 08-23-2017.

[17] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, et al. The design of the borealis stream processing engine. In *CIDR*, volume 5, pages 277–289, 2005.

[18] S. Agarwal, B. Mozafari, A. Panda, M. H., S. Madden, and I. Stoica. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In *ACM EuroSys*, 2013.

[19] Apache Hadoop NextGen MapReduce (YARN). Retrieved 9/24/2013, URL: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.

[20] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia. Spark SQL: Relational data processing in Spark. In *SIGMOD*, 2015.

[21] R. K. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb. Simplifying cyber foraging for mobile devices. In *Proceedings of the 5th International Conference on Mobile Systems, Applications and Services*, MobiSys '07, pages 272–285, New York, NY, USA, 2007. ACM.

[22] R. K. Balan, M. Satyanarayanan, S. Y. Park, and T. Okoshi. Tactics-based remote execution for mobile computing. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, MobiSys '03, pages 273–286, New York, NY, USA, 2003. ACM.

[23] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 242–253. ACM, 2011.

[24] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou. Apollo: Scalable and coordinated scheduling for cloud-scale computing. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 285–300, Broomfield, CO, 2014. USENIX Association.

[25] B. Chandramouli, J. Goldstein, M. Barnett, R. DeLine, D. Fisher, J. Wernsing, and D. Rob. Trill: A High-Performance Incremental Query Processor for Diverse Analytics. In *USENIX NSDI*, 2014.

[26] B. Chandramouli, S. Nath, and W. Zhou. Supporting distributed feed-following apps over edge devices. 2013.

[27] G. Cormode, M. N. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1-3):1–294, 2012.

[28] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[29] A. Deshpande and J. M. Hellerstein. Decoupled query optimization for federated database systems. In *IEEE International Conference on Data Engineering*, 2002.

[30] M. N. Garofalakis and Y. E. Ioannidis. Parallel query scheduling and optimization with time-and space-shared resources. 1997.

[31] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *USENIX NSDI*, 2011.

[32] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic. Adaptive offloading for pervasive computing. *IEEE Pervasive Computing*, 3(3):66–73, 2004.

[33] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, 2016.

[34] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, 2011.

[35] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica. Chameleon: Scalable adaptation of video analytics. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, 2018.

[36] T. Karnagel, D. Habich, and W. Lehner. Adaptive work placement for query processing on heterogeneous computing resources. In *VLDB*, 2017.

[37] K. Kloudas, M. Mamede, N. Preguica, and R. Rodrigues. Pixida: Optimizing Data Parallel Jobs in Wide-Area Data Analytics . In *VLDB*, 2015.

[38] S. Krishnamurthy, C. Wu, and M. Franklin. On-the-fly sharing for streamed aggregation. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 623–634. ACM, 2006.

[39] A. Kumar and et al. Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing. In *SIGCOMM*, 2015.

[40] Z. Li, C. Wang, and R. Xu. Task allocation for distributed multimedia processing on wirelessly networked handheld devices. In *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, pages 6–pp. IEEE, 2001.

[41] R. LiKamWa and L. Zhong. Starfish: Efficient concurrency support for computer vision applications. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 213–226. ACM, 2015.

[42] F. Loewenherz, V. Bahl, and Y. Wang. Video analytics towards vision zero. In *ITE Journal*, 2017.

[43] Y. Lu, A. Chowdhery, and S. Kandula. Optasia: A relational platform for efficient large-scale video analytics. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, pages 57–70. ACM, 2016.

[44] M. Luckie, A. Dhamdhere, D. Clark, B. Huffaker, and K. Claffy. Challenges in inferring internet interdomain congestion. In *IMC*, 2014.

[45] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, resource management, and approximation in a data stream management system. CIDR, 2003.

[46] B. Patt-Shamir and D. Rawitz. Vector bin packing with multiple-choice. *CoRR*, abs/0910.5599, 2009.

[47] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-aware operator placement for stream-processing systems. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 49–49. IEEE, 2006.

[48] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica. Faircloud: Sharing the network in cloud computing. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 187–198, New York, NY, USA, 2012. ACM.

[49] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica. Low latency geo-distributed data analytics. *ACM SIGCOMM Computer Communication Review*, 45(4):421–434, 2015.

[50] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan. Odessa: enabling interactive perception applications on mobile devices. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 43–56. ACM, 2011.

[51] J. Redmon and A. Farhadi. YOLO9000: Better, Faster, Stronger. In *CVPR*, 2017.

[52] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In *ACM SIGMOD Record*, volume 29, pages 249–260. ACM, 2000.

[53] M. Satyanarayanan, V. Bahl, R. Careres, and N. Davies. The Case for VM-based Cloudlets in Mobile Computing. In *IEEE Computer*, 2009.

[54] T. Sellis and S. Ghosh. On the multiple-query optimization problem. *IEEE Transactions on Knowledge and Data Engineering*, 2(2):262–266, 1990.

[55] A. Silberstein and J. Yang. Many-to-many aggregation for sensor networks. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 986–995. IEEE, 2007.

[56] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[57] U. Srivastava, K. Munagala, and J. Widom. Operator placement for in-network stream query processing. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 250–258. ACM, 2005.

[58] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[59] N. Trigoni, Y. Yao, A. Demers, J. Gehrke, and R. Rajaraman. Multi-query optimization for sensor networks. In *International Conference on Distributed Computing in Sensor Systems*, pages 307–321. Springer, 2005.

[60] C. J. Van Rijsbergen. Information Retrieval. *Butterworth, 2nd edition*, 1979.

[61] H. Varian. Equity, envy, and efficiency. In *Journal of Economic Theory*, volume 9, pages 63–91, 1974.

[62] D. Vasisht, Z. Kapetanovic, J. Won, X. Jin, R. Chandra, S. Sinha, and A. Kapoor. Farmbeats: An iot platform for data-driven agriculture. USENIX, March 2017.

[63] S. Venkataraman, A. Panda, G. Ananthanarayanan, M. J. Franklin, and I. Stoica. The power of choice in data-aware cluster scheduling. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 301–316. USENIX Association, 2014.

[64] A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.

[65] R. Viswanathan, G. Ananthanarayanan, and A. Akella. Clarinet: Wan-aware optimization for analytics queries. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 435–450. USENIX Association, 2016.

[66] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese. Global analytics in the face of bandwidth and regulatory constraints. In *NSDI*, pages 323–336, 2015.

[67] S. Xiang, H. B. Lim, K.-L. Tan, and Y. Zhou. Two-tier multiple query optimization for sensor networks. In *Distributed Computing Systems, 2007. ICDCS'07. 27th International Conference on*, pages 39–39. IEEE, 2007.

[68] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized Streams: Fault-Tolerant Streaming Computation at Scale. In *ACM SOSP*, 2013.

[69] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, V. Bahl, and M. J. Freedman. Live Video Analytics at Scale with Approximate and Delay-Tolerant Processing. In *NSDI*, 2017.

[70] T. Zhang, A. Chowdhery, P. Bahl, K. Jamieson, and S. Banerjee. The design and implementation of a wireless video surveillance system. In *ACM MOBICOM*, 2015.

[71] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In *Proceedings of the Twenty-eighth Annual Conference on Neural Information Processing Systems (NIPS)*, 2014.