A Relational Platform for Efficient Large-Scale Video Analytics

Yao Lu, Aakanksha Chowdhery, Srikanth Kandula

Microsoft

Cameras are ubiquitous; video analysis is a big-data problem

One surveillance camera for every 11 people in Britain, says CCTV survey

Photo: ALAMY

By David Barrett, Home Affairs Correspondent

6:30PM BST 10 Jul 2013

Follow 2,866 followers

Print this article

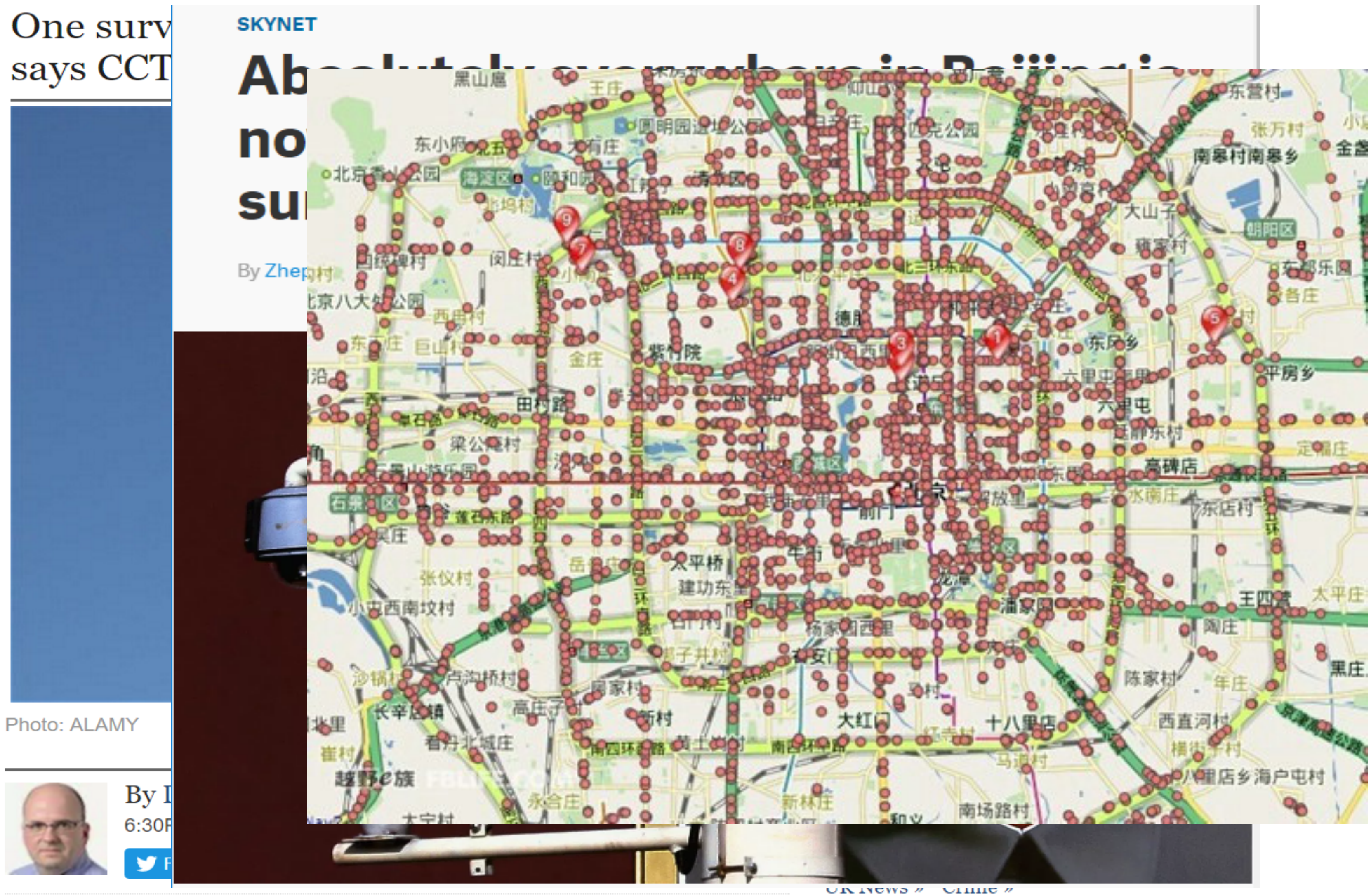
Technology

News » Politics »

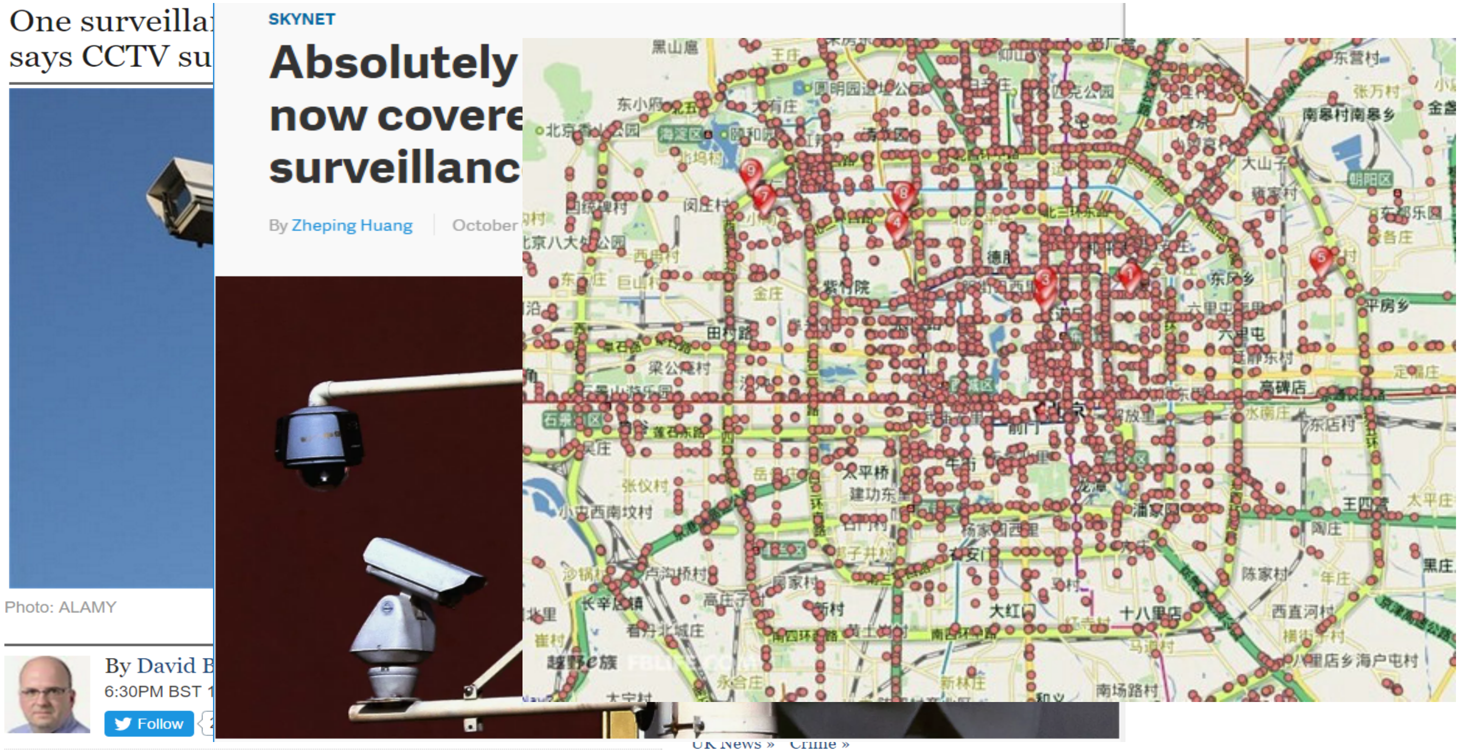
UK News » Crime »

Cameras are ubiquitous; video analysis is a big-data problem

Cameras are ubiquitous; video analysis is a big-data problem



Cameras are ubiquitous; video analysis is a big-data problem



1Mbps per camera, 10K cameras?

103 TB/day!

Analytics over city-scale cameras' video requires big-data processing

Video analytics in big-data systems

- Closed solutions: Omnicast, ProVigil, etc.
- Open solutions: MapReduce, Spark, etc.
- Spark example

Define application logic

Initialize Spark

User specifies input, parallelism etc.

Declares pipeline

```python
import logging
import io
import sys
import os
import cv2
import numpy as np

def extract_sift_features:
    def extract_sift_features_nested(imgfile_imgbytes):
        try:
            imgfilename, imgbytes = imgfile_imgbytes
            nparr = np.fromstring(buffer(imgbytes), np.uint8)
            img = cv2.imdecode(nparr, 0)
            extractor = cv2.SIFT()
            kp, descriptors = extractor.detectAndCompute(img, None)
            return [(imgfilename, descriptors)]
        except Exception, e:
            logging.exception(e)
            return []

    return extract_opencv_features_nested


if __name__ == "__main__":
    sc = SparkContext(appName="sift_extractor")
    sqlContext = SQLContext(sc)

    try:
        image_seqfile_path = sys.argv[1]
        feature_parquet_path = sys.argv[2]
        partitions = int(sys.argv[3])
    except:
        print("Usage: spark-submit sift_extraction.py "
              < image_input_path > < feature_output_path > < partitions > ")

        images = sc.sequenceFile(image_seqfile_path, minSplits=partitions)

        features = images.flatMap(extract_sift_features)
        features = features.filter(lambda x: x[1] != None)
        features = features.map(lambda x: (Row(fileName=x[0], features=x[1].tolist())))
        featuresSchema = sqlContext.createDataFrame(features)
        featuresSchema.registerTempTable("images")
        featuresSchema.write.parquet(feature_parquet_path)
```

Optimizing vision programs is a **manual** process convolving **systems** and **application** details.

Our goal

Make processing video feeds from **many** cameras

easy and **efficient**

- Auto-scaling and optimization of queries
- Vision engineers need not worry about //ism etc.
- End-users simply declare queries

Optasia: Design

Leverage relational QO for vision queries

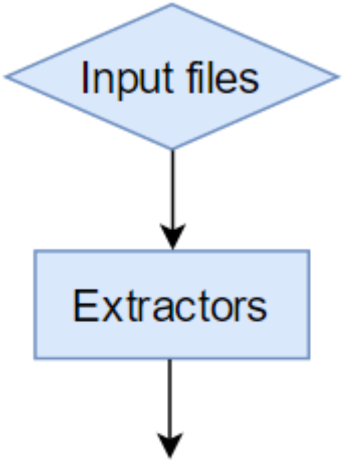
1. Vision tasks $\Rightarrow$ declarative dataflow
2. Query optimization over UDOs
3. Enhancing parallelism (eg chunk-level)

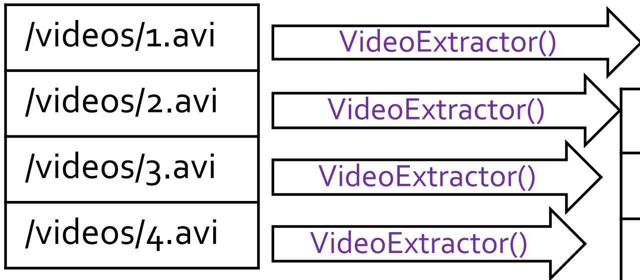
Wrapping vision modules as relational UDOs

Operator name	Relational analog
Extractor, Processor	Select and/or Project
Reducer	GroupBy and/or Aggregate
Combiner	Join

Extractors ingest data

```
$rawdata ← EXTRACT CameraID :int,
                   FrameID :int,
                   Frame :binary
FROM @"/videos/*.avi"
USING VideoExtractor();
```





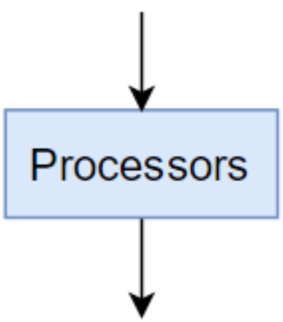
CameraID	FrameID	Frame
C1	1	[JPEG content]

CameraID	FrameID	Frame
C2	1	[JPEG content]
C2	2	[JPEG content]
C2	3	[JPEG content]
C2	4	[JPEG content]

$rawdata

Processors are row manipulators

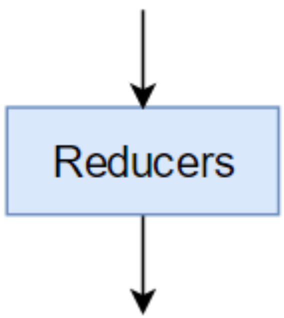
```
$lp ← PROCESS $images
USING
        HOGFeatureProcessor()
PRODUCE
        CameraID, FrameID, HOGFeatures;
```



frameId	Frame		frameId	HOGfeat
1	[JPEG content]	HOGProcessor()	1	HOG(frame 1)
2	[JPEG content]	HOGProcessor()	2	HOG(frame 2)
3	[JPEG content]	HOGProcessor()	3	HOG(frame 3)
4	[JPEG content]	HOGProcessor()	4	HOG(frame 4)

$images → $HOGfeat

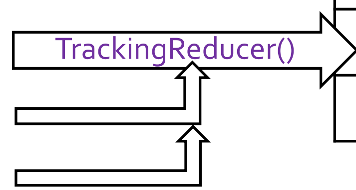
Reducers operate over groups of rows




```
$cars ← REDUCE
ON
        CameraID
USING
        TrackingReducer()
PRODUCE
        FrameID :int, CarBlob  :binary;
```

frameId	Frame
1	[JPEG content]
2	[JPEG content]
3	[JPEG content]

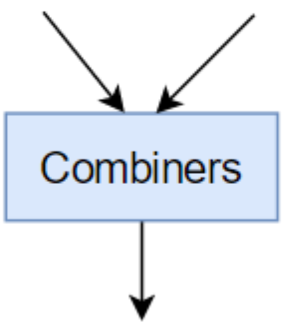
$images



FrameID	CarBlob
1	Track(frame 1,2,3)
5	Track(frame 4,5,6)

$cars

Combiners join two or more rowsets



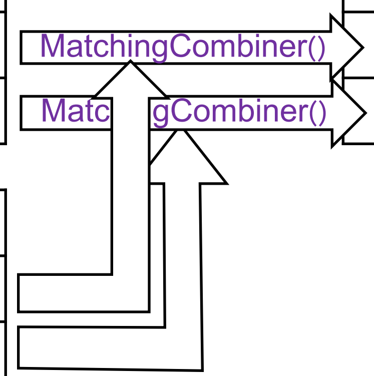

```
$distance ←
COMBINE
        $images_1, $images_2
ON
        $images_1.frameId = $images_2.frameId
USING
        MatchingCombiner()
PRODUCE
        FrameID :int, Distance :float;
```

frameId	Frame
1	[JPEG content]
2	[JPEG content]

$images_1

frameId	Frame
1	[JPEG content]
2	[JPEG content]

$images_2



frameId	distance
1	dist(frame 1.1, 2.1)
2	dist(frame 1.2, 2.2)

$distance

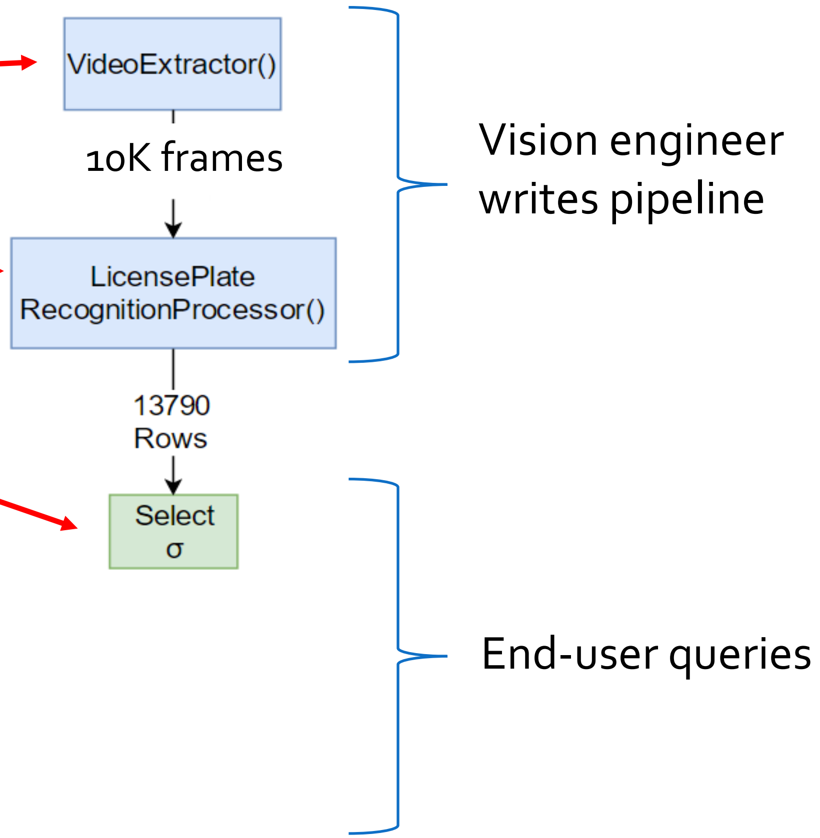
Pipelines (and queries) are declarative compositions

Example: License Plate Recognition

```
$rawdata ← EXTRACT camId :int,
                   frameId :int,
                   frame :binary
FROM @"/videos/*.avi"
USING VideoExtractor();

$lp ← PROCESS $rawdata
USING LicensePlateRecognitionProcessor()
PRODUCE camId, frameId, LP;

SELECT camId, frameId FROM $lp
WHERE $lp.LP = "ABC1234";
```



Query optimization

Cascades-style cost-based query optimizer

- Transformation rules generate alternatives, e.g., predicate push-down

  $$\varepsilon_1 \rightarrow S \rightarrow Filter \rightarrow \varepsilon_2 \quad \Rightarrow \quad \varepsilon_1 \rightarrow Filter \rightarrow S \rightarrow \varepsilon_2$$

- UDOs annotated with cost etc.

Query optimization

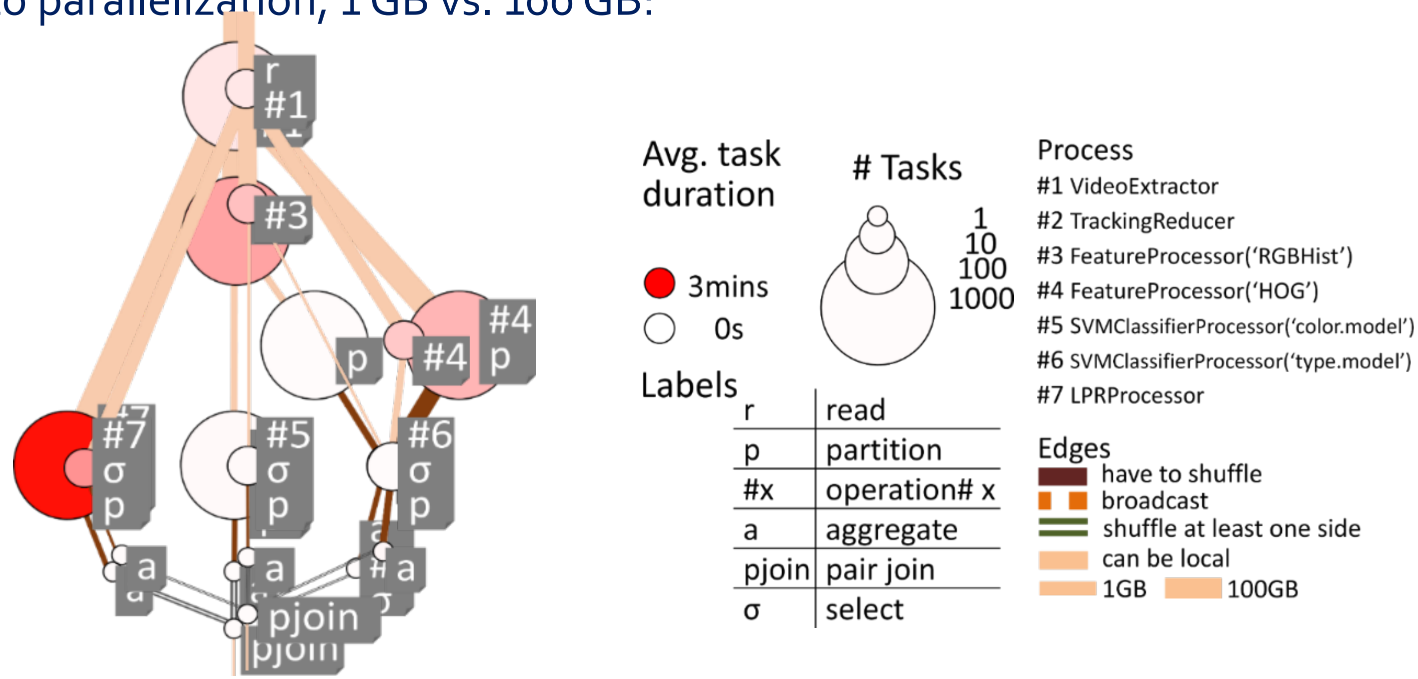
Cascades-style cost-based query optimizer

- Transformation rules generate alternatives, e.g., predicate push-down

  $\varepsilon_1 \rightarrow S \rightarrow Filter \rightarrow \varepsilon_2 \quad \Rightarrow \quad \varepsilon_1 \rightarrow Filter \rightarrow S \rightarrow \varepsilon_2$

- UDOs annotated with cost etc.

- Auto parallelization, 1 GB vs. 100 GB:

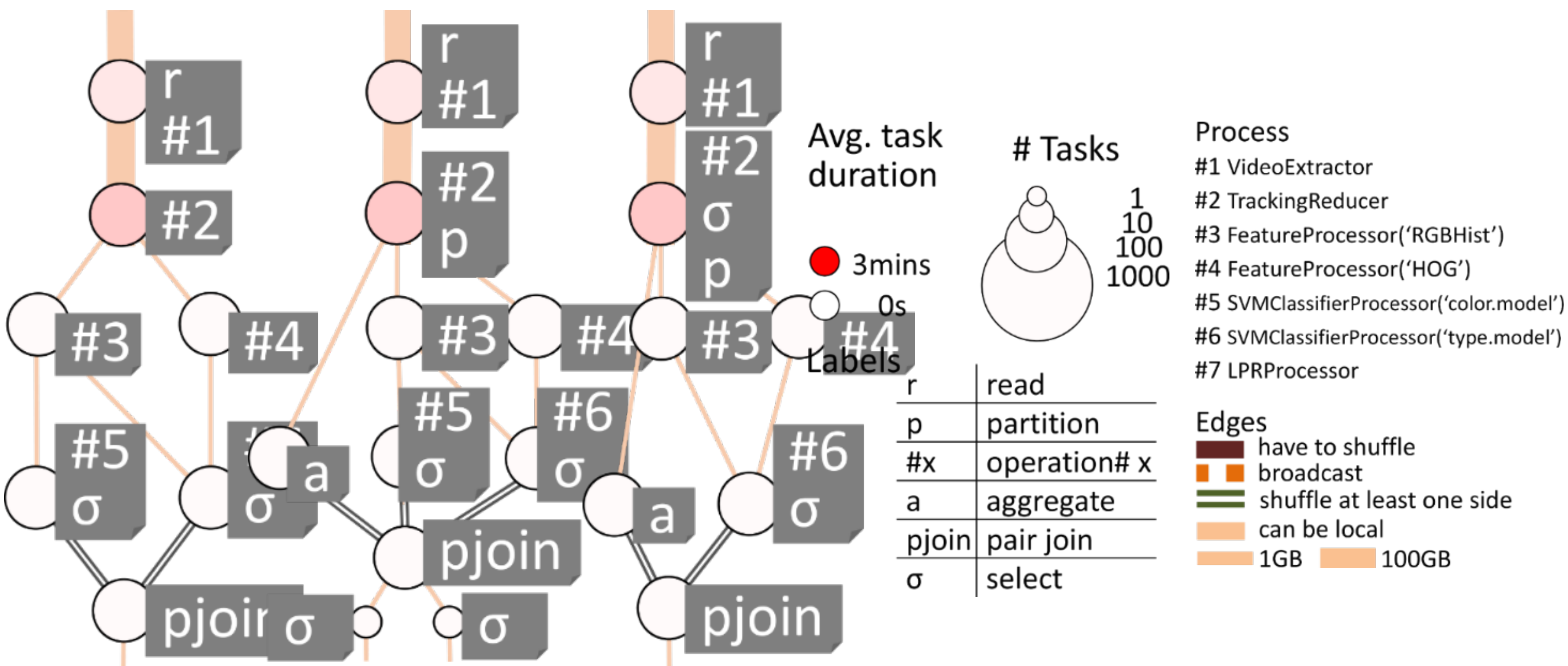


10 GB

Query optimization (contd.)

- Query de-duplication

Merge common modules across pipelines and queries.



Amber alert Combined query Traffic violation

Chunk-level parallelism

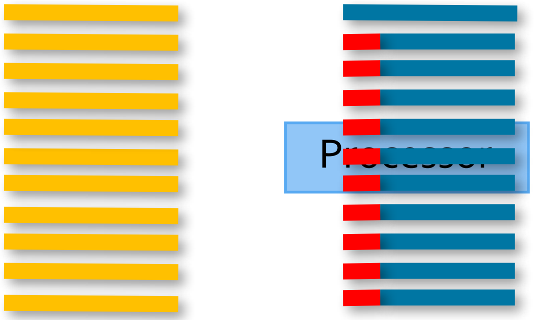
- Contextual analysis is limited to camera-level parallelism

- Idea: context of video processing is bounded in time
- Partition into overlapping chunks

Chunk-level parallelism

- Contextual analysis is limited to camera-level parallelism

- Idea: context of video processing is bounded in time
- Partition into overlapping chunks



System

1. Implemented state-of-art vision modules as dataflow operators

2. Built vision pipelines, using above modules, for
   - License plate recognition
   - Vehicle color/type recognition
   - Traffic flow mapping
   - Object re-identification

3. Query optimization extends SCOPE

Module Functionality	Operator Category	Specifics
Feature Extraction	Processor	RGB Histogram, HOG, PyramidSILTPHist PyramidHSVHist
Classifier/regressor	Processor	Linear SVM, Random Forest
Classifier/regressor	Combiner	XQDA for Re-ID
Keypoint Extraction	Processor	Shi-Tomasi, SIFT
Tracker	Reducer	KLT, CamShift
Segmentation	Processor	MOG, Binarization

Summarizing Design

Leverage relational QO for vision queries

1. Vision tasks $\Rightarrow$ declarative dataflow
2. Query optimization over UDOs
3. Enhancing parallelism (eg chunk-level)

Evaluation

- **Sample end-user queries**

Sample end-user queries

- Example query: Amber alert ("red honda civic with license AB*92*")

```
SELECT
            CameraID,
            FrameID,
            $Licenses.conf * $VehicleType.conf * $VehicleColor.conf AS Confidence
FROM        $Licenses, $VehicleType, $VehicleColor
ON          $Licenses.{CameraId, FrameId}=$VehicleType.{CameraId, FrameId} &
            $Licenses.{CameraId, FrameId}=$VehicleColor.{CameraId, FrameId}
WHERE
            $Licenses.plate LIKE l & $VehicleType.type=v & $VehicleColor.color=c;
```

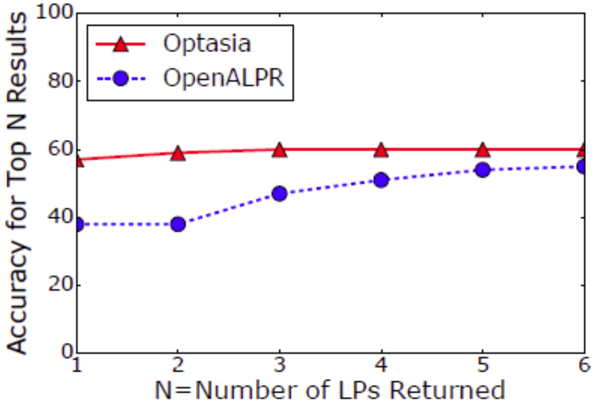
- Other examples in paper and code release

Evaluation

- Sample vision queries
- **Benchmarking vision pipelines**

Benchmarking vision pipelines

- License plate recognition:



- Vehicle counting

	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

- Vehicle type & color recognition:

	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

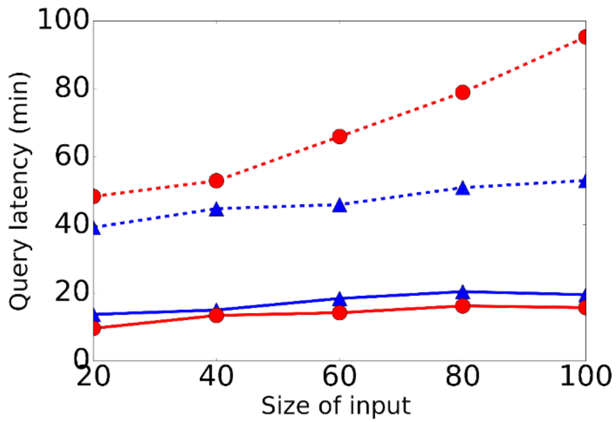
Evaluation

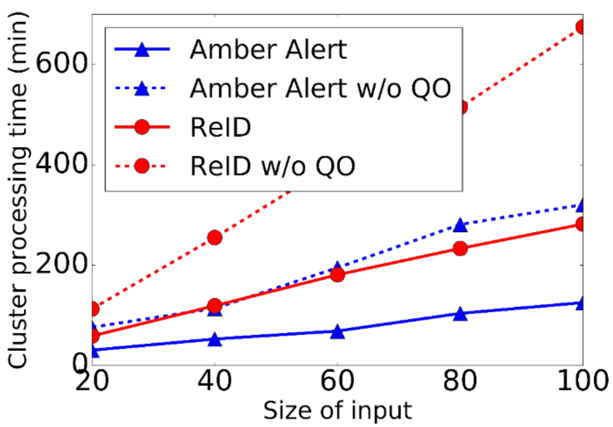
- Sample vision queries
- Benchmarking vision pipelines
- **Benchmarking end-to-end system**

Benchmarking Optasia

- **Data:** 100GB of traffic surveillance videos.

- **Queries**
  - **Amber alert** retrieves vehicles given color (red, etc.), type (SUV, etc.), and license plate number
  - **Re-ID** matches and tracks certain vehicles across two cameras.

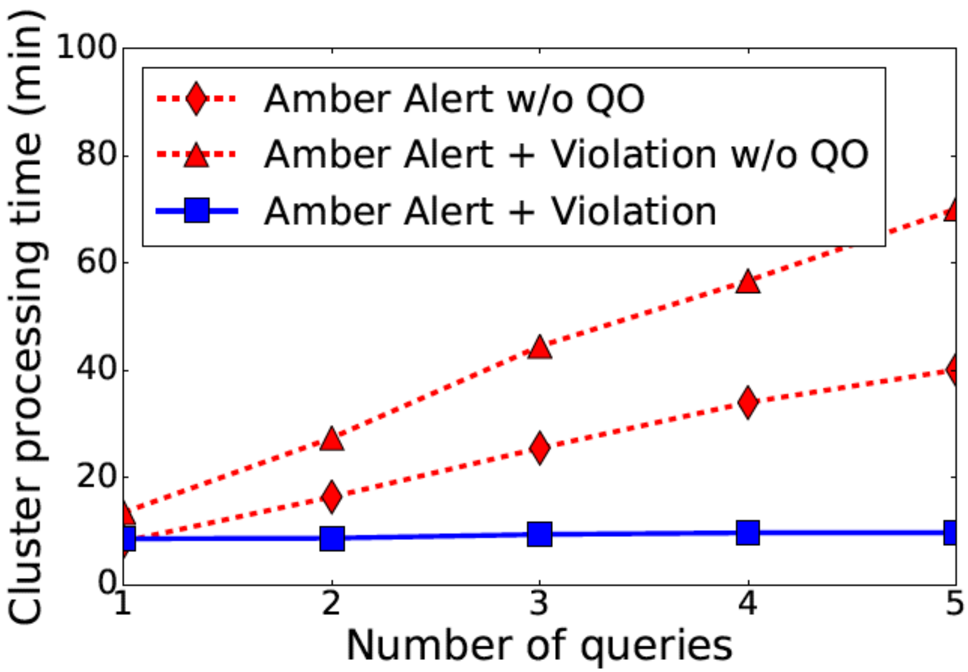




Faster query completion and lower usage of cluster resources

Benchmarking Optasia

- De-duplication



Cluster use remains unchanged due to effective de-duplication.

Benchmarking Optasia

- Chunk-level parallelism

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

Chunking increases high degree of parallelism (but overheads catch up)

Conclusion

- Video analytics in big-data systems is challenging
- Optasia: a user-friendly & efficient system
  - Leverages relational QO for vision queries
    - Relational wrappers for vision modules
    - Query optimization to de-dup, //ization, etc.
    - Enhanced parallelism (eg chunk-level)
- Evaluation shows gains in scalability and accuracy

Code and demo at http://yao.lu/optasia