Procedia Computer Science

Volume 51, 2015, Pages 2648–2657

ICCS 2015 International Conference On Computational Science

Adapting Stream Processing Framework for Video Analysis

S. Chakravarthy[1]*, A. Aved[2], S. Shirvani[1], M. Annappa[1], and E. Blasch[2]

[1] IT Laboratory and CSE department, UT Arlington, Texas (sharma@cse.uta.edu)
[2] AFRL Information Directorate/RIED, Rome, New York (alexander.aved@us.af.mil)

Abstract

Stream processing (SP) became relevant mainly due to inexpensive and hence ubiquitous deployment of sensors in many domains (e.g., environmental monitoring, battle field monitoring). Other continuous data generators (surveillance, traffic data) have also prompted processing and analysis of these streams for applications such as traffic congestion/accidents and personalized marketing. Image processing has been researched for several decades. Recently there is emphasis on video stream analysis for situation monitoring due to the ubiquitous deployment of video cameras and unmanned aerial vehicles for security and other applications.

This paper elaborates on the research and development issues that need to be addressed for extending the traditional stream processing framework for video analysis, especially for situation awareness. This entails extensions to: data model, operators and language for expressing complex situations, QoS (Quality of service) specifications and algorithms needed for their satisfaction. Specifically, this paper demonstrates inadequacy of current data representation (e.g., relation and *arrable*) and querying capabilities to infer long-term research and development issues.

Keywords: Stream processing, Image pre-processing, Video stream processing

1 Introduction

Complex event Processing (CEP) [5] started in the 80's by adding simple monitoring capabilities – in the form of triggers – to Database Management Systems (termed active DBMSs). The requirements as well as the capabilities needed for newer applications have changed drastically from its beginnings [1]. Many event specification languages, optimization techniques for processing live as well as collected data (or event logs) have been developed.

Stream processing (SP) [6] became relevant mainly due to inexpensive and hence ubiquitous deployment of sensors in many applications (e.g., environmental monitoring, battle field monitoring). Other continuous data generators (web clicks, traffic data, network packets) have also prompted processing and analysis of these streams for applications such as traffic congestion/accidents, network intrusion detection, and personalized marketing.

*Part of this work was done while the author was visiting AFRL (Air Force research Laboratory) during Summer 2014 as part of the VFRP (Visiting Faculty Research Program.)

 Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2015

doi:10.1016/j.procs.2015.05.372

Event stream processing systems integrate both event and data stream processing holistically to take advantage of both models for applications that need to process continuous queries (CQs) as well as detect complex events. These events can be combined in complex ways (using event operators) to infer complex situations.

Research and development on image processing [7, 12], computer vision [13], and video analysis [8] has been ongoing for several decades and strides have been made in object identification, object classification, and background analysis. The current state-of-the-art is in object recognition, object classification, identification of certain activities (e.g., tracking of objects over multiple frames, separating background from moving objets from frames).These results have been used in recent approaches proposed for video analysis [2, 3].

1.1 Problem Description

Video stream analysis can be characterized as extraction of interesting information from individual video frames, aggregation and correlation of feature to identify and infer a higher level activity or an even model these events for situation detection [10]. Once meaningful and discriminating aspects of video frames and sequences have been extracted (using various image processing, machine learning, and grouping/correlation techniques), inferring interesting events is important for recognizing situations. Event detection needs to be done as the video is generated and to satisfy quality of service (or QoS) requirements, such as latency, throughput, and memory usage.

In this paper, we employ stream processing framework for analyzing video in a general purpose manner for the purpose of situation recognition. Currently, most of the available published work in this regard is customized and work for specific applications or contexts. VIRAT (Video and Image Retrieval and Analysis Tool) is a good example of the customized approach[1].

The contributions of this paper are:

- Evaluation of current data models and query languages for their suitability of video processing,
- Identification of the class of queries that can be expressed using current data models and query language,
- Identification of extensions needed for both data models and CQL
- Proposing an architecture for video pre-processing.

Road map: Section 2 briefly discusses related work in relevant areas. In Section 3, we discuss motivation behind our effort and why relational representation and SQL is not well-suited for our purpose. Section 4 shows the stream processing architecture for video processing. Section 5 discusses an alternative model and how the same queries can be better expressed and processed. Conclusions and future work are discussed in Section 6.

2 Related Work

The amount of literature on various aspects of stream data processing, complex event processing and image and video processing is overwhelming. Hence we review the related literature very briefly.

Complex event processing has seen a resurgence although the need for events, rules, and triggers were realized more than two decades ago. The advent of stream processing applications

[1] "BAA-08-20: Video and Image Retrieval and Analysis Tool (VIRAT)". Information Processing Technology Office. March 3, 2008.

has strengthened the applicability of complex event processing and alerts/notifications in environments that were never considered earlier. SASE [16] is a complex event processing system that performs data to information transformations over real-time streams. It is targeted for filtering data and correlating them for complex pattern detection and transformation to events that provide meaningful, actionable information to end applications.

Aurora [15] is a stream processing system for processing continuous queries on data streams. Continuous queries in Aurora are specified in terms of a dataflow diagram consisting of boxes and arrows (using a GUI). *STREAM* [14] Stanford Stream Data Manager (or *STREAM*) is a general-purpose data stream management system which implements Continuous Query Language (CQL), an extension of SQL, as a part of it. *MavEstream* [6] has addressed stream processing holistically from a QoS perspective. This work has addressed modeling (for capacity planning), scheduling (for latency) and load shedding for bounding errors.

Automated real-time processing of video for surveillance and monitoring is a challenge in many ways. Current technology is not at the point where it can be done with complete accuracy. Hence the goal is assisted automation where the intent is to reduce the amount of video analyzed by a human and still be able to accomplish the goal[2]. The situation being monitored depends upon the context and the semantics can vary significantly. For example, parking lot surveillance for improperly parked cars is very different from identifying break-ins or identifying whether a vehicle turned back and did not cross a check point. As the number of video cameras are increasing in almost every walk of life, it is important to develop assisted ways of continuous monitoring as desired rather than manual identification of situations.

The work by Aved [3] is an effort in that direction and has proposed a Live Video Database Management System (LVDBMS) which provides a framework for managing and processing live motion imagery data. It takes a modular, layered approach (e.g., camera layer, spatial processing layer, etc.) by defining adapters (similar to device drivers) so that the upper layers are independent of the specific device characteristics. Apart from video analysis, there is considerable work on fusion that tries to improve object tracking and labeling [4] using additional sources of information.

3 Motivation

Stream processing came about mainly for overcoming some of the assumptions made for processing stored, large volumes of data using DBMSs. This has highlighted some fundamental differences between traditional data processing and stream processing. For example, most of the traditional stream processing deals with structured, text and numeric data. Many of the operations used on these data are well-defined and optimized by a query optimizer. For stream processing, the data representation was the same, but the characteristics of a stream was very different from that of stored relations. A stream is theoretically unbounded and is prone to bursty input rates (or fluctuations). Storing and processing of stream data using DBMS techniques introduced too much latency. In addition, QoS needs of stream data processing were completely lacking in a traditional DBMS. Computations have to be done in a single pass as the stream is not stored to be accessed multiple times.

Along the same lines, there are some differences between stream processing and video processing. There is significant amount of pre-processing required for extracting non-structured input into some structured representation. Also, a common, canonical representation need to be identified and extracted from each frame. In order to characterize contents properly in a frame, it may be necessary to consider a sequence of contiguous frames. As an example, if a machine wants to infer whether it is the *same object* in multiple frames, not only multiple frames need

[2]Refer to Rand reports RR154 on Motion Imagery Processing and Exploitation and TR 133.

to be processed, but a way to establish "sameness" with acceptable accuracy is needed. If a system wants to infer whether something was exchanged, it may be necessary to study several frames for the presence and subsequent absence of something. In addition, spatial and temporal operators are needed to express even simple situations in addition to using the state-of-the-art approaches for object identification, classification etc. which is constantly evolving.

However, the framework of stream processing seems well-suited for video stream analysis where a video can be viewed as a stream of frames. Multiple cameras may provide multiple streams that may need to be used for identifying complex situations (e.g., how many people who entered a building also left the building in an hour?). The processing requirements are significant as well as the types of operators and their computational complexity. But the same framework and hence some of the stream processing architecture and techniques (e.g., scheduling techniques for low latency) can be used for video analysis.

3.1 Stream Processing Approach for Video Analysis

Image processing, video analysis and fusion have been going on for a long time and a number of techniques have been developed for that purpose. The emphasis has shifted to situation awareness and fusion as object identification, classification, and tracking have reached a point where they have been employed in commercial and other systems (e.g., Google images, Facebook, image-based shopping, VIRAT project etc.) with considerable success.

Perhaps the current situation in video processing can be compared to that of pre-relational DBMS approaches to data processing and analysis. Both hierarchical and network DBMSs required writing searches and queries in a navigational style programming language. They were certainly better than the earlier file-based systems, but the burden of knowing the details of representation, traversing the data structures, and optimization were on the user.

The main reason relational DBMS gained popularity and wide acceptance, apart from its simplicity of representation, was the ability to express "what" to compute instead of "how" to compute (declarative or non-procedural vs procedural or navigational paradigms) using a few constructs. As the optimization techniques improved, relational systems started competing with earlier systems in efficiency and its usage increased due to the combined advantages of simple representation and non-procedural specification. This exactly what is lacking in the efforts on video stream analysis – a good data model to capture video contents, an expressive, declarative query specification language, and optimization and parallel processing techniques to match real-time and other QoS requirements.

Image and video analysis research has produced many research prototypes specifically addressing problems such as face recognition, object classification, and tracking objects etc. Our proposal for the use of the stream processing framework and approaches for video analysis follows similar reasoning. Given an appropriate representation that is rich enough to capture salient image contents, it can be viewed as stream processing of video frames (one or more streams depending on the number of cameras). Now, this can be combined with other streams (e.g., sensor data, intelligence reports, weather) as well as stored relational data (e.g., converting lat/lang to a context in terms of region etc.). By developing appropriate additional operators (e.g., semantic window, spatial and temporal as needed), hopefully, a wide variety of situations can be expressed *declaratively or non-procedurally.* Finally, these operators can be composed into a query plan and optimized by the system rather than by programmers. There may be multiple ways of computing an operator based on the characteristics of the video being processed and other considerations such as the accuracy of the result needed etc.

4 Video Stream Processing Architecture

Before we present the architecture for video stream processing, we need to understand how we convert a video (or video frames) into currently available representation or any other representation that we may develop for this problem. The first step is to convert each frame as well as consider sequence of frames for converting.

4.1 Frame Pre-Processing

The first step towards video stream analysis is to process each frame of a video separately and extract interesting features that are present and useful. This falls in the realm of image processing and using existing techniques interesting objects, feature vectors of objects, and bounding boxes along with background and scene information need to be extracted. As there are abundant results in this regard, we do not plan on developing new techniques for this. Our approach will be to use the best available techniques (from Matlab and OpenCV libraries) to extract what is useful for expressing and computing situation awareness. What is extracted may vary depending on the context of the video, various properties of the camera and the resulting video, and what kind of queries or fusion is intended on the video streams under consideration.

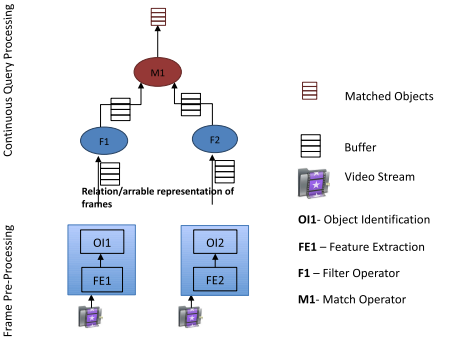

Figure 1: Video Stream Processing Architecture

There are a variety of feature vectors that can be extracted from a frame as well as color, position, and other information. Figure 1 shows the overall architecture we are proposing. It consists of two layers: i) a pre-processing layer to extract the desired representation and ii) a continuous query processing layer for optimizing and processing queries. The operator FE1 shown in Figure 1 does the feature extraction from *each frame individually*. There may be different types of feature extraction operators for different purposes. The input to this operator is a stream of frames and the output is a traditional relation that contains desired information including camera id, frame number, feature vector, object id etc. Note that each object is given a unique id during this step.

The next step in frame pre-processing is to process multiple frames to determine whether the same object is present in multiple frames (shown as OI operator). This will require an operator that matches feature vectors across multiple frames to determine whether they are the same (or similar) objects. We plan on leveraging some of the work done by [3] for this purpose.

CREATE TABLE	VS(
c_id	INT NOT NULL,
f_id	INT NOT NULL,
object_id	INT,
feature_vector	CHAR(10),
bounding_box	CHAR(10),
ts	INT NOT NULL,
PRIMARY KEY	(c_id, f_id, object_id));

Table 1: SQL Create Table Statement

At the end of frame pre-processing, we will generate the desired data model – either a relation or an arrable (described in Section 5) or an extended data model to be defined. The next layer will compute queries expressed on the representation produced at the end of the pre-processing layer.

4.2 Architecture

We will start with the current prototype (MavEStream [6]) for extensions, experimentation, and performance evaluation once the operators and their their semantics are clearly defined, and situations are expressed using the language developed for video stream analysis. The video stream processing architecture (adapted from [9]) is shown in Figure 1.

Continuous queries (CQs) are converted into query plans consisting of operators in a dataflow architecture for processing. A general example is shown where two video streams are pre-processed using filter and match operators. A filter operator (F1 and F2) filters tuples that may not be needed (e.g., background objects). A match operator (M1) compares objects from the two video streams to determine whether the same object (that entered a building) also exited the building. Windows will be used to constrain the number (or duration) of frames that are considered for this situation.

As MavEstream has been implemented using generic data types in Java, we will be able to keep the architecture and modify only the components that are specific to video analysis. Currently existing buffer classes, scheduling, feeder for feeding the frames at a specific rate can be reused to make progress on the solution to the problem discussed in this paper.

We plan on extending the MavEstream prototype as described above to perform video stream analysis based on the SQL or arrable operators and semantics. presented in this paper. We will also analyze and develop metrics that can be used to compare and study the integrated system. We will do extensive tests and experimental analysis using the developed metrics to ensure the performance and robustness of the system.

c_id	f_id	object_id	feature_vector	bounding_box	ts
1	1	1	RED	TOP_LEFT	1
1	1	2	GREEN	TOP_RIGHT	1
1	1	3	BLUE	BOTTOM_RIGHT	1
1	1	4	RED	TOP_RIGHT	1
1	2	1	GREEN	BOTTOM_LEFT	6
1	2	2	RED	TOP_RIGHT	6
1	2	3	GREEN	BOTTOM_LEFT	6
1	2	6	BLUE	BOTTOM_RIGHT	6
1	3	1	GREEN	TOP_LEFT	53
1	3	2	BLUE	TOP_RIGHT	53
1	3	3	GREEN	TOP_RIGHT	53
1	3	5	BLUE	BOTTOM_LEFT	53
1	4	1	BLUE	BOTTOM_RIGHT	72
1	4	2	RED	TOP_RIGHT	72
1	4	3	BLUE	BOTTOM_RIGHT	72
1	4	4	GREEN	BOTTOM_RIGHT	72
1	4	5	BLUE	TOP_LEFT	72

Table 2: Video stream representation as a traditional relation

5 Data Models and Query Expressiveness

As SQL is defined over the relational model and CQL is an extension of SQL that includes window operators for stream queries, the natural starting point for us would be to generate a relation as a result of the pre-processing stage described in Section 4.

Consider a single video camera and frames from that camera are pre-processed and objects identified to populate the schema generated by the create table

SELECT	object_id, count(frame_id)
FROM	VS
GROUP BY	object_id
HAVING	count(frame_id) > 3

Table 3: Example 1. Retrieve objects that appear in more than three frames

statement shown in Table 1.

A sample populated table that holds the contents of a few video frames is shown in Table 2. In this table, we have simplified the representation of the feature_vector and the bounding_box attributes as they can be vectors of different sizes (depending upon the feature vector type chosen during frame pre-processing) and cannot be represented in a traditional relation. For further simplicity, we have used color for the feature vector and quadrant of the frame in which the object was found for the bounding box. This table captures a few frames from a single camera and objects that have been identified among the frames at the end of the pre-processing stage. In the remainder of this section, we will illustrate, with a few examples, the types of SQL queries that can be expressed over the above schema, their ease of understanding, and their optimization issues.

The query in Table 3 groups the tuples by object_id and calculates the number of frames in which each object appears. Then objects that appear in more than three frames (not consecutive) are selected for output. Additional scalar conditions on feature_vector and bounding_box can be included in the above query. The number of frames in which an object appears may be indicative of the presence of the same object over multiple periods of time. However, if one wants to express queries where the same object appears in multiple consecutive frames, it requires cartesian products as shown in Table 4. If one wants to retrieve objects that appeared for the longest time in the video, it is extremely difficult to express in SQL as one cannot assume ordering on the timestamp attribute in a relational model. Additional information such as ROWID (a feature of Oracle indicating row number in a relation) may have to be used (where available) for this purpose.

The query shown in Table 4 will list all objects (ordered) and the 3 consecutive frames they appear in. The above query is difficult to understand and also difficult to optimize as self-joins and self cartesian-products are not optimized well by traditional DBMS optimizers. Furthermore, if the tables are large (as they are likely to be), the amount of intermediate results generated and hence the time taken to compute this query is likely to be prohibitive. The number of Cartesian products/joins will increase as the number of frames considered increases. This query can be easily modified to include conditions and filter objects based on feature vector values or bounding box values.

SELECT	DISTINCT vs1.object_id vs1.f_id, vs2.f_id, vs3.f_id
FROM	VS AS vs1, VS AS vs2, VS AS vs3
WHERE	vs2.f_id = vs1.f_id + 1
	AND vs3.f_id = vs2.f_id + 1
	AND vs1.object_id = vs2.objects_id
	AND vs2.object_id = vs3.object_id
ORDER BY	vs1.object_id ASC

Table 4: Example 2. Retrieve all objects that appear in three consecutive frames

The query in Table 4 did not use the notion of windows introduced in stream processing. CQL can be used to express queries that involves windows (time or tuple) over which computations need to be done. This is extremely important to express queries such as "compute the average number of cars in a parking lot on a hourly basis" or using rolling/tumbling windows such as "compute the number of people entering a building over one hour starting every half hour".

Windows are also needed for converting unbounded input steams into manageable partitions(or windows) over which computations are done. However, a better representation than the traditional relation is needed for expressing easy-to-understand and easy-to-optimize queries.

5.1 *Arrable* as an Alternative Data Model

```
SELECT   object_id, result.obj_count
FROM      ( SELECT vs1.object_id, count(*) as obj_count
            FROM VS as vs1 [Range 1 hour Slide 1/2 hour]
            GROUP BY vs1.object_id
          ) as result
WHERE    obj_count >= ALL
              (SELECT count(*)
              FROM VS as vs2 [Range 1 hour Slide 1/2 hour]
              GROUP BY vs2.object_id)
```

Table 5: Example 3. Retrieve all objects that appear most in each hour of video. Update the results for every 1/2 hour.

An extension to the relational model that has been proposed to deal with complex computations in the financial domain (e.g., running averages) is an *arrable* [11]. An arrable combines vector (as multi-set) representation with the relational model where an attribute can be an array of scalar values. All relational operations (e.g., select, project, join) can be performed on an arrable and in addition computations can be easily expressed on vectors by grouping on certain attributes and assuming an order (which is not possible in a relation). AQuery [11] is an extension of SQL which was introduced to deal with computations that require order dependence (e.g., running averages). AQuery has also introduced a number of vector operations (order preserving, size preserving, and size not preserving). Hence, the arrable and the AQuery language can be viewed as a generalization of the traditional relational model and SQL, respectively.

For obtaining an arrable representation, a group by operation can be performed to brig values together into a vector representation. For example, if we were to group Table 2 on c_id and object_id, we get the arrable shown in Table 6. Using this arrable representation, we will show how the examples shown earlier can be expressed in AQuery.

c_id	f_id	object_id	ts
1	[1, 2, 3, 4]	1	[1, 6, 53, 72]
1	[1, 2, 3, 4]	2	[1, 6, 53, 72]
1	[1, 2, 3, 4]	3	[1, 6, 53, 72]
1	[1, 4]	4	[1, 72]
1	[1, 2, 3, 4]	5	[1, 6, 53, 72]
1	[2]	6	[6]

Table 6: Arrable representation (VS1) when grouped by *c_id and object_id*

The specification of query shown in Table 3 is not that much different in AQuery as shown in Table 7. Square brackets are used to denote operations on vector attributes and count will return the number of elements in the vector. Note that this arrable representation can be used to compute the time period during which an object appeared in the video very easily by using a function on the ts vector (last[ts] - first[ts] with ASSUMING ORDER on ts.

However, Example 2 becomes much simpler both for ease of understanding and optimization as elaborated in [11]. For this, an ASSUMING ORDER clause has been introduced in AQuery that orders the specified vector attributes so that computations can make use of that order.

```
SELECT   object_id, count[f_id]
FROM     VS1
HAVING   count[f_id] > 3
```

Table 7: **Example 1a. Retrieve objects that appear in more than three frames**

As is easy to see, there are no multiple Cartesian products in Table 8. The boolean consecutive function (consecutive(n, vector)) is used to select tuples that satisfies the condition (hence its use in the where clause). The ASSUMING ORDER makes sure that the f_id vector is ordered so that the consecutive function

can be applied on the ordered vector. Consecutive function checks the vector for 3 consecutive values and returns the vector if that is satisfied. The above is a much simpler way to express and optimize the query. It is also possible to group by c_id and f_id to get a different arrable representation of the Table 2 using which different class of AQueries can be expressed.

5.2 Desiderata

In this section, we have evaluated the applicability of two data models (relational and arrable) and three query languages (SQL, CQL, and AQuery). It is clear that the representation of the contents of a video frame (in terms of feature vectors and bounding boxes) cannot be fully captured by these data models. Similarly, complex situations cannot be expressed using only these data models.

We plan on extending the arrable data model and AQuery language further to accommodate feature vectors and bounding boxes. Also, operators such as filter and match need to be precisely defined and algorithms developed. Query language need to be extended to express interesting events such as "approaching a checkpoint", "exchanged something", and "turned away from checkpoint" for composing complex situations from interesting events. The architecture need to be implemented to showcase the generality of processing (and optimizing) any continuous that can be expressed. QoS issues need to tackled.

SELECT	object_id, f_id
FROM	VS1
ASSUMING	ORDER f_id
WHERE	consecutive(3, [f_id])
ORDER BY	vs1.object_id ASC

Table 8: **Example 2a. Retrieve all objects that appear in three consecutive frames**

6 Conclusions

In this paper, we have proposed and evaluated the applicability of stream processing framework for video stream analysis and demonstrated the types of situations that can be expressed using the relational & *arrable* representations. We showed examples of situations that can be expressed using SQL, CQL, and AQuery languages. It is evident that the available representations and language constructs are not sufficient to express complex situations. Extensions may also needed in the realm of image (or frame) pre-processing to convert contiguous frames into the desired data representation for further processing.

6.1 Future Work

We have demonstrated that the *arrable* representation is better for expressing and computing situations as compared to the traditional relational representation and SQL. *Arrable* can be used a basic data model for expressing queries for video analysis. However, the data model needs to be further extended for accommodating feature vectors and bounding boxes in their generality to specify spatial and temporal computations. Furthermore, current windows used for stream processing are also not sufficient for expressing more complex situations that are not time- or tuple-based. Finally, New operators and their implementations – both for image pre-processing as well as situation monitoring – need to be identified and defined along with QoS specifications and algorithms for their satisfaction. New temporal and spatial operators need to defined with proper semantics and algorithms to express more interesting situations.

Here are a few queries (situations) that cannot be currently expressed or computed using the data models and query languages discussed in this paper:

- List all objects that approached a (checkpoint, entrance) but did not enter,

- Generate a report for each day a specific object (e.g., car) left and returned (from a parking lot),
- Did two operators exchange anything. Exchange can be interpreted as one of the operators carrying an object for a time, but after the operator came very close to another operator (e.g., bounding boxes intersected), that object was not with the operator anymore (and is with the other operator).

References

[1] Eman Anwar, L. Maugis, and Sharma Chakravarthy. A New Perspective on Rule Support for Object-Oriented Databases. In *SIGMOD Conference*, pages 99–108, 1993.

[2] Alexander J Aved. *Scene Understanding for Real Time Processing of Queries Over Big Data Streaming Video*. PhD thesis, University of Central Florida Orlando, Florida, 2013.

[3] Alexander J Aved and Kien A Hua. An informatics-based approach to object tracking for distributed live video computing. *Multimedia Tools and Applications*, 68(1):111–133, 2014.

[4] E Blasch, J Nagy, A Aved, W Pottenger, M Schneider, R Hammoud, EK Jones, A Basharat, A Hoogs, G Chen, et al. Context aided video-to-text information fusion. In *Intl Conf. on Information Fusion*, 2014.

[5] S. Chakravarthy, B. Blaustein, A. Buchmann, M. Carey, U. Dayal, D. Goldhirsch, M. Hsu, R. Jauhari, R. Ladin, M. Livny, D. McCarthy, R. McKee, and A. Rosenthal. HiPAC: A Research Project in Active, Time-Constrained Database Management. Technical report, Xerox Advanced Information Technology, Cambridge, 1989.

[6] Sharma Chakravarthy and Quinchun Jiang. *Stream Data Management: A Quality of Service Perspective*. Springer, April 2009.

[7] Randy Crane. *Simplified Approach to Image Processing: Classical and Modern Techniques in C*. Prentice Hall PTR, 1996.

[8] Nevenka Dimitrova, Hong-Jiang Zhang, Behzad Shahraray, Ibrahim Sezan, Thomas Huang, and Avideh Zakhor. Applications of video-content analysis and retrieval. *IEEE multimedia*, 9(3):42–55, 2002.

[9] Qingchun Jiang, Raman Adaikkalavan, and Sharma Chakravarthy. MavEStream: Synergistic Integration of Stream and Event Processing. In *International Conference on Digital Communications*, pages 29–29, 2007.

[10] Gal Lavee, Ehud Rivlin, and Michael Rudzsky. Understanding video events: a survey of methods for automatic interpretation of semantic occurrences in video. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 39(5):489–504, 2009.

[11] Alberto Lerner and Dennis Shasha. Aquery: Query language for ordered data, optimization techniques, and experiments. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 345–356. VLDB Endowment, 2003.

[12] John C Russ and Roger P Woods. The image processing handbook. *Journal of Computer Assisted Tomography*, 19(6):979–981, 1995.

[13] Mubarak Shah. Fundamentals of computer vision. 1997.

[14] STREAM. Stanford Stream Data Management (STREAM) Project, 2003. `http://www-db.stanford.edu/stream`.

[15] Stanley B. Zdonik, Michael Stonebraker, Mitch Cherniack, Ugur Çetintemel, Magdalena Balazinska, and Hari Balakrishnan. The aurora and medusa projects. *IEEE Data Eng. Bull.*, 26(1):3–10, 2003.

[16] Haopeng Zhang, Yanlei Diao, and Neil Immerman. On complexity and optimization of expensive queries in complex event processing. In *SIGMOD*, pages 217–228, 2014.