Optasia: A Relational Platform for Efficient Large-Scale Video Analytics

Yao Lu, Aakanksha Chowdhery, Srikanth Kandula

Microsoft *

Abstract

Camera deployments are ubiquitous, but existing methods to analyze video feeds do not scale and are error-prone. We describe Optasia, a dataflow system that employs relational query optimization to efficiently process queries on video feeds from many cameras. Key gains of Optasia result from modularizing vision pipelines in such a manner that relational query optimization can be applied. Specifically, Optasia can (i) de-duplicate the work of common modules, (ii) auto-parallelize the query plans based on the video input size, number of cameras and operation complexity, (iii) offers chunk-level parallelism that allows multiple tasks to process the feed of a single camera. Evaluation on traffic videos from a large city on complex vision queries shows high accuracy with many fold improvements in query completion time and resource usage relative to existing systems.

Categories and Subject Descriptors D.4.7 [*Operating systems*]: Distributed systems; I.2.10 [*Artificial intelligence*]: Vision and scene understanding - Video analysis

Keywords Video analytics, parallel systems, query optimization, dataflow engines, relational languages

1. Introduction

Recently, there has been a rapid growth in camera deployments. Many cities have cameras on traffic lights and street corners [43]; police departments use mounted cameras on cars and body-cams on personnel [19]. IHS Research [14] forecasts the video surveillance market to grow over the next four years to $2.7 billion. A key enabler for this change is the lower cost of high quality cameras and data storage.

Automatic analysis of surveillance videos removes the human-in-the-loop and has the potential to be more accurate, faster and more comprehensive. Our use cases fall into two main bins– (i) *real-time analysis* to detect anomalies such as security lapses or to maintain dashboards such as the number of cars on a highway [68] and (ii) *longitudinal or post-facto analysis* that retroactively examines video streams to look for a certain person, a car, or a pattern [26, 49].

The state of the art in surveillance is custom closed solutions. Vendors such as Omnicast [58], ProVigil [60] and Avigilon [23] deploy and maintain the cameras. Often, the video is live streamed to an operations center for manual observation. This process is error-prone and expensive. Some vendors also provide video storage and analytics software and the larger customers (e.g. cities) have curated in-house systems [1]. However, such automated analysis is restricted to specific goals such as estimating the traffic congestion. Consequently, the *vision pipelines* are carefully hand-crafted with the engineers focusing on nitty gritty details such as how to parallelize, which order to execute the modules in etc. Existing dataflow systems such as Spark require similar hand-crafting of pipelines because they lack query optimization. Supporting ad-hoc queries or post facto analysis on stored video or scaling to a large number of cameras remain key open problems [67].

In this paper, we ask whether bringing together advances from two areas—machine vision and big data analytics systems, can lead to an efficient query answering system over many cameras.

A first challenge is to execute basic vision tasks on surveillance videos, such as detecting persons and tracking moving vehicles, with high precision. Surveillance videos have low resolution, low frame rate and varying light and weather conditions. More importantly, executing multiple analytic tasks on dense frames (many objects per frame) is computationally expensive. We build upon some state-of-the-art vision techniques to address these issues. Further, since surveillance cameras have a *fixed frame of reference*, we also use camera-specific information to improve accuracy and efficiency. We have built several vision *modules* such as classifying vehicles by color and type, re-identifying vehicles across cameras, tracking lane changes, identifying license plates etc.

* This work was done when the first and second authors were at Microsoft.

SoCC '16, October 05 - 07, 2016, Santa Clara, CA, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-4525-5/16/10. . . $15.00.
DOI: http://dx.doi.org/10.1145/2987550.2987564

We modularize each implementation so that the dataflow system can de-duplicate and parallelize the processing. Further details are in Section 3.1.

Next, to address the challenge of scaling to a rich set of ad-hoc queries and to many cameras, we cast the problem as an application of a relational parallel dataflow system. Our key contribution is to wrap the above-described vision modules inside some well-defined interfaces (processors, reducers and combiners [28, 72]) – this allows them to be expressed as a composition of the corresponding relational operators (select, project, aggregate, and cartesian product). End-users simply declare their queries over the modules in a modified form of SQL. Then, a cost-based query optimizer, built per the Cascades [40] framework, applies a large family of transformation rules to translate user queries into appropriate parallel plans [27] over the vision modules. Various standard query optimization improvements such as predicate push down (execute filters near input) and choosing appropriate join orders come to bear automatically [20]. Prior work in vision community has not leveraged relational operators or SQL to effectively run video queries. Further details are in Section 3.2.

The primary advantages of this combination are (i) ease-of-use for end-users; we will show that complex queries such as amber alerts and traffic dashboards can be declared within a few lines, (ii) decoupling of roles between end-users and the vision engineers; the vision engineers can ignore pipeline construction and need only focus on efficiency and accuracy of specific modules, and (iii) automatic generation of appropriate execution plans that among other things de-duplicate similar work across queries and parallelize appropriately; we will show examples where the resultant plans are much improved over those literally declared by the user query.

In hindsight, the case for a system such as Optasia, which casts vision queries as an application for big-data platforms, seems rather obvious given the possible gains. Along the way, we also discovered a few simple yet useful tricks. For instance, for some queries, neither camera-level nor frame-level parallelism is appropriate. Consider counting the traffic volume (# of vehicles/min/lane) from a highway video. The query requires context across frames to avoid duplicate counts and hence frame-level parallelism leads to an incorrect answer. However, camera-level parallelism leads to skew (if one camera processes a busy road portion) and slow response times, because a single task has to process all the frames from a camera. By observing that the context required is bounded to the duration for which vehicles remain in the frame of reference, Optasia breaks the feed from each camera into overlapping chunks of frames. This concept of *chunk-level* parallelism is important to combat skew and speed-up response times (see Table 7).

We have built an initial version of Optasia on top of Microsoft's Cosmos system [76]. Optasia supports several common vision modules (see Table 2) and we describe some exemplar user queries (see Section 3.2.2). We evaluate Optasia by analyzing the video feeds from tens of cameras from a highway monitoring company. We also use a variety of video feeds collected in and around the Microsoft campus (see Figure 1). Our results show that the combination of vision modules and dataflow reduces resource requirements by about 3×; details are in Section 5.

To summarize, the novel contributions of Optasia are:

- Fast and accurate modularized implementation of several vision modules needed in surveillance scenarios.
- A unified and customizable dataflow framework that computes optimal parallel query plans given any number of end-user queries for execution on a cluster. Our modularization of vision tasks makes explicit their analogy with relational operators which in turn allows relational query optimization to come to bear.
- A chunk-level parallelism technique that allow queries to keep context for bounded time across frames.
- Implementation and initial results.

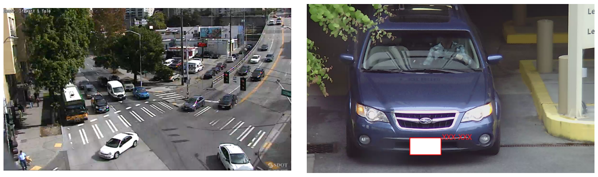
Much work remains; in particular, Optasia will benefit from more principled approaches to privacy (such as differential privacy or taint tracking) and improved video stores (compression, careful index generation). Nevertheless, we believe that Optasia targets a rich space of potential customers– customers that have a server farm, or can upload videos to a secure cloud provider [4, 55], can use Optasia today to benefit from fast, accurate, scalable, and customizable analysis of their videos.

2. Primer on video surveillance analytics

2.1 Example surveillance use-cases

Analytics on intersections and roadways: Surveillance cameras are installed on major intersections and highways in many cities. One use case is to understand the flow of vehicles and pedestrians to improve traffic planning (e.g., determine the hours for HOV or pay-to-use lanes, estimate the need for pedestrian or bicycle lanes etc.). Another use-case is to detect traffic congestion, violations and accidents in realtime [26, 67, 69]. A third use-case is to search over time for vehicles and license plates associated with an amber alert [64]. In parking structures, surveillance video can help ensure security of parked vehicles, detect squatters or other anomalies and serve as evidence for accidents. Figure 1 shows an example from a Seattle intersection and a Microsoft garage.

Enterprises deploy cameras primarily for insurance (evidence) purposes. Some link cameras with the facilities department to, for example, react faster to spills or to readily access what is going on in response to a fire alarm. **Retail** use-cases revolve around data-driven decisions; it is common to use videos to determine which hours to staff more and to optimally position products.



(a) Intersection. (b) Parking garage.

Figure 1: Examples of traffic surveillance video feeds.

Scenario	#cam	Feed type	Supp.	Ingest rate	Storage
Highway	1,000	mpeg2, 352p/15fps	50%	192Mbps	28 TB
City	1,000	h.264, 360p/24fps	80%	140Mbps	51 TB
Enterprise	100	h.264, 720p/30fps	80%	48Mbps	18 TB

Table 1: Back-of-the-envelope estimation of the problem size in different surveillance settings: ingest rate in Mbps and storage size for a week of surveillance videos. Supp. denotes the typical average suppression rates achievable in each setting.

Table 1 lists some back-of-the-envelope numbers for a video surveillance system. The data volume is affected by the application scenario, type of the video feed (e.g. frame rate, resolution, and video format), and camera specifics (some suppress frames early for example by only emitting frames that have motion [11]). The table lists the data ingest rate and storage required per week. It is easy to see that the ingest rate is rather small (relative to Youtube [17] or Netflix [8] which use up to 2Mbps for HD quality); however, since the video is continuously acquired, executing complex analysis on the stored video is a big-data problem.

2.2 Requirements for a surveillance system

The above use-cases lead to the following requirements.

- **Precision and recall:** Anomalies should be detected with a small number of false positives and true negatives; counts of objects should be approximately correct.
- **Timeliness:** Quick response time is the primary requirement to both realtime and post-facto use-cases.
- **Resource efficiency:** We are interested primarily in scaling out to a large number of cameras and analyses (queries) with few machines, that is, frames/sec/$ and queries/sec/$.
- **Customizability:** The video storage and analyses system should readily accept new queries- both realtime and post-facto. Further, the best execution plan for a given set of queries, in terms of resource efficiency and timeliness, may change when new queries are added.
- **Probabilistic/ confidence estimates:** Vision algorithms are probabilistic (e.g., what is the license plate? what is the vehicle type?). A surveillance system should have probability as a first class entity to simplify the decision making of end users.

2.3 State-of-the-art in surveillance systems

In the early 2000s, the US government funded a Video Surveillance and Monitoring (VSAM) [5] program that led to several real-world deployments and research [62]. The IBM Smart Surveillance System (S3) [68] was one of the most notable. They had a pilot deployment in Chicago, developed middleware that monitors scenes, stored video in a SQL database, and provided a web interface that reported both real-time alerts and allowed for long-term pattern mining. While Optasia has the same overall goals, our key contributions (improved vision modules and casting vision queries into a distributed dataflow system with query optimization and scale-out) substantially improve upon S3. In particular, each vision query in S3 ran with its own independent hand-optimized pipeline.

In the commercial space, as already mentioned, several vendors support video surveillance deployments that have thousands of cameras. However, these systems rarely use automated analyses. Nor do they offer dataflow pipelines for queries. Typically, contracts consider availability (uptime) of video feeds and longevity of storage. Even the many simple use-cases listed above are outside of these contractual agreements. Hence, they remain expensive and cumbersome.

2.4 Challenges

Realizing the requirements in §2.2 for the use-cases in §2.1 leads to the following two challenges.

- **Surveillance video ≠ images**: Most vision research uses images as input. These images are often high-resolution, collected in ideal light conditions and are from decent benchmarks [6, 7, 37]. In contrast, typical inputs to a surveillance system are low resolution videos [13]. The lighting conditions can vary continuously; there are often multiple objects per frame and occlusions [9]. In this sense, surveillance videos differ substantially from movies or talks. However, surveillance cameras are mostly fixed and the data is available continuously. These aspects allow some optimizations as we will see shortly.

- **Vision queries ≠ SQL queries (e.g., TPC-DS [12])**: Decades of work in relational algebra have codified design patterns to express a data analysis query in a manner that can be automatically optimized. Recent work also considers the automatic generation of parallel plans [22, 76]. However, a typical vision query consists of multiple algorithmic modules, such as cleaning the input and classifying the objects. Each module is a user-defined operator (UDO), which brings trouble to many query optimizers, as their semantic details are not specified. Further, even seemingly diverse queries such as traffic counting and amber alert can have similar components such as background subtraction and extracting HOG features. Ideally, a query optimizer should avoid duplication of work. Hence, we are interested in a system that optimizes the execution of multiple queries and adapts the execution gracefully when new queries or more data arrives.

3. Optasia Design

3.1 Vision modules for surveillance

We develop several vision modules to support popular surveillance use-cases. In each case, we emphasize our innovations that (i) improve the accuracy and/or (ii) lower the computational cost on input videos collected from deployments in the wild. We begin with a simple module.

3.1.1 Automatic license plate recognition (LPR)

The license plate recognition module takes as input one or more images of vehicles passing through a *gateway* and outputs a set of possible license plates. The *gateway* can be a virtual line on a roadway or inside a garage.

Our goal here is to build a license plate recognition module over video that requires no additional hardware (such as magnetic coils, flash lights or special-band light [46]). Furthermore, the video resolution is whatever is available from the wild. We would like to extract for each frame the top few likely license plate numbers and the confidence associated with each number. The following pipeline is applied:

- *License plate localization* looks for a bounding box around the likely location of the license plate. We move a sliding window over the video frame and apply a linear SVM classifier [32, 52] to estimate how likely each window is to have a license plate; the windows are sized in a camera-specific manner. The output is a set of potential bounding boxes per frame.
- *Binarization and character segmentation* converts each bounding box into binary and cuts out individual characters of the license, if any. We use standard image processing techniques here such as adaptive image thresholding [24], RANSAC baseline detection [39], and blob and character detection.
- *OCR:* We apply a pre-trained random forest classifier [25] to identify each character; we search for the characters 0–9, A–Z, and '-'. This yields, for each character in the image, several predicted values with soft probabilities for each value. The overall license plate is a combination of these predictions with confidence equal to their joint probability.
- *Post-processing:* Since license plates have some common formats (e.g. three numerals followed by three characters for plates in Washington state predating 2011), we use a pre-defined rule database to eliminate predictions that are unlikely to be valid license plates.

We acknowledge that the LPR module requires a certain amount of resolution to be applicable. For example, we detect almost no license plate from the videos in Figure 1(a) but can find almost every license plate from the videos in Figure 1(b).

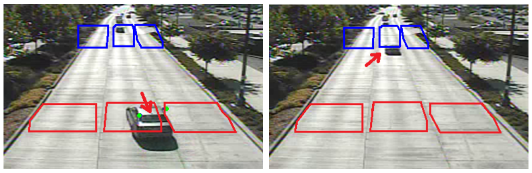


Figure 2: Step-by-step process of mapping traffic flow. Left: a vehicle entering the entry box. Right: a vehicle entering exit box.

Qualitatively, we outperform existing LPR softwares due to the following reasons. (1) We leverage the exemplar SVM [52] for license plate localization, while prior work [2] applies keypoint matching, which is less accurate. (2) We train a different OCR model per state to account for the differences in characters across states; the baseline approach has a single OCR model which is less accurate.

3.1.2 Real-time traffic flow mapping

On highways and at intersections, understanding the traffic flow has a variety of use-cases as described in §2.1, including planning restricted-use lanes, speed limits, traffic signs and police deployment. Hence, there has been much interest in modeling vehicular traffic flow [34, 42, 53, 70].

The most widely used method, however, is to deploy a set of cables ("pneumatic road tubes") across the roadway; this enables counting the number of vehicles that cross the coils and their velocity [15, 54, 56]. Such counts are typically not available in real-time. Further, the cables cannot capture information that is visible to the human eye (vehicle types, aggressive driving, vehicle origin-destination or how many turn right etc.).

Our goal here is to develop a module that extracts rich information about traffic flow from a video feed. Roadway surveillance cameras are typically mounted on towers or cross-beams; we use their fixed viewpoint to place labeled entrance and exit boxes on the roadway. An example is shown in Figure 2. Such annotation simplifies our traffic flow pipeline:

- Using a keypoint detection algorithm [65], we identify and track a vehicle that passes through the entrance box based on its keypoints [10, 51].
- If (and when) the keypoints cross the exit box, we generate a *traffic flow record* stating the names of the entrance and the exit box, the corresponding timestamps, and an estimate of the vehicle velocity.
- These records are processed by our dataflow engine (§3.2) into real-time estimates of traffic flow or can be appended to a persistent store for later use.

Note that the above logic can simultaneously track the traffic flow between multiple entrance and exit boxes. In fact, we can compute a 3x3 matrix of traffic flow between each pair of entrance and exit boxes shown in Figure 2; the matrix denotes volume in each lane and how often traffic changes lanes. Qualitatively, using keypoints to track objects is not new; we cite the following relevant prior work [65]. However, to the best of our knowledge applying these ideas in the context of real-time traffic flow is novel.

3.1.3 Vehicle type & color recognition

Building on the above pipeline, we do the following to identify the type and color of each vehicle.

- Once a vehicle is detected as above, we obtain an image patch for the vehicle by segmenting the image (see §3.1.5).
- Given the image patch of a vehicle, we extract various features including RGB histogram, and histogram of gradients (HOG) [33] and send them to a classifier.
- We use a linear SVM classifier trained with approximately 2K images belonging to each type and color. The output of the SVM is a class label (type or color) and the associated confidence. For vehicle type recognition we classify the vehicles into 'bike', 'sedan', 'van' , 'SUV', or 'truck'. For vehicle color recognition we classify the vehicles into 'white', 'black', 'silver', 'red', or 'others'. These labels were chosen based on their frequency of occurrence in the analyzed videos.

Our takeaway from this portion is that standard feature extraction and classifiers suffice to extract vehicle type and color from surveillance video; they do not suffice for more complex tasks such as detecting vehicle make and model. We chose mature and light-weight features and classifiers (see Table 2 for details) and find that they yield reasonable results.

3.1.4 Object re-identification

The problem here is to identify an object that may be seen by different cameras. Potential applications include region-wise tracking of vehicles and humans.

At a high level, object reidentification involves (1) learning an effective image and object representation over features and (2) learning a feature transform matrix between each pair of cameras [48]. We do the following:

- We learn a kernel matrix K for each camera pair by training on images of the same object that are captured at the two cameras. This matrix encodes how to "transform" an image from one camera's viewpoint to the viewpoint of the other camera.
- Then, the objects x seen at one camera are compared with objects z that appear at the other camera by computing a similarity score $d(x,z) = \phi(x) \cdot K \cdot \phi(z)^T$ where ϕ is a feature extraction function. Table 2 describes the features that we use for re-identification.

In practice, both x and z can contain multiple objects and hence the answer $d(x,z)$ could be interpreted as a pair-wise similarity matrix.

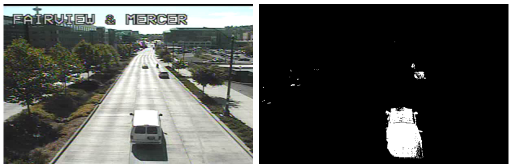


Figure 3: Background subtraction. Left: a vehicle entering the camera view. Right: binary mask indicating moving objects.

3.1.5 Background subtraction and segmentation

Background subtraction is a common practice; it reduces the redundancy in surveillance videos [36, 77, 78]. The general pipeline is below.

- Construct a model of the background (e.g., Mixture of Gaussians) based on pixels in the past frames.
- Use the model to identify moving pixels in each frame, and then update the model.

Relative to the other vision modules described thus far, background subtraction is lightweight and often executes first, as a pre-processor, in our analysis pipelines. Consider the example in Figure 3, we segment the images into portions that are needed for further analyses as follows:

- We group the moving pixels using a connected-component algorithm [41] and return each component as a segment.
- The above approach does not work well with occlusions and dense frames; it can group cars in adjacent lanes as one object for example. Hence, we use heuristics based on the fixed viewpoint of surveillance cameras (e.g. typical size of objects of interest, lane annotations etc.) and an exemplar SVM [52] to further break the segments.

3.1.6 Conclusion on vision pipelines and modules

Table 2 describes a partial list of the techniques used in our system. Our takeaway is that the described design lets us perform typical vision tasks with good accuracy and efficiency. We are unaware of a system that performs all of these tasks on surveillance videos. Furthermore, Optasia improves upon point solutions (e.g. OpenALPR [2] for license plate recognition) because it (i) uses state-of-the-art vision techniques, and (ii) combines them with heuristics based on the fixed viewpoint of surveillance cameras. We note however that some of our video datasets have insufficient resolution for some tasks (e.g. inferring vehicle make/model). We next describe how to efficiently support user queries that use these vision modules at scale.

3.2 A dataflow platform for vision queries

We build on top of the SCOPE [27] dataflow engine. Besides general SQL syntax, the dataflow engine offers some design patterns for user-defined operators: **extractors**, **processors**, **reducers** and **combiners**. We first describe how Optasia adopts these design patterns for vision modules. Next, we describe our query optimization over vision queries.

Module Name	Description	Involving Query
Feature Extraction - RGB Histogram	Extract RGB histogram feature given the image patch.	Amber Alert, Re-ID
Feature Extraction - HOG	Extract Histogram of Gradient feature given the image patch [33].	Amber Alert, Re-ID
Feature Extraction - Raw Pixels	Extract raw pixel feature given the image patch.	Amber Alert
Feature Extraction - PyramidSILTPHist	Extract Pyramid SILTP histogram feature [48] given the image patch.	Re-ID
Feature Extraction - PyramidHSVHist	Extract Pyramid HSV histogram feature [48] given the image patch.	Object Re-ID
Classifier/regressor - Linear SVM	Apply linear SVM classifier/regressor [38] on the feature vector.	Amber Alert, Re-ID
Classifier/regressor - Random Forest	Apply Random forest classifier/regressor [25] on the feature vector.	Amber Alert
Classifier/regressor - XQDA	Object matching algorithm used in [48].	Object Re-ID
Keypoint Extraction - Shi-Tomasi	Extract Shi-Tomasi keypoints in the given image region [65].	Traffic Violation
Keypoint Extraction - SIFT	Extract SIFT keypoints in the given image region [50].	Amber Alert, Re-ID
Tracker - KLT	Tracking keypoints using KLT tracker [51].	Traffic Violation
Tracker - CamShift	Tracking objects using CamShift tracker [30].	Traffic Violation
Segmentation - MOG	Generate Mixture of Gaussian background subtraction [47].	All
Segmentation - Binarization	Binarize license plate images.	Amber Alert

Table 2: A partial list of vision modules provided in our system.

3.2.1 Dataflow for Vision

Extractors ingest data from outside the system. We support ingesting data in different video formats. An extractor translates video into a timestamped group of rows. An example follows.

```
... ← EXTRACT CameraID, FrameID, Blob
FROM video.mp4
USING VideoExtractor();
```

The columns have both native types (ints, floats, strings) and blobs (images, matrices). We encode image columns in the JPEG format to reduce data size and IO costs. The dataflow engine instantiates as many extractor tasks as needed given the size of input. Extractor tasks run in parallel on different parts of the video input.

Processors are row manipulators. That is, they produce one or more output rows per input row. Several vision components are frame-local such as extracting various types of features (see Table 2), applying classifiers etc. A few examples follow. As with extractors, processors can be parallelized at a frame-level; Optasia chooses the degree-of-parallelism based on the amount of work done by the processor [20] and the available cluster resources.

```
... ← PROCESS ...
PRODUCE CameraID, FrameID, HOGFeatures
USING HOGFeatureGenerator();

... ← PROCESS ...
PRODUCE CameraID, FrameID, License, Confidence
USING LPRProcessor();
```

Reducers are operations over groups of rows that share some common aspects. Many vision components such as background subtraction (§3.1.5) and traffic flow (§3.1.2) use information across subsequent frames from the same camera. They are implemented using reducers.

Observe that naively, the degree-of-parallelism of a reducer is bounded by the number of cameras. Because, an algorithm maintains state per camera (e.g. which vehicles were in the previous frame), randomly distributing frames across tasks will lead to incorrect output. Furthermore, camera-level parallelism can lead to skew: tasks corresponding to cameras with busy views may have much more work than other tasks.

Optasia uses a novel trick that increases the degree of parallelism many fold and can combat skew. Our intuition is that the state maintained across frames has a bounded time horizon. For the traffic flow example: each vehicle stays in the camera's frame-of-view for only a limited period of time and hence, we can *chunk* the video into overlapping groups of frames. If vehicles transit the frame-of-view in δ frames, then chunk-n may have frames $[ns - \delta, ns + s]$. That is, the reducer processing chunk-n uses the first δ frames only to warm-up its internal state (e.g., assess the background for background subtraction or detect keypoints of vehicles that overlap entrance boxes); it then processes the remaining s frames. The number of the frames per chunk s and the amount of overlap δ are configuration variables. Note that δ is bounded above by an installation-specific constant: the camera orientation and minimum speed of vehicles in view limit the extent of relevant history. In practice, we find that some reducers use less than this upper-bound. Chunk size s is calculated by comparing the benefits from parallelization (smaller s implies more parallelism) with the overheads (a fraction $\frac{\delta}{s}$ of the overall work is wasted). The amount of available resources and the need for fast query completion may also impact choice of s. Observe that with chunking the available degree of parallelism is now limited only by the chunk size (s) and no longer limited by the number of cameras. An example follows (the net effect of chunking is shown in bold; it is an additional group-by column):

```
... ← REDUCE ...
PRODUCE CameraId, FrameId, VehicleCount
ON {CameraId, ChunkId}
USING TrafficFlowTrackingReducer();
```

Reducers translate to a partition-shuffle-aggregate. That is, the input is partitioned on the *group* and shuffled such that rows belonging to a group are on one machine. The number of reducers and partitions is picked, as before, per the amount of work to be done. Our underlying dataflow engine supports

```
Func: AmberAlert:
Input: search terms: vehicle type $v_t$, vehicle color $v_c$, license $l$
Output: matching {camera, timestamp}
State: Real-time tables for $LPR, $VehType and $VehColor

SELECT CameraID, FrameID, ($LPR.conf * $VehType.conf *
    $VehColor. conf) AS Confidence
FROM $LPR, $VehType, $VehColor
ON $LPR.{CamId,FrameId}=$VehType.{CamId,FrameId},
    $LPR.{CamId,FrameId}=$VehColor.{CamId,FrameId}
WHERE $LPR.licensePlate=$l$ ∧ $VehType.type=$v_t$ ∧
    $VehColor.color=$v_c$
```

Figure 4: User query 1: Amber Alert.

both hash partitioning and range partitioning to avoid data skew [21].

Combiners implement custom join operations; they take as input *two groups* of rows that share some common aspects. Optasia uses combiners for correspondence algorithms, such as object re-identification (§3.1.4). Recall that re-identification joins an incoming frame (its features to be precise) with a reference set and a kernel matrix that encodes the mapping between the two cameras. An example follows:

```
... ← COMBINE X, Kernel, Z USING ReIDCombiner()
ON X.CamId = Kernel.Cam1, Z.CamId = Kernel.Cam2
PRODUCE Cam1, Cam2, FrameID1, FrameID2, Score;
```

A combiner and other joins, can be implemented in a few different ways. If one of the inputs is small, it can be broadcast in its entirety and joined in place with each portion of the other input; else, either side is partitioned and shuffled on the join keys and each pair of partitions are joined in parallel. The dataflow engine automatically reasons about the various join implementations.

Notes: We note a few benefits from this design. First, wrapping a vision module in one of the above design patterns lets the query optimizer reason about semantics. For example, a pair of processors is commutative if the columns that one processor manipulates or creates are *pass-through* columns for the other processor. Second, this design allows a vision engineer to focus on efficiently implementing core functionality; they can ignore details about how to parallelize, which order to join etc. Further, we encourage vision modules to perform a single role and explicitly declare all configuration. Not doing so can prevent reuse. For example, consider a *black-box* implementation of the traffic counter pipeline that implements all of the functionality described in §3.1.2 as one reducer. Such a module would preclude reusing intermediate content generated after each of the steps in the traffic counter pipeline {background subtraction → segmentation → vehicle bounding box identification → feature extraction → classification} with another query that may for example be searching for red cars (§3.1.3). Finally, we ensure that the overhead from using more statements is negligible. Each operator is implemented as an iterator that pulls from its parent. The operators are chained in memory and data is written to disk only when needed such as for the input to a shuffle. The output of this part is that each vision task translates to a directed acyclic graph (DAG) of logical operations; the DAG is used as input by query optimizer as we will describe shortly.

```
Func: Traffic violation alert:
Input: Search terms: vehicle type $v_t$, vehicle speed $v_s$, illegal origin
    and destination boxes $o$, $d$
Output: Matching {Camera, Timestamp, VehicleImage}.
State: Real-time tables for traffic flow mapping Traf, VehType

SELECT CameraID, FrameID, VehImage
FROM Traf, VehType
ON Traf.{CameraID,FrameID}=VehType.{CameraID,FrameID}
WHERE VehType.vType=$v_t$ ∧ (Traf.vSpeed≥$v_s$ ∨ (Traf.vOri=$o$ ∧
    Traf.vDes=$d$))
```

Figure 5: User query 2: Traffic Violation.

```
Func: Re-ID: tracking a vehicle between two cameras:
Input: Search term: vehicle type vt
Output: Matching {camera1, timestamp1, camera2, timestamp2}.
State: Real-time tables for re-identification ReID, VehType{1,2}

SELECT cameraId1, frameId1, cameraId2, frameId2
FROM ReID, VehType1 as VT1, VehType2 as VT2
ON ReID.{camId1,frameId1}={VT1,VT2}.{camId,frameId},
WHERE VT1.vType=vt ∧ VT2.vType=vt;
```

Figure 6: User query 3: Re-identification.

3.2.2 Example user queries

To ground further discussion, we show three example scripts that mimic common queries to a video surveillance system. The complete data flow and user scripts can be found at http://yao.lu/Optasia.

User query 1: Amber alert
We consider the problem of amber alert– retrieving a vehicle of certain color, type, and license plate number. The user query is shown in Figure 4. Assume that vision engineers have written their modules in §3.1 using the dataflow in §3.2.1 and that the output of these modules is available as *system tables*: $LPR, $VehType, $VehColor corresponding to license plates, vehicle types and vehicle colors. The user's query shown here is one select statement that joins three tables. Optasia only materializes the system tables when needed by user queries.

User query 2: Traffic violation
We consider the problem of detecting traffic law violations– vehicles that are overspeeding, weaving between lanes, or making illegal turns. The user query is shown in Figure 5. It is a single select statement.

User query 3: Re-identification
We consider the problem of retrieving a vehicle of the same type across two different cameras. The user query is shown in Figure 6.

3.2.3 Optimizing vision queries

Beyond the ease of specifying queries, we point out a few aspects of the above design. First, the end-user only needs to know the schema of the system tables that have been made available by the vision engineers. As long as they maintain the schema, vision engineers can change their pipeline transparent to users.

Second, Optasia substantially optimizes the execution of these queries. By recognizing that the filters are local to each input, they are pushed ahead of the join. That is, only rows matching the filters are joined rather than filtering after the join. This feature, called predicate push down [40], is standard in SQL query optimization. Other more novel aspects of Optasia follow. (1) The system tables are materialized only on demand. That is, if no current query requires license plate recognition, the DAG of operations associated with that module do not execute. (2) Optasia exploits commonality between the various tables. For example, both `VehType` and `VehColor` require similar features from the raw video frames; and such features are computed only once. (3) When many queries run simultaneously, Optasia does even better. This is akin to multi query optimization [63] in database literature. The filters coalesce across different queries. For example, amber alerts for `red SUV` and `green sedan` can be pushed down on to the `VehColor` table as the filter `red ∨ green`. After join, the individual amber alerts can separate out the frames that they desire (e.g. red frames). (4) Finally, a key aspect is that Optasia performs the most expensive operations over video frames exactly once (i.e. de-duplication) irrespective of the number of queries that may use such system tables.

To a reader familiar with relational operators [61], we note that `PROCESS` is a user-defined select and/or a project, `REDUCE` is a user-defined group-by and/or an aggregation and `COMBINE` is a user-defined join. Consequently, expressing visual queries with this vocabulary allows the query optimizer to reuse optimization rules from the corresponding relational operators. We believe that this is crucial for the QO find good plans.

Method: Optasia achieves these advantages by treating all queries as if they were one large query for the purposes of optimization. However, during execution, the jobs corresponding to each query are only loosely coupled. As with other data-parallel frameworks [16, 21], Optasia stores the output of "tasks" in persistent storage; each task is a unit of execution that is idempotent and should finish within seconds. Optasia retries failing tasks. Faults in user-code will cause consistent failures and hence such queries will fail; queries with defect-free user code rarely fail in Optasia.

QO details: Here, we sketch how the dataflow optimizations mentioned above are realized in Optasia. We do not claim contribution for these details, they build upon a large body of work in relational query optimization [40] and in adapting QO for parallel plans with user-defined operations [76]. We include them here for completeness. Optasia's contribution lies in translating visual queries to a format that makes relational query optimization effective.

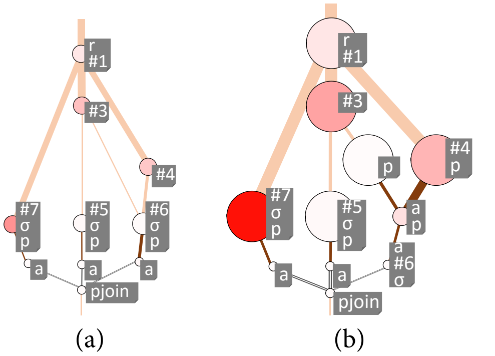


Figure 7: Dataflow and query Plans of Amber alert for (a) 1 GB input and (b) 100 GB video input. Note that 100 GB input automatically parallelizes the tasks to minimize the query plan cost and the query latency. Please refer to Figure 12 for legend.

The input is a collection of queries, each of which is a directed acyclic graph (DAGs) of logical operations. The desired output is an execution plan that can be translated to a set of loosely coupled jobs. This plan should have the above-mentioned properties including appropriate parallelization and de-duplication of work.

Our QO can be explained with two main constructs. A memo data structure remembers for each sub-expression (i.e., an operator and its descendants) the best possible plan and the cost of that plan. A large collection of transformation rules offer alternatives for sub-expressions. Examples of rules include predicate push-down:

$$\mathcal{E}_1 \to \mathtt{S} \to \mathtt{Filter} \to \mathcal{E}_2 \Longleftrightarrow \mathcal{E}_1 \to \mathtt{Filter} \to \mathtt{S} \to \mathcal{E}_2.$$

Transformations may or may not be useful; for example, which of the above choices is better depends on the relative costs of executing `Filter` and `S` and their selectivity on input. Hence, we uses data statistics to determine the costs of various alternatives. The lowest cost plan is picked. Here, cost is measured in terms of the completion time of the queries given available cluster resources. The memo also allows de-duplication of common sub-expressions across queries. By applying these transformation rules till fixed point, Optasia searches for an efficient plan for all the queries.

To speed-up the search, we defer a few aspects such as the choice of appropriate degree-of-parallelism and avoiding re-partitions till after a good *logical* plan is discovered. Given a logical plan, the QO costs a variety of serial and parallel implementations of sub-expressions (e.g., 20 partitions on column `X`) and picks the best parallel plan [75].

Stepping back, we highlight with examples two aspects of the query optimization that we found useful for vision queries. First, Optasia adapts plans with varying input size. Simply changing the degree of parallelism (DOP) does not suffice. When plans transition from serial (DOP = 1) to parallel, corresponding partition-shuffle-aggregates have to be added and join implementations change (e.g. from broadcast join to pair-join). Figure 7 illustrates the plan for amber-alerts (Figure 4) at two different input sizes. Next, Optasia automatically de-duplicates common vision portions of seemingly unrelated user queries. We illustrate this in Figure 12 when different user queries described above run together. We defer further discussion to §5.2.

4. Optasia System

Data acquisition: To evaluate Optasia on realistic inputs, we collected video data in two ways. (1) We collected high-resolution video data ourselves, in and around Microsoft campus, using the IP surveillance camera ACTi B21 with 1920x1080 resolution and 12x zoom. We collected video at the entrances to a few parking garages (from the windows of an adjacent building) as well as curb-side videos. (2) We also gathered publicly available video traces from the Washington State Department of Transportation (WSDOT). These are typically low res videos (352x258 resolution, 15FPS) from cameras mounted on crossposts along Washington state highways and at traffic intersections.

Core vision modules: We have built several vision modules, including all those described in Table 2. The modules are in C++ with the OpenCV library [10]. This codebase contains about 5K lines of code.

Dataflow modules: Each of the vision modules are mapped to a declarative dataflow system: SCOPE [27] using wrappers. These wrappers are about 700 lines of code in C#.

User queries and query optimization: The end user writes vision queries in the SCOPE language, an example of which is shown in Figure 4. We built several user queries including every one of the use-cases mentioned in §2. All queries are within a few tens of lines of code.

QO enhancements: We made a few enhancements to the core QO in SCOPE. We add an [`OPCOST` = ...] *hint* to the language, using which the user can specify the per-row cost of an operator. When used appropriately, with `PROCESS`, `REDUCE` and `COMBINE`, the hint allows the QO to choose appropriate parallelism (e.g. partition size). A similar [`ROWCOUNT` =] hint lets users specify the selectivity of operators (e.g. $\frac{\text{output}}{\text{input}}$). Together, these hints let the QO place inexpensive operators or those with high selectivity earlier in the plan. Appropriate values of operator cost and selectivity can be identified from past executions of *recurring jobs* [20]; doing so for new jobs remains future work.

Cluster: We built our system on top of Microsoft's Cosmos system [27], a large shared production cluster. For each case, we report performance and accuracy with Optasia.

Streaming: While the execution plans output by Optasia can also be used in a stream engine such as Trill [29], we have thus far only used them in the *batch* mode. When new data arrives online, the plans can be periodically re-executed say every minute with the outputs shared through memory. This is the so-called *mini-batch* model [74]. Applying Optasia to a distributed stream engine, especially one that scales beyond the total memory size of the cluster, is a key area of future work.

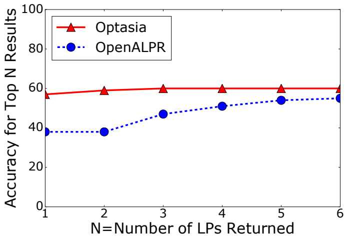


Figure 8: LPR Accuracy for Top N results.

Method	0 miss	≤ 1 miss	≤ 2 miss	rate (fps)
Optasia	0.57	0.75	0.82	4.8
OpenALPR	0.38	0.61	0.67	3.2

Table 3: LPR Evaluation.

5. Evaluation

5.1 Microbenchmarks of vision modules

5.1.1 License plate recognition

Methodology: The dataset for this evaluation is a day-long video of the cars exiting a Microsoft campus garage. The video is pre-processed using background subtraction to prune frames that have no cars. We draw a random sample of 1000 images from the remaining frames and annotate the license plate area manually to train the localization module. Furthermore, we annotate the license plate characters manually in 200 images to train the optical character recognition module. We use a test set of 200 *different* images to evaluate the License Plate Recognition module, end-to-end.

We benchmark our module against state-of-the-art OpenALPR [2], an open source Automatic License Plate Recognition library. Two metrics are used in the comparison: (i) *accuracy*, which measures the probability that the top N results contain the ground truth answer, and (ii) *maximum frame ingestion rate*, which is based on the processing time per frame. Both our module and OpenALPR run single threaded, and the average ingestion rate over a batch of video frames is reported.

Results: Figure 8 shows that accuracy (the probability that the true license plate is output) increases with N (the size of answers returned ordered by confidence); our method achieves reasonable results with only one answer. Table 3 demonstrates the quality of the highest likelihood answer that is output by the various schemes in terms of the number of wrong characters. The table shows that our LPR module (i) processes frames roughly 1.5× faster than the state-of-the-art license plate recognition software and (ii) achieves better accuracy on both top-1 and top-N answers.

	Seq1	Seq2	Seq3	Seq4	Avg	rate(fps)
Optasia	0.87	0.88	0.88	0.89	0.88	77
Baseline	0.46	0.40	0.31	0.58	0.44	42

Table 4: Vehicle counting accuracy and efficiency on four video sequences.

	Bike	Sedan	SUV	Truck	Van
Optasia	1.00	0.92	0.34	0.70	0.65
Baseline	0.01	0.67	0.17	0.05	0.10

Table 5: Car type classification accuracy. We compare with a simple guess according to the class distribution as baseline.

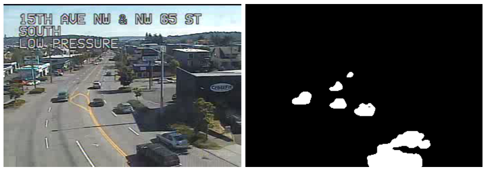


Figure 9: Failure case for blob detection.

5.1.2 Real-time traffic flow mapping

Methodology: The dataset for this evaluation is 10 minute segments from WSDOT [3]; we picked cameras in the city of Seattle on both highways and surface roads. The goal is to count the vehicles in each lane.

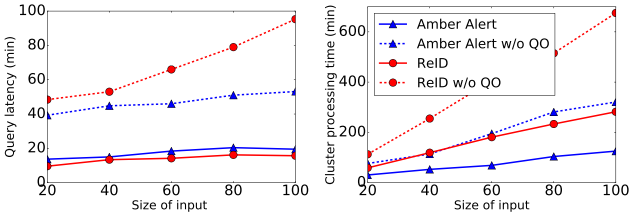
We compare against an open-source module [15], which does background subtraction and tracks blobs in the video. We measure the processing speed for each frame and the accuracy of the traffic volume in each lane.

Results: Table 4 shows that Optasia achieves an accuracy of 85–90% on four different video segments, while the accuracy of the car blob detection module is less than 60%. The baseline method detects blobs of moving objects and often fails when different vehicles occlude with each other, as shown in Figure 9. Unlike this approach, our proposed method is based on keypoints and leverages per-camera annotation (entry and exit boxes in each lane) to protect against such shortcomings. We also see that our approach is less computationally complex leading to a 1.8× higher frame processing rate compared to the baseline.

5.1.3 Classification of vehicles

Methodology: The dataset for this evaluation is a one hour video of the intersection of Fairview avenue and Mercer street available from WSDOT [3]. We apply the above discussed traffic flow module to segment this video into per-vehicle patches. Our goal here is to classify these patches into types and colors; that is, assign to each image the labels listed in §3.1.3. We compare against a baseline that guesses the class for each image with probability equalling the likelihood of that class. [1]

Results: Table 5 shows that Optasia achieves different accuracy levels per class; across all classes Optasia is much better than random guesses. The relatively lower accuracy for the SUV class is because SUVs are routinely confused with sedans on the low-resolution videos in the dataset; the two classes have a similar size especially with "cross-overs". Overall, we believe that coarse granular categorization of vehicles is possible with the techniques built into Optasia.



(a) (b)

Figure 10: Query Optimization reduces the query completion time significantly for both amber alert and Re-ID (a) as the number of input videos increases for each query. Further, query optimization ensures the most efficient cluster resource utilization in terms of processing time (b).

	1 GB input	100 GB input
Average Task Duration	18.3 sec	38.6 sec
Cluster Computing Time	37.78 min	4101.75 min
Intermediate data size	1.95 GB	188.95 GB
Cross-rack Network IO	8.9%	8.9%

Table 6: Query optimization ensures efficient resource usage as the input video size scales from 1 GB to 100 GB for Amber alert with LPR query.

5.2 Optimizing dataflow

Methodology: Over the video dataset from a Microsoft campus garage, we execute two end-to-end user queries: amber alert and car re-identification across 10-100 sets of input. For amber alert, each inputset contains a 90MB video from one camera, while for re-identification, each inputset contains videos from two cameras. All the videos are 1 minute in length. We experiment by running each amber alert and car re-id query independently as well as a group of (different) amber alert queries at one time on the input video set. Recall that an amber alert consists of a triple of (partial) license plate information, vehicle type and color. Further, for car re-identification, we first filter by vehicle type, and then use re-identification over the set of matching frames.

Additionally, on a dataset of videos available from Seattle WSDOT website, we execute two end-to-end user queries: amber alert, and traffic violations across 50 sets of input. The amber-alert query is similar to above, except it does not have license plate recognition; while for traffic violations, we measure the weaving of cars in the traffic flow from the leftmost lane to the rightmost lane.

We compare Optasia against a version of Optasia without query optimization. That is, the queries expressed by the end-user are run literally by the system. We measure the completion time of the query as well as the total resource usage across all queries (measured in terms of compute hours on the cluster). We repeat the experiment with different sizes of input to examine how Optasia scales. Besides, for amber alert, we vary the size of the query set (number of amber alert

[1] Uniformly random guesses for the class were less accurate.

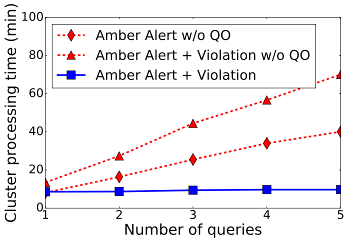


Figure 11: As the number of queries scale, query optimization ensures that the cluster processing time for both sets of queries stays constant by using auto-parallelization and de-duplication.

# of chunks	Query latency (in min)	Cluster Processing Time (in min)
1	16.1	20.2
3	7.6	23.4
8	5.2	24.2
10	5.4	25.4

Table 7: For traffic violation query, chunking the video minimizes the query latency by exploiting higher degree of parallelism.

triples) from one to five to see how queries are affected by the optimizer.

Results: Figure 10 (a) plots the ratio of the completion time for Optasia with the version of Optasia that has no query optimization, for single queries on the garage feed. We see that, with query optimization, Optasia is roughly 3× faster. Further, the completion time of Optasia remains constant as dataset sizes increase, illustrating the fact that the QO sets the degree-of-parallelism correctly. The large gains arise from de-duplicating the work in the vision modules (e.g., generating HOG features etc.).

Further, Figure 10 (b) demonstrates the amount of cluster resources used by Optasia and the version of the Optasia that does not perform query optimization. We observe similar behavior to Figure 10 (a). The key difference is that the gap between the two lines in Figure 10 (b) measures the total-work-done by the query and is directly related to the size of the inputset; for small inputs the gap is lost in noise but at large inputs, the gap opens up quite a bit. On the other hand, the gap in Figure 10 (a) is query completion time; even a query that does more work can finish quickly because our production cluster, where these experiments were conducted, uses a work-conserving scheduler; that is, it offers queries more than their share of resources if some other group is not using their full share.

Next, we evaluate how Optasia scales with different sizes of videos from the garage feed. Figure 7 shows the query plans for amber alert with LPR for two input sizes: 1 GB and 100 GB. In Figure 7 (b), the larger circle sizes and darker circles illustrate that the degree of parallelism is set correctly; hence, as Table 6 shows, the query completion time is almost similar even for larger input.

Figure 11 compares the improvement in completion time due to QO while varying the number of queries on the WSDOT feed. We see that the improvements of Optasia increase when there are many similar queries; the value of the X axes here denotes the number of different queries of each type being executed simultaneously. Due to careful de-duplication of work, the completion time of Optasia is roughly constant as the number of queries increase; the latency is only proportional to the amount of video examined. In contrast, the version of Optasia without QO is unable to de-duplicate the work, leading to substantially worse completion time as the number of queries increase.

Figure 12 (a) and (b) show the query plans when the amber alert and re-identification queries are run individually, while Figure 12 (c) shows the query plan when the two queries are run simultaneously. QO ensures de-duplication of the common modules in (c) thereby minimizing the query latency and resource usage on the cluster.

It is of course possible to carefully handcraft these vision pipelines to achieve a similar result. The key aspect of Optasia, however, is that such de-duplication (and query optimization, in general) occurs automatically even for complex queries. Thus, Optasia can offer these performance improvements along with substantial ease-of-use and can naturally extend to future user queries and vision modules.

Table 7 shows the effectiveness of chunking the videos with overlap for traffic violation queries on the WSDOT feed. Query completion times improve by using more chunks and hence leveraging higher degree of parallelism on the cluster (more cluster processing time). The optimal number of chunks in this case is 8; breaking into more chunks is not advisable because gains from added parallelism are undone by the overhead in processing the overlapping frames. We believe chunking to be rather broadly applicable to scenarios that are otherwise limited to camera-level parallelism.

Overall, we conclude that Optasia's dataflow engine not only allows end-users to specify queries in simple SQL-like syntax but by employing a powerful query optimization engine offers (i) the ability to run *similar queries* with nearly zero additional cost, and (ii) automatically scales the execution plan appropriately with growing volume of datasets.

6. Related Work

To the best of our knowledge, Optasia uniquely shows how to execute sophisticated vision queries on top of a distributed dataflow system. Below, we review some prior work.

6.1 Video analytics systems

We already discussed notable systems such as the IBM Smart Surveillance System and start-ups in this space in §2. Automatic analyses of videos, including that collected from highways and intersections, has a rich literature; the following are excellent surveys of the latest in this space [26, 49, 67, 73]. Key differences for Optasia are its use of simple camera-specific annotation and state-of-the-art vision techniques such as exemplar SVMs.

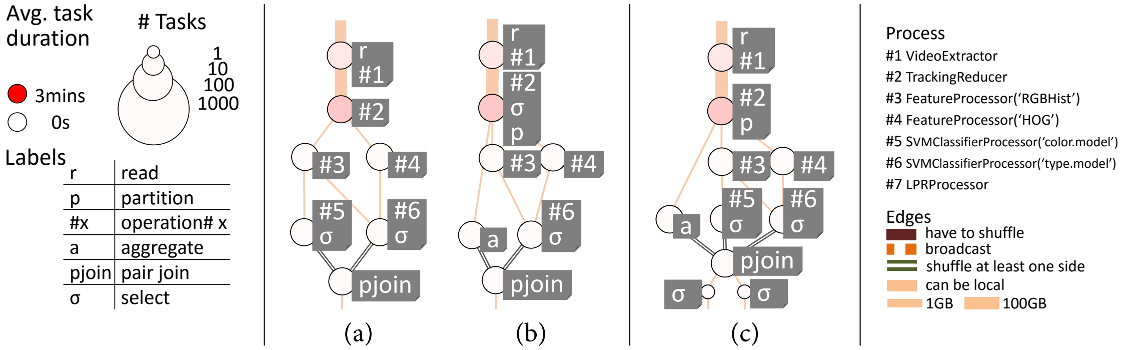


Figure 12: Query Plans of (a) Amber alert query, (b) Traffic Violation query, and (c) Amber Alert+Traffic Violation query. Note that the combined query plan in (c) deduplicates the common modules, thus minimizing the query plan cost and the query latency for both queries.

6.2 Dataflow systems

There has been significant recent interest in distributed dataflow systems and programming models, e.g., Dryad [44], Map-Reduce [31, 35, 45], Hive [66], Pig [57], Sawzall [59] and Spark [22]. At a high level, the recent work is characterized by a few key aspects: much larger scale as in clusters of tens of thousands of servers, higher degrees of parallelism, simpler fault-tolerance and consistency mechanisms, and stylistically different languages. The more recent frameworks adopt relational user-interfaces [22, 27, 66]. Most have a rule-based optimizer [22, 66]; except for SCOPE, which uses a Cascades-style [40] cost-based optimizer. The key distinction between the two is that the latter allows considering alternatives that need not be strictly better than the original plan; rather which alternative is better depends on properties of the code (e.g. the computational or memory cost of an operation) as well as data properties (e.g. the number of rows that pass through a filter).

Relative to these systems, Optasia offers a library of vision-specific modules built in a manner that lets users specify their queries in a SQL-like language.

For a more direct comparison, we show how to encode a certain simple vision pipeline in three systems, Optasia, Spark, and Sun Grid Engine (SGE). Please see http://yao.lu/Optasia. Sun Grid Engine (SGE) is a platform that manages and dispatches tasks that in turn can execute arbitrary user-defined scripts. Spark is a newer system that allows users to specify dataflow programs in a functional manner. Table 8 summarizes the key differences. As the table shows, SGE is indicative of earlier parallel systems in that it lacks adequate support for fault tolerance (one failing task may lead to a job failure) and is not backed by a distributed file system. Spark and other systems fix these challenges and offer more intuitive language interfaces; however, as the example shows, the burden of how exactly to parallelize a query (how many extractors, how many partitions etc.) is left to the user. Furthermore, the lack of a cost-based query optimizer is clearly felt in the inability to structure plans more efficiently than the literal queries that are specified by users. Optasia's key value-add is a method to modularize vision pipelines so that the result is close enough to relational algebra and then adapting existing query optimization techniques for these pipelines.

System	DFS	Task scheduler	Fault tolerance	Programming Language	Auto Parallelize	Query Optimizer
SGE	×	√	×	Symbolic	×	×
Spark	√	√	√	Symbolic	×	×
Optasia	√	√	√	Relational	√	√

Table 8: Comparison of Optasia to existing big-data processing systems Spark and Sun Grid Engine (SGE). DFS stands for distributed file system.

Note that our focus in this paper is on query answering systems over surveillance video. Optasia is orthogonal to the commendable recent work in training deep neural networks [71] on GPUs such as TensorFlow [18]. Our dataflow system focuses on efficiently executing video queries (that can use trained DNNs or other modules) on a cluster.

7. Conclusion

We present Optasia, a system that combines state-of-the-art techniques from the vision and data-parallel computing communities for a variety of surveillance applications. Optasia provides a SQL-like declarative language and substantially simplifies the job of end-users and vision engineers. Optasia adapts a cost based query optimizer (QO) to bridge the gap between end-user queries and low-level vision modules. The QO outputs good parallel execution plans, scaling appropriately as the data to be processed increases. Further, the QO also scales nicely across similar queries; it is able to structure the work of each query such that the overall work is not duplicated. Our evaluation on surveillance videos and experiments on a large production cluster shows that Optasia improves upon prior art by several times on accuracy and performance.

Acknowledgements

We acknowledge fruitful discussions with Paramvir Bahl, Matthai Philippose, Ganesh Ananthanarayanan, and Peter Bodik. This work has also benefited substantially from the work of the COSMOS and SCOPE production team at Microsoft.

References

[1] Operation virtual shield: a homeland security grid established in chicago. http://bit.ly/2bb33F7.

[2] OpenALPR: Open automatic license plate recognition library. https://github.com/openalpr/openalpr.

[3] Seattle department of transportation live traffic videos. http://web6.seattle.gov/travelers/.

[4] Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/.

[5] Defense advanced research projects agency information systems office's three-year program on video surveillance and monitoring technology. http://bit.ly/2bQIixs.

[6] ImageNET. http://www.image-net.org.

[7] Microsoft COCO: Common objects in context. http://mscoco.org.

[8] Netflix tech blog: High quality video encoding at scale. http://nflx.it/2bdMceJ.

[9] Earthcam live feeds from NYC. http://bit.ly/1SZgZQv.

[10] OpenCV. http://opencv.org/.

[11] Video surveillance storage: How much is enough? http://bit.ly/2bb3rDR.

[12] TPC-DS Benchmark. http://bit.ly/1J6uDap.

[13] Trafficland. http://www.trafficland.com.

[14] Supply of video management software remains fragmented. http://bit.ly/1TiDnVr.

[15] Vehicle counting based on blob detection. http://bit.ly/2bdjNWA.

[16] Hadoop YARN Project. http://bit.ly/1iS8xvP.

[17] Youtube: Video encoding settings. http://bit.ly/2bygQ8m.

[18] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous systems. http://www.tensorflow.org.

[19] ACLU. Police body-mounted cameras: With right policies in place, a win for all. http://bit.ly/1RBzI1i.

[20] S. Agarwal, S. Kandula, N. Burno, M.-C. Wu, I. Stoica, and J. Zhou. Re-optimizing data parallel computing. In *NSDI*, 2012.

[21] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris. Reining in the outliers in mapreduce clusters using mantri. In *OSDI*, 2010.

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

[23] Avigilon. Video surveillance solutions. http://bit.ly/21EIIr3.

[24] D. Bradley and G. Roth. Adaptive thresholding using the integral image. *Journal of graphics, gpu, and game tools*, 12(2):13–21, 2007.

[25] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[26] N. Buch, S. Velastin, and J. Orwell. A review of computer vision techniques for the analysis of urban traffic. *IEEE T. on Intelligent Transportation Systems*, 12(3):920–939, 2011. .

[27] R. Chaiken, B. Jenkins, P.-Å. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. Scope: easy and efficient parallel processing of massive data sets. *VLDB Endowment*, 1(2):1265–1276, 2008.

[28] C. Chambers, A. Raniwala, F. Perry, S. Adams, R. R. Henry, R. Bradshaw, and N. Weizenbaum. Flumejava: easy, efficient data-parallel pipelines. In *PLDI*, 2010.

[29] B. Chandramouli, J. Goldstein, M. Barnett, R. DeLine, D. Fisher, J. C. Platt, J. F. Terwilliger, and J. Wernsing. Trill: A high-performance incremental query processor for diverse analytics. *VLDB Endowment*, 8(4):401–412, 2014.

[30] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE T. on Pattern Analysis and Machine Intelligence*, 17(8):790–799, 1995.

[31] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. Mapreduce online. In *NSDI*, 2010.

[32] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. 2000.

[33] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR.*, 2005.

[34] K. Davidson. A flow travel time relationship for use in transportation planning. In *Australian Road Research Board Conference, 1966, Sydney*, 1966.

[35] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[36] A. Elgammal, D. Harwood, and L. Davis. Non-parametric model for background subtraction. In *ECCV*. 2000.

[37] M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge. *IJCV*, 88(2):303–338, 2010.

[38] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *JMLR*, 9:1871–1874, 2008.

[39] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24 (6):381–395, 1981.

[40] G. Graefe. The cascades framework for query optimization. *IEEE Data Eng. Bull.*, 18(3):19–29, 1995.

[41] R. M. Haralick and L. G. Shapiro. Image segmentation techniques. *Computer vision, graphics, and image processing*, 29(1):100–132, 1985.

[42] S. P. Hoogendoorn and P. H. Bovy. State-of-the-art of vehicular traffic flow modelling. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 215(4):283–303, 2001.

[43] IIHS. Communities using red light cameras and speed cameras. http://bit.ly/2bb3OOA.

[44] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *EuroSys*, 2007.

[45] D. Jiang, B. Ooi, L. Shi, and S. Wu. The performance of mapreduce: An in-depth study. *VLDB Endow.*, 3(1), 2010.

[46] J. Juang and Y.-C. Huang. *Intelligent Technologies and Engineering Systems*. 2013.

[47] P. KaewTraKulPong and R. Bowden. An improved adaptive background mixture model for real-time tracking with shadow detection. In *Video-based surveillance systems*, pages 135–144. Springer, 2002.

[48] S. Liao, Y. Hu, X. Zhu, and S. Z. Li. Person re-identification by local maximal occurrence representation and metric learning. In *CVPR*, 2015.

[49] S. Liu, J. Pu, Q. Luo, H. Qu, L. Ni, and R. Krishnan. Vait: A visual analytics system for metropolitan transportation. *IEEE T. on Intelligent Transportation Systems*, 2013.

[50] D. G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, 1999.

[51] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI*, 1981.

[52] T. Malisiewicz, A. Gupta, and A. A. Efros. Ensemble of exemplar-svms for object detection and beyond. In *ICCV*, 2011.

[53] A. D. May. *Traffic flow fundamentals*. 1990.

[54] P. McGowen and M. Sanderson. Accuracy of pneumatic road tube counters. In *Proceedings of the 2011 Western District Annual Meeting, Anchorage, AK, USA*, 2011.

[55] Microsoft. An Overview of Windows Azure. http://bit.ly/1Qo6yUg.

[56] L. E. Y. Mimbela and L. A. Klein. Summary of vehicle detection and surveillance technologies used in intelligent transportation systems. http://bit.ly/2bEZsgW.

[57] C. Olston et al. Pig Latin: A Not-So-Foreign Language for Data Processing. In *SIGMOD*, 2008.

[58] Omnicast. Video Management Software. http://bit.ly/2bIKnYK.

[59] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. Interpreting the data: Parallel analysis with sawzall. *Scientific Prog.*, 2003.

[60] Pro-Vigil. Video surveillance. http://pro-vigil.com/.

[61] R. Ramakrishnan and J. Gehrke. *Database management systems*. 2000.

[62] P. Remagnino. *Video-Based Surveillance Systems: Computer Vision and Distributed Processing*. 2001.

[63] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In *SIGMOD*, 2000.

[64] M. Satyanarayanan. Mobile computing: The next decade. *SIGMOBILE Mob. Comput. Commun. Rev.*, 2011.

[65] J. Shi and C. Tomasi. Good features to track. In *CVPR*, 1994.

[66] A. Thusoo et al. Hive: A Warehousing Solution Over A Map-Reduce Framework. *VLDB Endow.*, 2009.

[67] B. Tian, B. Morris, M. Tang, Y. Liu, Y. Yao, C. Gou, D. Shen, and S. Tang. Hierarchical and networked vehicle surveillance in its: A survey. *IEEE T. on Intelligent Transportation Systems*, 16(2):557–580, 2015.

[68] Y.-l. Tian, L. Brown, A. Hampapur, M. Lu, A. Senior, and C.-f. Shu. Ibm smart surveillance system (s3): Event based video surveillance system with an open and extensible framework. *Mach. Vision Appl.*, 19(5-6):315–327, 2008.

[69] H. Vceraraghavan, O. Masoud, and N. Papanikolopoulos. Vision-based monitoring of intersections. In *IEEE International Conference on Intelligent Transportation Systems*, pages 7–12, 2002. .

[70] X. Wang, X. Ma, and W. E. L. Grimson. Unsupervised activity perception in crowded and complicated scenes using hierarchical bayesian models. *IEEE Transactions on PAMI*, 31(3): 539–555, 2009.

[71] S. Yeung, O. Russakovsky, G. Mori, and L. Fei-Fei. End-to-end learning of action detection from frame glimpses in videos. In *CVPR*, 2016.

[72] Y. Yu, M. Isard, D. Fetterly, M. Budiu, U. Erlingsson, P. K. Gunda, and J. Currey. Dryadlinq: a system for general-purpose distributed data-parallel computing using a high-level language. In *OSDI*, 2008.

[73] G. Yuan, X. Zhang, Q. Yao, and K. Wang. Hierarchical and modular surveillance systems in its. *IEEE Intelligent Systems*, 26(5):10–15, 2011.

[74] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica. Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. In *HotCloud*, 2012.

[75] J. Zhou, N. Bruno, and W. Lin. Advanced partitioning techniques for massively distributed computation. In *SIGMOD*, 2012.

[76] J. Zhou, N. Bruno, M.-C. Wu, P.-A. Larson, R. Chaiken, and D. Shakib. Scope: parallel databases meet mapreduce. *VLDB Endowment*, 21(5):611–636, 2012.

[77] Z. Zivkovic. Improved adaptive gaussian mixture model for background subtraction. In *ICPR*, 2004.

[78] Z. Zivkovic and F. van der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern recognition letters*, 27(7):773–780, 2006.