AFRL SUMMER FACULTY FELLOWSHIP PROGRAM (SFFP)
(10 weeks – June 12^{th} 2017 through Aug 18^{th} 2017)

Final Report submitted by

Sharma Chakravarthy
The University of Texas at Arlington
sharma@cse.uta.edu
http://itlab.uta.edu/sharma

Augmenting Video Analytics Using Stream and Complex Event Processing

To

Alexander J Aved, Ph.D.
Email: alexander.aved@us.af.mil
AFRL Information Directorate/RIED
525 Brooks Road
Rome, NY 13441-4505

September 9, 2017

Contents

1 Motivation **6**

1.1 Focus of This work 8

2 Relevant Work **10**

2.1 Image and Video Analysis (IVA) 10

2.2 Complex Event Processing (CEP) 12

2.3 Stream Data Processing (SP) 13

2.3.1 Arora and Borealis 13

2.3.2 STREAM 14

2.3.3 MavStream 14

2.3.4 Others 15

2.4 Discussion 15

3 Challenges **16**

3.1 What to Extract From a Video 18

3.1.1 Primitive Contents 18

3.1.2 Non-Primitive Contents 19

3.2 How to Represent Extracted Data? 19

3.3 Expressing Queries – Towards CQL-VA 21

3.4 Evaluating CQL-VA Queries 21

3.5 Optimization of CQL-VA Queries 21

3.6 Architectural Requirements 22

4 Objects and Event Extraction/Detection **23**

4.1 Primitive Event/Object Detection 23

4.2 Non-primitive Event Detection 24

5 Representation Of Extracted Data **25**

5.1 Discussion 32

5.2 Testing SET1 Using Oracle (Release 10.2.0.4.0) 33

5.3 Proposed Data Representation 48

6 Express and Evaluate CQL-VA **50**

7 Optimizing CQL-VA **51**

8 MavVStream Architecture **52**

9 Conclusions **53**

List of Figures

1 Integrated Event Stream Processing Architecture 52

Augmenting Video Analytics Using Stream and Complex Event Processing

September 9, 2017

Abstract

Image and Video Analysis (IVA) has been ongoing for several decades and has developed a suit of techniques for image and video processing: object identification and re-identification, event & activity modeling, and summarizing a video to name a few. A large number of techniques (both pattern recognition-based and machine learning-based) for processing video frames to characterize objects have been developed to deal with camera angles, lighting eects, color dierences, as well as object identication and re-identication. Feature extraction, segmentation, and separation of background from foreground use numerous algorithms and machine learning techniques to deal with specic application domains.

In addition, several indexing approaches have been proposed and developed for storing and efficiently retrieving the information extracted. Once indexed, the videos can be *searched* for specific occurrences of events and objects. Separately, *querying* based on image/video contents (QBIC/QBVC) has been developed by matching features of a given image to a database of extracted feature sets. Most of these use specific techniques developed for specific purposes. In contrast, database management research has focused on a canonical, rich data representation that is amenable to general-purpose, non-procedural querying, aggregation, and report generation. Although the query language is limited, it can be optimized using general principles. The query language used (Structured Query language or SQL) is grounded in set theory.

This work posits that the advantages of general-purpose querying and aggregation can be extended to augment analyzing of video contents. This is a significant departure from the traditional way of analyzing images and videos. This approach, we believe, will augment the current image and video analysis capabilities. For certain types of videos, queries expand the scope of analysis that can be performed. In order to accomplish this, first, we need to determine what to extract from a video by fully leveraging the state-of-the-art techniques available in image processing. The extracted content need to be suitably represented for expressing queries. In this report, we focus on: i) detecting primitive events using window operators and stream processing techniques in addition to traditional image processing techniques, ii) event operators for composing and detecting "interesting" higher-level events from primitive events, and ii) new operators as well as spatial, temporal, and window-based operators for expressing queries that correspond to situations.

Key Terms: Stream Data Processing, Complex event processing, Video content extraction, Image and video analysis

1 Motivation

Complex event Processing (CEP) has come a long way (since the mid Eighties) to support the *push* paradigm. The basic idea behind the push paradigm is to specify and *automatically detect* events in different contexts/domains so that some meaningful action can be taken in a *timely manner and without user intervention* when those events occur and are detected. Push-based applications typically use rules with an event component, and optional condition component, and an action component. For example, turning the lights off in a building at a specific time can be modeled as a rule whose event is a periodic temporal event (occurs at, say, 9 pm every day.) Similarly, inventory can be re-ordered automatically by writing a rule whose event is quantity-on-hand going below a specified threshold. Finally, taking some action when an *interested person* comes out of a building under surveillance can be specified as a rule where the event corresponds to the detection of that person coming out of the building in a video. Whether this is done in real-time or as forensics has significant implications which will not be addressed in this report.

The research on event specification and processing has identified domain specific *primitive* events (e.g., insert, delete, and update for Relational Database Management Systems or RDBMSs, vehicle slowdown, stoppage for traffic domains, etc.) and mechanisms (i.e., event operators) to compose these primitive, temporal, and other types of events into higher level or *interesting* events in that domain. For example, integrity of an "entire transaction" (as an interesting event in that domain) in a relational DBMS can be quantified as a sequence event of "begin transaction" (primitive event) followed by any number of inserts/deletes (again primitive events) followed by the "end transaction" (primitive event.) Similarly, for the traffic domain, an "accident" as an interesting event can be specified by composing "vehicle slowing down" (as a primitive event) followed by "vehicle stoppage" (as a primitive event) in a particular lane in a particular direction on the same road. Some of the primitive events may have to be computed using data in the form of streams coming from, for example, a vehicle. Similarly, it should be possible to express either as an event or as a query on stream data that a person who entered a building came out within m minutes. Or list all persons who stayed in the building beyond n minutes. Languages for event specification, their semantics, efficient algorithms for their detection, and different architectures/paradigms have been developed.

Stream processing (SP) became relevant mainly due to the availability of inexpensive sensors at the beginning of the 21st century and their ubiquitous deployment in many applications (e.g., environmental monitoring, battle field monitoring). The basic idea of stream processing was to provide computations as well as real-time responses for monitoring tasks (to take the human out of the loop) based on large amounts of non-stop (continuous 24/7 or unbounded streams) raw data coming in from sensors. There are a number of continuous data generators – web clicks, traffic data, network packets, mobile devices – which required a different kind of processing and analysis than those supported by a relational Database management System (or RDBMS.) Traditional persist and process paradigm used in traditional RDBMSs introduced too much latency (in spite of efficient associative retrieval using indexes and clever buffer management policies to reduce swaps) that was not satisfactory for real-time applications. Hence, a single pass approach (You Look Only Once or YOLO) using *only* main memory was adapted for processing such stream data to include support for Quality of Service (or QoS) requirements. Stream processing used three QoS requirements: latency (average time a tuple spends in processing), memory utilization (amount of

maximum memory needed), and throughput characteristic (whether the output is continuous or bursty.) These requirements were new and not supported by traditional RDBMSs mainly designed for a different class of applications (banking, airline reservation.) SP has developed techniques for capacity modeling, scheduling for latency, memory, and throughput, and load shedding techniques for improving latency with error bounds. SP and CEP are currently used in many diverse application domains.

Currently we are witnessing a similar explosion in inexpensive devices (similar to sensors around 1998) that are capable of generating continuous 24/7 image/video streams. Dash/Personal cams, Mobile phones, unmanned aerial vehicles (or UAVs), and Google glasses are examples of devices that can generate continuous, large amounts of videos either for surveillance or for security or for personal interest. This has brought about a requirement that is somewhat similar to the difference in requirements between traditional database processing vs. stream data processing. Traditional approach to image and video analysis (IAP), primarily customized, dealt with small amounts of videos, and were primarily analyzed for forensics. They do not support the kind of querying that is needed for analyzing large video streams and move towards real-time response.

Image and Video Analysis (IAP) has been researched much longer than the above two and has a much larger body of research that the above two. A large number of techniques – pattern recognition-based, machine learning- and artificial neural network-based – have been developed for processing image or video to characterize objects captured with different camera angles, lighting effects, color differences. Techniques have been developed for object identification and re-identification. Feature extraction, segmentation, and separation of background from foreground use numerous algorithms and machine learning techniques to deal with specific application domains/contexts. Also, the input quality and characteristics vary quite significantly among video streams (based on frame rate, pixel density, and appliances used, etc.) which necessitates the use of specific algorithms and approaches. Of course, the image processing technology is continuously improving as also the collection technology. Object tracking, video summarization, as well as specific and unlikely events detection have been addressed as well. Indexing techniques for videos based on extracted contents for search has been developed as well.

Based on the above observations and requirements, we postulate that both CEP and SP can be effectively adapted for analyzing videos in a different way than how it is being done currently. surprisingly, *CEP and SP technologies have not been leveraged or exploited for video analysis*, to the best of our knowledge. Analysis in this domains seems to require complex event processing techniques, stream processing techniques as well as techniques developed for satisfying QoS requirements of real-time applications. Video analysis can be viewed as stream processing of video frames from which *relevant contents* have been extracted. Of course, what you can query depends both on the amount of information extracted as well as the quality of information extracted. So, there is a dependency (or a bound) on the type and accuracy of analysis using the proposed approach. Extraction of relevant video (or frame) characteristics are process-intensive and requires computations that are very different from those used on categorical and numeric data (for example, computing histograms, bounding boxes, feature vectors, SVM, PCA etc.) Moreover, modeling of complex situations using extracted events and aggregation of extracted data from videos is also not straightforward. Furthermore, image or video analysis has primarily focused on customized solutions (rather than extensible ones) for specific problems (e.g., object identification, re-identification, object classification, overlaps of objects, tracking object movements.)

We believe that this approch can even be useful beyond video analysis. For example, *fusion* requires collecting, aggregating, and analyzing disparate data types (e.g., video, sensor, audio, call reports) in order to make informed decisions. If it is a real-time or near real-time decision making (surveillance as compared to forensics where archived data is analyzed), additional real-time processing and interactive requirements become critical. We believe that our approach of using CEP and SP, and leveraging this with the state-of-the-art image and video extraction techniques can bring video component to the kind of analysis that is done with other data types. Fusion depends on this versatility. We view our proposed work as a necessary precursor for addressing the fusion as a query processing problem and to add QoS and real-time support.

1.1 Focus of This work

The focus of this work is to leverage the current state-of-the-art in IAP – specifically, video content extraction – to identify and extract appropriate and useful generic contents to detect primitive events for the video domain. Using the stream processing approach, we will explore the class of primitive events that can be detected using current or added stream operators. We postulate that some of the primitive events can be detected using stream processing techniques on generic content extracted from video frames. However, other primitive events may have to be detected directly from a video using sophisticated techniques currently used for video analysis. The next step is to use and/or develop event operators for composing "interesting events" for this domain from extracted and/or detected primitive events. For example, static object extraction and identification have to be done using image processing techniques where as re-identification can be incorporated into stream processing in terms of similarity of appropriate feature vectors over a window specification. We will focus on identifying a class of primitive events using stream processing and composing them with appropriate operators for detecting higher level or "interesting" domain events. We will develop necessary spatial and/or temporal event operators for composing higher level events. we will also examine the current notion of windows used to specify queries that are not currently possible by the customized video processing approaches. We will extend it to cover a wide class of queries that correspond to situations of interest. The work proposed here comes under the sub-category of Human In the LOOP (HIL) according to [1].

In this report, we first analyze the current state-of-the-art in image/video processing to draw boundaries on what (and how much) should be done as part of image/video processing and what can be realistically relegated to stream and complex event processing. We then analyze the current state-of-the-art in stream and complex event processing to determine whether there is any advantage to combining them (rather than using them separately) to detect composite events as well as express a wider class of situations as continuous queries. For this, we first identify the class of primitive events that can be detected from generic contents of video frames. Then we develop event operators for composing these primitive events into higher level events for taking actions. Finally, we combine all of these to express situations on video that may require both stream as well as event processing. We also analyze the limitations of current window specification for video analysis and propose extensions to mitigate them.

The distinction of what should be done by image processing and what should be addressed as part of stream/complex event processing is important for several reasons: i) to leverage current video extraction techniques (and even the future ones) to the fullest extent rather than duplicating

them, ii) to identify the class of queries that cannot be expressed or are not detectable using current customized video processing techniques, and iii) to leverage a combination of stream and event processing for video analysis to complement what has been done in image/video processing.

The remainder of the reportis organized as follows. Section 2 will discuss relevant work to the focus problems of this report. In Section 3, we elaborate on the different steps to be taken as challenges to achieve continuous query processing over video contents. Section 4 will discuss the need for detecting primitive events using traditional video processing techniques and some non-primitive events using stream processing techniques. It will discuss stream and event operators for specifying non-primitive event computations and their composition, respectively. It will further elaborate their semantics, and algorithms for computation. It will also provide examples of situations that will require both stream and event processing capabilities. Section 5 will discuss requirements of data representation and propose an extension to the relational data representation. In Section 6, we discuss the language (CQL-VA) for expressing situations and their evaluation. Section 7 discussion optimization of CQL-VA expressions. Section 8 will discuss how stream and complex event processing architectures can be combined into an end-to-end integrated architecture that can be used for continuous query processing of videos. Section 9 has conclusions.

2 Relevant Work

The amount of literature on image and video analysis, stream data processing, and complex event processing is overwhelming. Hence, instead if reviewing the literature as a whole, we have analyzed portions of those research that are relevant to the focus of this work. Briefly, in image and video analysis (IVA), we have looked at the current status of object extraction, object identification and re-identification from frames of a video as well as complete video analysis for performing summaries based on events in specific domains (such as baseball, football), one-of-kind events, and other forms of tracking. In stream processing (SP), we have focused on types of windows and their specification as well as their current limitations. In complex event processing (CEP), we have analyzed computation/detection of primitive events using stream data processing and the composition of primitive events into higher level or "interesting" events using event operators. We summarize each below.

2.1 Image and Video Analysis (IVA)

The NSF report[1] of the workshop held in 2014 provides a good assessment of the current state-of-the-art in image and video processing including what are considered as solved problems, what are considered as problems that require near term and long term investments. This report further analyzes these in nine categories that cover the whole spectrum of work in image/video processing. In this section, we summarize some of the conclusions from that report that are relevant to this work. Specifically, Human In The Loop (HIL) research according to [1] includes all the aspects of image and video analysis (IVA) in which people are actively involved.

For our purpose of querying video contents, object identification, re-identification, and the ability to detect primitive and composite events are important. Also the ability to correlate between meaningful camera streams such as entry and exit camera streams for the *same* building is important for expressing a class of queries. Extracted content accuracy will play a critical role with respect to the accuracy of the answers to queries evaluated on those contents. For example, if two overlapped objects are not differentiated at extraction time, queries on count or match are not likely to give accurate results. Similarly, our approach to detect that an object turned around may depend on events such as moving forward, making turns and them moving forward. if these "primitive events" (whether detected directly or using stream processing on a sequence of frame contents) are not properly detected then the higher level event will also be not detected correctly.

Of the problems that need long term investments according to the NSF report, video summarization, visual analysis of large-scale imagery, *person re-identification*, human activity understanding in a video, semantic summarization are mentioned. This should be kept in mind for this attempted work which is a first of its kind as for as general-purpose querying is concerned. It also mentions the unavailability of data sets as well as privacy concerns of the contents of the data sets. Hence, you see a proliferation of customized data sets used for research and hence the difficulty in reproducing results.

The NSF report also points out that some of the problems that have been solved (e.g., triage of summarization) using constrained videos are not solved in an unconstrained setting that is becoming common place due to images and videos taken from personal devices (which are unconstrained.) Related to unconstrained environments, Re-identification of a person in a crowd is a

challenge. This problem is tackled in [2] to identify the same person viewed by disjoint cameras at different time instants and locations. A Landmark-Based model (LBM) is used along with movement extrapolation using landmarks and extracting features from the upper body which is less likely to be occluded to overcome the difficulties associated with non-overlapping cameras and spatio-temporal calibration needed. This indicates the difficulties of re-identification in a unconstrained setting.

Another recent survey [3] only focuses on the re-identification problem and the techniques as well as difficulties associated with this problem even in a constrained setting. We draw upon these to identify what additional work is needed to maximize the scope of our work. We have also reviewed other work that specifically addresses different kinds of event/action detection in video streams. We will summarize what we can benefit from and what may be needed in the future to leverage content extraction for our proposed analysis. To quote from [3] "most of the work in person re-id leverages clothing appearance based features designed for short period re-id and is evaluated in closed set re-id scenarios. The issue of long-period re-id is entirely unexplored and open set re-id is not completely tackled".

Event detection and summarization of sports videos is discussed in [4]. This work generalizes "play" detection to multiple sports (football, baseball, and sumo wrestling) instead of generating summaries of individual sports as is done in [5] for basketball, [6] for soccer, and [7] for baseball. Summarization here corresponds to objectively modeling and detecting every action that is *essential* to the game video being summarized. A compact summary (segments of video containing the all the plays) is generated based on the notion of plays for each type of game.

To appreciate the difficulty of extracting events from videos, it is interesting to see how a variety of techniques have been employed for this purpose. For example, [8] discuss key event detection in video using Automatic Speech Recognition (ASR), external corpora, and visual data. For this work, an event is something that happens at a certain moment in time and at a certain location possibly involving different actors. Different types of events (announcement, ceremony, ...) are learned from manual, external corpora (wiki), and ASR on videos of British royal weddings.

There have been some attempts to move towards the approach proposed in this report. The work by Aved [9, 10] 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.

A stream processing layer receives subquery evaluation streams from spatial processing layer and computes final query results. A query language (LVQL along the lines of SQL) has been proposed for specifying spatio-temporal queries. An object is considered as the fundamental component/unit of an event specification. Three classes of objects – static, dynamic, and cross-camera dynamic are handled by the system. A query is an action specification on an event specification. Spatial operators, such as "appear", "north", "inside", "meet" and others are supported. Temporal operators, such as "meets" and "before" are supported. Traditional Boolean operators and, or, not are supported as well. Bounding boxes can also be specified and used for trajectory detection.

An informatics-based approach to object tracking is used in this work. An LVQL query is

decomposed into subqueries which are used for optimization (common subqueries being processed only once) and processing. As appropriate, subqueries are pushed to the spatial layer.

A sample query in LVQL looks like:

```
Action 'q3' on Before(
West(c2.*, c2.s565b46, 10),
North(c2.s565b46, c2.*,250), 120);
```

the above is a translation of an intuitive query using operators. A query plan can be generated and processed for computing the intent of the query.

What we are proposing is a generalization of the above approach to bring in non-procedural operators, window constructs, and the full force of stream and complex event processing to Video analysis.

2.2 Complex Event Processing (CEP)

Complex event processing has seen a resurgence although the need for events, rules, and triggers were realized more than two decades ago. The foundation for complex event processing, in terms of event specification languages, their semantics, event detection algorithms, and integration of event processing with RDBMSs (as triggers) as well as stand-alone applications, has been well-established. The advent of stream processing has strengthened the applicability of complex event processing and alerts/notifications in environments that were never considered earlier.

Without differentiating between the event specification languages and the systems that included events and triggers, several event specification languages and systems have been developed. These systems focused only on complex event processing as stream processing came about much later: ACOOD [11], ADAM [12], Alert [13], A-RDL [14], Ariel [15, 16], Chimera [17, 18, 19], COMPOSE [20], EXACT [21], HiPAC [22, 23, 24], NAOS [25], ODE [26, 27], Postgres [28], REACH [29, 30], Rock & Roll [31, 32], SAMOS [33, 34], Sentinel [35, 36], SEQ [37], Snoop [38, 39], STARBURST [40, 41], TriGs [42, 43], UBILAB [44], and others [45, 46]. A comprehensive description of some of the above systems can be found in [47, 48] and an exhaustive annotated bibliography on active databases up to 1994 can be found in [49].

Once stream processing need was identified, many event processing work tried to combine them in various ways. Cayuga is proposed as general purpose, scalable event processing model [50, 51]. Cayuga proposes an algebra for expressing complex event patterns. It is claimed to incorporate several novel system design and implementation issues, focusing on Cayugas query processor, its indexing approach, handling of simultaneous events, and specialized garbage collecting. SASE [52, 53] is a complex event processing system that performs data to information transformations over real-time streams. It is targeted for filtering data (e.g., focus is on RFID applications) and correlating them for complex pattern detection and transformation to events that provide meaningful, actionable information to end applications. SASE has: designed a complex event language for specifying application logic for such transformation, devised new query processing techniques to efficiently implement the language, and developed a comprehensive system that collects, cleans, and processes RFID data for delivery of relevant, timely information as well as storing necessary data for future querying.

The operators of SASE is very similar to that of Snoop including the negation operation with a slight twist (within time T is included instead if start and end events in Snoop that can also be temporal events.) The structure is somewhat similar to SQL in that it has a set of event patterns, a where clause for attribute filtering and a within window. Although sliding windows are mentioned, there is no example in the paper as to how to represent it. SASE takes a sequence of event inputs and outputs a sequence of events that satisfy the pattern and the attributes that are requested. The formal semantics of the language is specified algebraically with a striking similarity to the semantics expressed in Snoop. We believe that some of the operators of SASE, such as SEQ_WITHOUT can be easily expressed in Snoop using its negation operator. The Any operator of SASE is also the same as the ANY operator of Snoop (although it is not a primitive operator and can be expressed using other operators). One of the limitations explicitly mentioned for SASE does not exist for Snoop as a complex event can be used in place of an event in any of the Snoop operators.

Several extensions have been proposed to earlier work on events: interval-based event semantics [54], event generalization & enhanced event consumption modes [55, 56, 57], and the use of the event language Snoop in novel applications [58]. The novel applications include role-based access control, expressive pattern searching over text streams & stored documents, and monitoring of web pages for customized changes.

2.3 Stream Data Processing (SP)

The motivation for stream data processing was somewhat different from that of event processing. The basic idea was to reduce large amounts of raw data being generated by sensors at some rate into some aggregate information/knowledge that is useful for further processing. For example, detection of 10% increase in a specific stock price value within a specified period (as many times as it happens in that interval) can provide information that is not discernible from raw data. This can be even more meaningful if it happens in conjunction with another stock going down/up during the same interval. However, these need to be detected in real-time in order to be meaningful in some applications. At the same time, for many of these applications, the raw data around a temporal interval is meaningful rather than values that are far apart temporally. Hence the notion of windows was added to support these window-based aggregates and all the relational operators were re-examined for computation in a window-based query specification. The proposed new architecture removed the high latency disk and developed scheduling algorithms to optimize various QoS requirements.

2.3.1 Arora and Borealis

Aurora [59, 60, 61, 62] is a system for managing data streams for monitoring applications. Continuous queries in Aurora are specified in terms of a dataflow diagram consisting of boxes and arrows (using a GUI). It uses SQuAL (Stream Query ALgebra) consisting of several primitive operators for expressing stream processing requirements. Aurora supports continuous queries, views, and *ad hoc* queries using the same conceptual building blocks.

Borealis [63, 64, 65, 66, 67, 68, 69] is a second generation system that addresses the problems of distributed operators for stream processing, dynamic revision of query results, dynamic query modification, and flexible and highly-scalable optimization. Load management in distributed stream processing systems is discussed in [70, 71, 72].

StreamBase [73], a commercial real-time, stream and complex event processing system, is based on Aurora. As of June 2013, TIBCO has acquired StreamBase.

2.3.2 STREAM

Stanford Stream Data Manager (or *STREAM*) [74, 75, 76] is a general-purpose data stream management system that address problems ranging from basic theoretical results to implementing a comprehensive prototype. The Chain scheduling algorithm for evaluating conjunctive queries with arithmetic comparison using a bounded amount of memory is presented in [77, 78].

Continuous Query Language (CQL) is a declarative language developed as an extension of SQL that includes streams and relations and additional constructs for specifying sliding `window` and sampling. CQL has been implemented as part of the STREAM prototype. The denotational semantics of CQL is presented in [79]. An abstract semantics for continuous queries based on formal definitions of streams & relations and the concrete language CQL that instantiates the abstract semantics using SQL is presented in [80, 81].

Other issues addressed as part of this project include: adaptive filters for continuous queries over distributed data streams [82], resource sharing in continuous sliding-`window` aggregates [83], resource management and approximate answers in a DSMS [84], and adaptive query caching [85].

2.3.3 MavStream

`MavStream` [86, 87, 88, 89, 90] has addressed stream processing holistically from a QoS perspective. The `MavHome` project at UTA [91, 92] is a multi-disciplinary project involving machine learning, predicting movements of occupants in a smart space, real-time processing of stream data, and automating the smart space/environment for: security, energy optimization, and assistance by predicting the behavior of occupants.

Starting with the intent of capacity planning and QoS verification, individual relational operators (e.g., `select`, `project`, `join`) initially, and continuous queries later, were modeled using queuing theory. Tuple latency and memory usage are computed using closed-form solutions [93]. Other operators can be modeled using this approach. The Path Capacity (PC) scheduling strategy [94] aims to minimize the overall tuple latency in contrast to minimizing the memory requirement. Several variants of the PC strategy have been proposed that can schedule an operator or a an operator path, or something in between. Of the variants proposed, based on their characteristics, the simplified segment strategy and the threshold strategy are better-suited for a DSMS. The optimal properties of the PC and the Memory Optimal Segment (MOS) strategies are established in the context of shared subqueries (in a multiple CQ processing system) as compared with the memory optimal property of the Chain strategy [95, 96] established for queries with no shared subexpressions. Load shedding [97, 89, 98] addresses the ideal location of load shedders among the alternative options, optimal placement based on weights computed statically (and refined at run-time, if necessary), activation/deactivation of load shedders at run-time and the amount of load to be shed for both random and semantic load shedders.

A DSMS prototype (`MavStream`) has been designed and implemented [99, 98, 100, 87, 101, 102, 103, 104] based on the algorithms and approaches developed in the `MavHome` project.

2.3.4 Others

Several other stream processing work include: *TelegraphCQ* [105, 106, 107, 108], Gigascope [109, 110], StatStream [111], *Tribeca* [112, 113], *Tapestry* [114, 115],

2.4 Discussion

The challenge for this work is to understand the capabilities and limitations of each of the participating techniques and based on that understanding determine the class of situations that can be expressed and detected on videos using the contents extracted from a video. It is important to identify: i) the types of aggregations on raw data (corresponding to the extracted contents) that are meaningful, ii) the class of primitive events that are meaningful in this domain, and iii) how to compose them to detect higher-level (or interesting) events. Once they are identified, it is equally important to flesh out the extensions needed in terms of operators, window constructs and whether using some primitive events directly will be better than computing them from raw data.

It is also important to understand that this being a novel approach and hence has to be developed in stages to gain insights along the way. Hence, it is not be likely that we will be able to support a very wide variety of situations to start with and should not be evaluated on that basis. Identifying boundaries of what can be done in the short, near, and long term is extremely important for this to become a research area where both groups can collaborate meaningfully and gain from each other.

Finally, there are some fundamental differences between traditional stream processing and video analysis. 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 in the context of Database Management Systems (DBMSs). For example, operators such as SELECT and JOIN have been optimized for over 3 decades for various scenarios and architectures. 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 was theoretically unbounded and storing and processing introduced too much latency. In addition, QoS needs of stream data processing was completely lacking in a traditional DBMS.

Importantly, there is significant amount of pre-processing required for extracting the ”proper contents” of a video as stream data in contrast to traditional stream data generated directly from sensors. This extraction is likely to lose or ignore some information (such as the background, spatial relation among objects) that will limit the class of situations that can be supported/detected.

3 Challenges

The goal of this project is to augment the work in traditional image and video analysis to support video analytics in different ways. This work is not intended to support arbitrary queries on arbitrary videos. Specifically, we intend to explore the class of *queries* that are meaningful on videos that correspond to surveillance, public safety monitoring, security monitoring, and a few other contexts. Although there is a lot of research on processing videos, querying extracted video contents in a generic manner has not received much attention. Video contents have been extracted and stored for *searching* which is different from *querying*. Querying, especially in a non-procedural manner, has low learning curve and allows the formulation of a large number of alternatives (as opposed to searching) depending upon the operators supported and their compositions. Querying also supports "what-if" queries to look for situations that were not initially considered. For example, one could query for the instances of the car moving before formulating the query about someone being picked up by the car. Hopefully, the class of queries supported can be expanded as we understand how to leverage extant IVA techniques to extract the right contents and represent it in a way to facilitate querying. Identifying challenges will also shed light on what "need" to be extracted in order to process some class of queries providing valuable input to the research issues that can be addressed by the IVA community.

Below, we formulate the challenges by using the types of situations we want to express and compute rather than doing it in the abstract. This also helps in understanding the limitations of this approach as well. Consider two streams of videos from two cameras – one for entry and one for exit, the following queries can be formulated on the footage[1]

[Situations – Set 1]

1. How many people/cars entered the building/parking lot every hour on the hour or between a specified interval or on a particular day
2. Identify the slowest hour of the day
3. List people/cars who stayed for less than an hour in the building/parking lot
4. List the average duration a person/car stayed at the property on Mondays/weekends
5. indicate when 2 different people (images given) enter or leave the building/parking lot within *n* minutes of each other
6. Indicate when a specific person (whose image is given) entered the building
7. Did these two individuals enter and exit the building within 5 minutes of each other?

The above queries are of aggregation types where information from the video is aggregated on different temporal dimensions. It is also possible to ask queries that include spatial dimension as well as both spatial and temporal dimensions. For example, it is possible to extend the above class of queries to:

[1]This could be for a building, mall, check post, or a parking lot etc. and the assumption is that other mechanisms are not available except acquiring a video stream.

[Situations – Set 2]

1. Did a person go up to the check post and turn around (or did not cross)?
2. Did two people bump into each other?
3. Did two people exchange any object?
4. Was a person picked up by a vehicle?
5. Did a specific person cross the bridge?

Note that the above queries are some what different from the earlier ones and requires identification and extraction of various types of actions, such as walking, turning around, carrying an object, getting rid of an object etc. It is also possible to frame more complex queries on videos. For example, consider the following queries:

[Situations – Set 3]

1. How many times a base was stolen in this baseball game video footage?
2. how many home runs were hit by a specific player in this game?
3. List the running touchdowns in each quarter of a football game.

The above can apply to different games as well as different situations such as recognizing low tide, recognizing specific activities in a specific location with certain objects/background etc.

It is evident that the complexity of the queries and the types of processing needed is increasing with the three sets of queries shown above. Also, the type and quality of extractions needed from videos is equally important. Traditional image processing has not tried to address the class of queries in the first set. The second set of queries have been addressed in a custom manner (VIRAT being a good example and many research prototype, see Section 2 on related work) using techniques that are difficult to generalize to other situations. More recently, third set of queries have been addressed again in a customized manner.

In the rest of this section, we identify the different challenges that need to be addressed for progressing towards extracting, expressing, evaluating, and optimizing queries that correspond to various classes starting from the easiest and increasing in scope and difficulty.

Current approaches to support the above is to develop appropriate tools to answer each query individually in specific contexts. This is the prevailing approach taken by the image and video processing community. This requires a prior knowledge of the queries that need to be processed and as such has a limitation on formulating similar, new queries. Of course the queries can be parameterized to some extent to provide some flexibility, but the basic limitation, in terms of the number and the kinds of queries that can be asked, remains.

An alternative approach taken in this report is to develop an expressive query language for expressing the classes of queries shown earlier. This will expand on the scope of queries that can

be asked. However, this approach also assumes a data representation over which these queries are expressed. A formal basis is also needed for defining the semantics of operators clearly and unambiguously describing the computation to be performed. As video itself is *not* a representation on which queries can be processed, a video need to be transformed into an appropriate representation.

Below, we identify the challenges and briefly indicate what each challenge entails for supporting our hypothesis. Subsequent sections make an effort to address these challenges based upon the current state of the art in image and Video Analysis and Stream & complex event processing

3.1 What to Extract From a Video

As the saying goes "a picture is worth a thousand words". A video captures a lot of information: objects – moving in spatial and temporal dimension and static, their spatial relationships over time, background objects and their changes, etc. It is not possible to extract all of this and their relationships. Most approaches to extract content from a video focus on the key contents thus leading to loss of information contained in a video. Hence, it is important to identify what "need" be extracted as well as what "can" be extracted using the current state of the art.

3.1.1 Primitive Contents

We need to identify the information that is relevant for the class of queries to be evaluated and can be extracted from a video using currently available techniques. We assume that individual objects from a frame can be extracted, objects can be identified into predefined types[2] and other relevant information such as a bounding box of each object, and quadrant location of the object can be extracted as part of image and video analysis[3]. If it is possible to extract and quantify some background information, that can be incorporated into the data representation as part of the extracted information from the video. For example, a bridge, check post, or mountain in the background can be associated with all the objects extracted from that frame. We also assume that some load shedding techniques can be used during extraction (at the frame level) without affecting the correct evaluation of the queries. For example, instead of processing all the 30 frames in a second, one (or two) frame per second may be enough for the types of information that is needed for processing these types of queries. This will expedite the processing of video streams without loss of information. Although this contraction can be achieved using queries, doing it as part of preprocessing is certainly more efficient.

To summarize, we will assume that during the *preprocessing phase*, the following content can be extracted as a minimum from a video:

a. Individual *moving* objects and their identification in each frame assuming a small set of object types (e.g., people, vehicle)

b. object bounding boxes and feature vectors (parameterized as needed)

[2]Such as a person, vehicle, hand-held object etc.

[3]Based on the NSF workshop report, we do understand that some of these are unsolved problems. For example, identifying individual objects from a crowd of objects is still problematic. We also understand that some background information is lost as part of this extraction as well as spatial relationships of objects in a frame or a sequence of frames.

c. Re-identification of objects in the video. If the video is very long, some limits may have to be imposed on the re-identification duration.

d. any static object (e.g., background objects) that can be extracted and identified.

3.1.2 Non-Primitive Contents

In addition to the above, to answer queries in Set 2, some activities (also called events) may have to be detected using image processing techniques as part of pre-processing. For example, an object moving forward, object turning right or left, overlap of different objects over a short period may be needed for composing higher-level actions or events. Initially, we assume that the following can also be extracted from a video using available image and video preprocessing techniques:

a. Direction in which an object is moving (forward, left, right or backwards)

b. Two objects overlapping in a video from the same camera

We will extend this list based on the capabilities available in IVP. It may also be possible to identify some of the above events using a query on the primitive events extracted. We will outline how this can be done. This will provide a choice for content extraction. We can also evaluate the differences with these approaches with respect to accuracy, false positives etc,

3.2 How to Represent Extracted Data?

$$VS1 \quad (Camera\_id, \underline{Frame\#}, \underline{Obj\_id}, Feature\_Vector\ as\ [FV], \\ Bounding\_box\ as\ [BB], Timestamp\ as\ ts) \tag{3.1}$$

Traditional Relational representation does not allow multiple values (related or not) for an attribute due to the First Normal Form constraint forcing one to use multiple attributes for bounding boxes and feature vectors. This makes representation as well as expression of queries complicated and difficult to understand. It also makes computation and optimization of queries difficult and expensive. One drawback of the Relational data model is its inherent row-oriented computation; if one wants to compare elements from two rows, a join is needed to bring them into a single row.Another drawback is the

Camera_id	**Frame#**	**obj_id**	**[FV]**	**[BB]**	**ts**
1	1	1	[f1]	[0,0,10,12]	1
1	1	2	[f2]	[bb2]	1
1	1	3	[f3]	[bb3]	1
1	31	1	[f4]	[bb4]	2
1	31	2	[f5]	[bb5]	2
1	31	4	[f6]	[bb6]	2
1	61	2	[f7]	[bb7]	3
1	61	4	[f8]	[bb8]	3
1	61	5	[f9]	[bb9]	3
1	91	2	[f10]	[bb10]	4
1	91	4	[f11]	[bb11]	4
1	91	5	[f12]	[bb12]	4

Table 1: VS1: Video frame representation as a relation with vector attributes

lack of ordering for performing computations (as it is a set-based model). Both of the above are a problem for video content analysis as ordering is important to make sure frames are not in arbitrary order and also for ordering the sequence of bounding boxes for computing the direction of motion. Also, being able to compare attributes from multiple rows is also important. For example, to compute the number of frames (or duration) in which the same object appears, one needs consecutive frames. Otherwise, large numbers of joins need to be performed.

Hence, as a minimum, a vector (whose contents may themselves be vectors) need to be added to the basic relational representation is needed for attributes and the contents of these vectors can also be ordered on some (one or more attribute) values. With the assumption of converting attribute values to a vector, and optional ordering, *each object* extracted from a frame can be represented as a tuple. Assuming, multiple objects in a frame and 30 frames per second, same objects will be present in several tuples. We can reduce this redundancy as mentioned earlier by sampling the frames during preprocessing.

Tables 1 and 2 show a relational representation of video frames from two cameras corresponding to the schema shown in 3.1. Vectors are used for attributes value in this representation. Other meta data such as camera attributes can be include either as part of the relation or kept in a separate traditional relation with camera_id as the foreign attribute.

Given a video stream from a surveillance camera in a parking lot, consider the extracted information from four frames shown in Table 1 using the schema shown in 3.1. In this table, we are annotating a vector attribute by adding [and]. For example, feature vector attributes is shown as $[FV]$. The size of a feature vector actually depends on its type (e.g., histogram , SIFT, or SURF.) Similarly, a bounding box is represented using coordinates of the frame (if the camera is fixed) or can be normalized for the motion of the camera if it is known. Defining computations over these attributes need to be investigated further as they are important for identifying situations. For example, two overlapping bounding boxes where the objects are persons may indicate exchange of things which may be of significance.

Unlike the traditional relational representation, ordering is important for extracted video contents. For example, frame ordering is important to infer movements, timestamp ordering is important for inferring duration and if a data set is ordered on an attribute (or a set of attributes), one wants to arrange other attribute values in the same order if they are grouped. We will elaborate on dealing with these issues in subsequent sections.

Camera_id	**Frame#**	**obj_id**	**[FV]**	**[BB]**	**ts**
2	1	51	[f21]	[0,0,80,96]	31
2	1	62	[f22]	[bb11]	31
2	1	63	[f23]	[bb12]	31
2	31	55	[f24]	[bb13]	32
2	31	64	[f25]	[bb14]	32
2	31	65	[f26]	[bb15]	32
2	61	53	[f27]	[bb16]	33
2	61	66	[f28]	[bb17]	33
2	61	68	[f29]	[bb18]	33

Table 2: VS2: Video frame representation as a relation with vector attributes

Given the above representation of extracted data, if we need to compute consecutive frames in which the

same object appears, instead of doing multiple Cartesian products (the number of Cartesian products depends on the number of frames we need to compare), it is better to order and group to obtain values from different rows into an attribute as we can capture them as vectors for the value of an attribute.

The above representations do not capture non-primitive events if they are detected as part of preprocessing. In general, the above representation need to be further extended to accommodate video processing needs. Table 2 shows another table for a different camera.

3.3 Expressing Queries – Towards CQL-VA

Once the representation is determined, it is important to make sure that it is backwards compatible with operators already defined on the models which have been extended. We will start with CQL (Continuous Query Language) extended from SQL for processing queries on streams whose data types were compatible with the relational data types. CQL-VA will be an extension and integration of AQuery features [116] to support operators and computations that are needed for video analysis (VA). This will be elaborated in Section 6.

We will introduce operators for new data types introduced. Bounding boxes and feature vectors are examples of new attribute data types.For backward compatibility, we need to define the existing operators (such as SELECT, PROJECT,...) for these data types. In addition, their usage in the larger CQL-VA language need to be established as well.

We will also introduce vector operators for computations. For example, a forward movement can be computed similar to moving averages by considering consecutive bounding boxes and computing a trajectory.

We will also critically evaluate the current window mechanisms (tuple-, time-, and predicate-based) to identify additional mechanisms that may be needed for video content processing.

3.4 Evaluating CQL-VA Queries

Once the operators and their semantics are unambiguously defined, the next step is to develop algorithms for computing these operators. Closure property of the computation is important for composability of operators. Operator properties need to be established for transformations of expressions leading to efficient evaluation of the expression.

Similar to CQL processing, we assume extracted data in the form of streams and a you look only once (YOLO) prompting the adaptation of various strategies (e.g., scheduling alternatives for latency and memory usage and load shedding alternatives for bounding errors and improving latency) developed for stream data processing. These will be discussed in Section 6.

3.5 Optimization of CQL-VA Queries

Finally, optimization of CQL-VA expressions is important in order to improve latency and/or memory utilization. This optimization is traditionally cost-based. However, the cost here is not disk I/O as in traditional optimization, but need to be quantified in terms of QoS parameters such as latency, memory usage, and throughput. This will be discussed in Section 7.

3.6 Architectural Requirements

The above discussion indicates that both stream processing and complex event processing are needed for expressing and evaluating situations on video contents. It may also be possible to express some non-primitive (and even primitive) event detection using stream processing operators. Hence, there is a need for integrating independently developed (or available) stream and complex event processing systems into an integrated system.

We have developed, at UTA, a MavStream system for stream data processing which supports traditional relational data types and a separate complex event processing system (Snoop and Sentinel at UFL) which supports a number of event operators, their semantics and algorithms for their computation. In Section 8, we will elaborate on the architectural details of MavEStream, where we have not only extended the above two systems, but integrated them as well for situation analysis of videos. This needs to be further extended to accommodate the extensions proposed for video analytics.

4 Objects and Event Extraction/Detection

As mentioned earlier, a video has a number of frames per second and the same objects appear in multiple frames. Typically, each frame is processed for extracting content and a sequence of identified frames (using shot boundaries and others) are used for delimiting and extracting an action or an event o a play. Background information is not always extracted or used unless the processing warrants it. also, it is somewhat difficult to extract background information as most of the object detection is based on movement. Typically, Spatial relationships among objects (both background and otherwise) and change in background image/information may be used for delimiting scenes.

Hence, it is an important decision to determine what to extract as both the types of queries that cab be asked and the correctness of answers depends on the information extracted and their quality.

4.1 Primitive Event/Object Detection

At the least, objects, object types, bounding boxes, and their appropriate feature vectors need to be extracted from each frame. Extraction of moving objects, their bounding boxes, and feature vectors can be extracted with the current IVA techniques. Identifying object types (e.g., person, car, bridge) is not that easy [1], but doable for a limited class of objects under some constraints. Detection of stationary objects is still hard, error prone, and requires a different set of techniques. This is needed for cars parked in a parking lot, background objects that do not move such as check posts, bridges, buildings etc. Moving objects can be extracted either from each frame or from periodic frames (say every 30 frames as there are 30 frames per second and objects of our interest do not move very fast.) Each moving object from a frame can be represented as a tuple as shown in Section 3. If the camera is fixed and the background does not change (known from meta data), the same object and type information can be extracted and stored as a single tuple for the entire video using a schema that is camera dependent and not time or frame dependent as shown in Schema 4.1. Either more attributes can be added to capture multiple background objects or each unique object can be represented as a separate tuple,

$$BGRD1 \quad (\underline{Camera\_id}, \underline{Obj\_id}, Feature\_Vector\ as\ [FV], Bounding\_box\ as\ [BB]) \quad (4.1)$$

The next step is object labeling or object re-identification where we want to re-identify the *same object* across frames. Typically, a moving object (e.g., person, car) is present in several frames depending on the field of view, camera angle, and the pace of the moving object. Frame differencing works reasonable well for moving objects. Re-identification of moving objects can pose some problems depending on lighting conditions, posture, overlap with other objects, and occlusion. Feature vector choices and many techniques are available in IVA for moving object re-identification. If the number of unique objects in a video is not very large (tens and not thousands), it is possible to re-identify an object even if it appears again after a long time. If the number of unique objects in a video are very large, some temporal limits may be imposed on object re-identification.

We have considered whether object re-identification can be expressed as a continuous query based on the extracted frame data and using window constructs. Typically, re-identification involves quadratic comparisons and/or keeping and managing a pool of feature vector of unique objects identified so far. Although parts of this computation can be expressed as self-joins followed by computations on grouped objects from each frame, it may be easier to do it as part of

preprocessing. We may experiment with this approach to streamline the process and reduce pre-processing effort. For the time being, we assume the output shown in Table 1 in Section 3 is a result of preprocessing.

4.2 Non-primitive Event Detection

Using continuous queries on primitive events, is it possible to detect non-primitive events that can be further composed. For example, using the displacement of bounding boxes, it should be possible to detect whether an object is moving forward, sideways, or backwards from a frame of reference. It should also be possible to detect whether two objects in a video overlap or close to each other in one or more frames.

Since vector representations are not part of the traditional representation, we need to develop operators (spatial and temporal) for defining the above computations so that non-primitive events, such as moving forward, object turns around, object passes in close proximity to others can be detected and raised. Using these it may be possible to define complex or higher-level events such as object turned around, object was in close proximity with a vehicle etc. which are situations that one is interested in detecting. This can be further extended to situations such as hitting a home run, stealing a base etc.

It is also possible to detect some of these non-primitive events using image and video processing techniques during preprocessing. We need to evaluate which one is better and why. There may be some difference in the way it is represented and used depending on when and how it is detected. For example, it may be raised as an event if a continuous query is used for its detection. If it is detected during pre-processing, a suitable data representation is needed fro representing it. We discuss these further in the Section on data representation (Section 5.)

5 Representation Of Extracted Data

Before we introduce our data representation, operators, and array-based computations, it is useful to gain insight into using the SQL standard for expressing some of the situations listed in Section 3 using traditional relational data representation to understand why a richer representation is needed. For window- and order-based computation, the OVER clause introduced into SQL 2003 and onwards is used. Please refer to `https://www.postgresql.org/docs/9.1/static/tutorial-window.html`, `https://en.wikibooks.org/wiki/Structured_Query_Language/Window_functions`, and `https://community.modeanalytics.com/sql/tutorial/sql-window-functions/` for a tutorial on SQL 2003. It should be noted that not all of the specification are supported by all vendors and there may be some ambiguity in the interpretation of the constructs as well.

Briefly, the OVER clause can be used in a SELECT clause to aggregate using a window specification. A window function in SQL performs a calculation across a set of table rows that are somehow related to the current row. This is comparable to the type of calculation that can be done with an aggregate function. But unlike regular aggregate functions, use of a window function does not cause rows to become grouped into a single output row the rows retain their separate identities. Behind the scenes, the window function is able to access more than just the current row of the query result.

A window function call always contains an OVER clause directly following the window function's name and argument(s). This is what syntactically distinguishes it from a regular function or aggregate function. The OVER clause determines exactly how the rows of the query are split up for processing by the window function. The PARTITION BY list within OVER specifies dividing the rows into groups, or partitions, that share the same values of the PARTITION BY expression(s). For each row, the window function is computed across the rows that fall into the same partition as the current row.

You can also control the order in which rows are processed by window functions using ORDER BY within OVER. The window ORDER BY does not even have to match the order in which the rows are output. There is a notion of a frame which is defined with respect to the CURRENT ROW *as each row of the table is considered*. Several keywords can be used such as ROWS, RANGE, PRECEDING n, FOLLOWING n, PRECEDING UNBOUNDED etc. to formulate a frame. The best way to understand is that a *table* can be reduced into a *result set* (using the WHERE clause); each *result set* can be further *partitioned* using the PARTITION component within the OVER clause; further *frames* can be defined over each *partition* using ROWS and RANGE expressions to perform aggregates. Several different kinds of frames can be defined using the ROWS, RANGE, PRECEDING, FOLLOWING, and UNBOUNDED key words. The key word CURRENT ROW refers to the current row with respect to which a frame is defined. A restriction of the OVER clause is that it cannot be used with a GROUP BY clause. Also, joins are not possible over frames. Multiple tables can be used in the FROM clause as the OVER clause is applied to the result of teh WHERE clause (i.e., theoretically, after the Cartesian product and restriction by the WHERE clause.)

Below we use the OVER clause provided in SQL 2003 and beyond in Oracle version 10.2.0.4.0 (at UTA) to express the queries in **set 1** shown in Section 3 using the following tables that are

similar to the Tables 1 and 2.

All of the above queries are tested using the Oracle on Omega at UTA. For this, two sample relations were created and populated to mimic the extracted contents of videos from two cameras along the lines of the tables shown in Section 3. Importantly, it contains frame numbers (not all of them), timestamp (as integers), and object numbers assuming they have been re-identified during preprocessing. Bounding box and feature vectors are excluded for the time being as they are vectors. obj_match() function is simulated using a natural join on the obj_id attribute as equality.

Tables camera1 and camera2 were defined in Oracle using the following schema.

```
create table camera1 (
   cameraId1 int not null,
   frame1 int not null,
   objId1  int not null,
   ts1  int,
   primary key (frame1, objId1));

create table camera2 (
   cameraId2 int not null,
   frame2 int not null,
   objId2  int not null,
   ts2  int,
   primary key (frame2,objId2));
```

camera1 relation was populated as follows (actual output of select * from camera1; statement in SQL.

```
SQL> select * from camera1;

 CAMERAID1     FRAME1     OBJID1         TS1
---------- ---------- ---------- ----------
         1          1          1          1
         1          1          2          1
         1          1          3          1
         1          1          4          1
         1          1          5          1
         1         10          2          5
         1         10          3          5
         1         10          4          5
         1         10          5          5
         1         20          2         10
         1         20          3         10

 CAMERAID1     FRAME1     OBJID1         TS1
---------- ---------- ---------- ----------
```

1	20	4	10
1	20	5	10
1	30	3	15
1	30	4	15
1	30	5	15
1	40	6	20
1	40	4	20
1	40	5	20
1	50	6	25
1	50	5	25
1	60	10	30

CAMERAID1	FRAME1	OBJID1	TS1
1	60	11	30
1	60	12	30
1	70	10	35
1	70	11	35
1	70	12	35
1	80	10	40
1	80	11	40
1	80	12	40
1	90	11	45
1	90	12	45
1	100	12	50

CAMERAID1	FRAME1	OBJID1	TS1
1	100	13	50
1	100	14	50
1	100	15	50
1	100	16	50
1	110	14	55
1	110	15	55
1	110	16	55
1	120	14	60
1	120	15	60
1	120	16	60
1	130	14	65

CAMERAID1	FRAME1	OBJID1	TS1
1	130	15	65
1	130	16	65
1	140	14	70
1	140	15	70

```
          1         140          16          70
          1         150          16          75
          1         150          17          75
          1         150          18          75
          1         160          18          80
          1         170          18          85
          1         180          19          90

 CAMERAID1     FRAME1     OBJID1         TS1
---------- ---------- ---------- ----------
          1         180          20          90
          1         190          19          95
          1         190          20          95
          1         200          20         100

59 rows selected.
```

1. **How many people entered the building every hour on the hour or between a specified interval or on a particular day**[4]

   For formulating this query, we need window frame that is time-based as well as the ability to move the window frame arbitrarily. Both of these are not currently available with the OVER clause. Hence we use the ts attribute

   Note that the above query uses time instead of *n preceding ROWS* as that may not constitute an hour. Hence, we have used an hour window for each tuple by specifying time instead of the number of rows (this is strictly not allowed in SQL as we will see later in Section 5.2.) Also, there is no way to specify the start of a new frame as it is fixed to the next row/tuple. Hence, there is no way to advance the window by time or tuples to make them disjoint[5]. The above query will calculate an hour window for each tuple. In our case, there will be many rows with the same time value (unless we filter it as part of preprocessing) as there are 30 frames per second. Another limitation of the above query is that the DISTINCT is needed as the same object may appear (is certain to appear) in many frames.

   Number of distinct persons entering in a specified interval does not require a window mechanism and can be expressed as a simple query with COUNT as an aggregate and a WHERE clause as shown below.

```
SELECT MIN(ts) AS start_time, MAX(ts) AS end_time,
         COUNT(DISTINCT obj_id)
FROM VS1
WHERE ts >= 1pm and ts <= 2pm;
```

[4]For parking lot queries, entry and exit can be done similarly. However, if one wants to know the average time cars stay in the parking lot or duration of stay of a particular car or a slot, they have to be done differently.

[5]If the frame is a single row, they will be disjoint!

It is also possible to rank each hour (either regular or dense) based on the number of people to determine the best and worst intervals. For this the OVER clause is needed as shown below.

```
[Set1:Q1]

WITH head_count AS
  (SELECT FIRST_VALUE(ts)
         OVER (ORDER BY ts
           RANGE BETWEEN 1 hour AND CURRENT ROW) AS start_time,
       LAST_VALUE(ts) as end_time
         OVER (ORDER BY ts
           RANGE BETWEEN 1 hour AND CURRENT ROW) AS end_time,
       COUNT(DISTINCT obj_id)
         OVER (ORDER BY ts
           RANGE BETWEEN 1 hour AND CURRENT ROW) AS NumObjects
   FROM VS1
   WHERE ts >= 9am and ts <= 5pm
 )
 SELECT start_time, end_time,
    RANK() OVER(ORDER BY NumObjects DESC) rank
    DENSE_RANK() OVER(ORDER BY NumObjects DESC) Dense_rank
 FROM head_count;
```

2. **Identify the slowest hour of the day (with respect to the number of people entering)**

   The query [Set1:Q1] computes the count of the number of people in each hour over a sliding window. The slowest or busiest hour can be extracted by computing the minimum or maximum as shown below. Let Hourly_count be the result of [Set1:Q1] with the schema Hourly_count(start_time, end_time, NumObjects). A subquery is needed to project out the times corresponding to the desired value. Note that there may be multiple intervals with the maximum or minimum value.

```
WITH Hourly_count AS [SET1:Q1 above goes here]
SELECT start_time, end_time, NumObjects
FROM Hourly_count
WHERE NumObjects IN (SELECT MAX(NumObjects)
                     FROM Hourly_count);
```

3. **List people who stayed for less than an hour in the building/parking lot**

   This query needs (and assumes) 2 relations VS_enter and VS_exit – one for entering and one for exiting[6].

[6]If a single camera is recording both entering and exiting of objects, it is necessary, as part of preprocessing, to identify objects as entering or exiting using an attribute with 2 values. In that case, a single relation can be used and by partitioning the relation on that attribute this query can be expressed.

Although it is possible to join two computed relations each using window operators in SQL, to the best of my knowledge, it is not possible to perform joins on window frames generated for each relation (as only one can be used along with the OVER clause. This query, ideally, need to check (match or join) each object entering (with a timestamp ts1) with all objects exiting with timestamp ts to ts+1hr as a window frame. This window frame cannot be specified using another relation window frame.

An alternative is to recast (or rethink) this query in a different way. One can do a regular join and *then* check whether the time intervals were within the range expected. The following query does that.

```
SELECT VSE.ts as enter_time, VSX.ts as exit_time, VSE.Obj_id
FROM VS_entry VSE, VS_exit as VSX
WHERE obj_match(VSE.FV, VSX.FV) AND VSX.ts <= VSE.ts + 1 hr ;
```

In the above, the obj_match function is for comparing feature vectors to determine whether they match based on a threshold value of distance between the two feature vectors which can be included optionally as part of the function. As it is easy to notice, the above will perform join on *all pairs of tuples* (actually a Cartesian product) which is an over kill. This will not be efficient for large videos or when the monitoring is continuous.

4. **List the average duration a person/car stayed at the property on a given day**

   This query theoretically does not need window support as we need to compute an average. But here an interval need to be computed before computing the average.

```
SELECT avg(duration)
FROM (
   SELECT VSE.Obj_id, VSX.ts - VSE.ts as duration
   FROM VS_entry VSE, VS_exit as VSX
   WHERE obj_match(VSE.FV, VSX.FV) AND VSX.ts <= VSE.ts + 1 hr
     ) as obj_duration ;
```

5. **Notify when 2 different people (images given) enter or leave the building/parking lot within *n* minutes of each other**

   Again this query can be formulated as independent queries to extract their entry or exit times and then compared as shown below. We will illustrate the entering within *n* minutes. We will assume we have as Obj1_FV and Obj2_FV as the extracted feature vectors, respectively, from their image inputs[7].

```
SELECT   vs1.Obj_id, vs1.ts, vs2.Obj_id, vs2.ts
FROM     VSE vs1, VSE vs2
WHERE    vs1.FV == Obj1_FV AND vs2.FV == Obj2_FV
         AND (vs1.ts <= vs2.ts + n OR vs2.ts <= vs1 + n)
ORDER BY vs1.ts, vs2.ts ;
```

[7]This is the equivalent of relational algebraic select where a value of appropriate type needs to be provided for comparison.

The above does gives the required answer and may list the tuples multiple times. However, it has a join and it is not clear how it is optimized to avoid performing a Cartesian product first.

The above can also be expressed using derived tables in the FROM clause as follows:

```
SELECT   Obj1.Obj_id, Obj2.Obj_id
FROM     (SELECT *
          FROM VSE
          WHERE    vs1.FV = Obj1_FV) AS Obj1
         INNER JOIN
          (SELECT *
          FROM VSE
          WHERE VSE.FV = Obj2_FV) AS Obj2
         ON Obj1.ts <= Obj2.ts + n OR Obj2.ts <= Obj1 + n) ;
```

This is a better formulation as each derived table is computed first before taking the join as each derived table is likely to be much smaller than the original. The query is also more intuitive to understand.

6. **Indicate when a specific person (whose image is given) entered the building along with the time of entry**

   This is a simple one.

```
SELECT   vs1.Obj_id AS person, vs1.ts AS entered_at
FROM     VSE vs1
WHERE    vs1.FV = Obj1_FV ;
```

7. **Did these two individuals enter and exit the building within *n* minutes of each other?**

   Two people entering the building within a given period can be easily expressed as the following query. It has a self-join and will compare every pair of objects (or tuples)

```
SELECT   vs1.Obj_id, vs1.ts, vs2.Obj_id, vs2.ts
FROM     VSE vs1, VSE vs2
WHERE    vs1.FV = Obj1_FV AND vs2.FV == Obj2_FV
         AND (vs1.ts <= vs2.ts + n OR vs2.ts <= vs1 + n) ;
ORDER BY vs1.ts, vs2.ts
```

   The above can also be expressed using derived tables in the FROM clause. Now the result of entering within a specified period need to be compared with their exit with in a specific period. This can be done using derived tables as shown below.

```
SELECT Obj1_in, Obj1_out, Obj2_in, Obj2_out
FROM   (SELECT Obj1_in.Obj_id, Obj2_in.Obj_id, Obj1_in.ts - Obj2_in.ts
        FROM (SELECT *
```

```
        FROM VSE
        WHERE VSE.FV = Obj1_FV) AS Obj1_in
       INNER JOIN
        (SELECT *
         FROM VSE
         WHERE VSE.FV = Obj2_FV) AS Obj2_in
       ON Obj1_in.ts <= Obj2_in.ts + n OR Obj2_in.ts <= Obj1_in +
)
INNER JOIN
 (SELECT Obj1_out.Obj_id, Obj2_out.Obj_id,
                Obj1_out.ts - Obj2_out.ts
  FROM (SELECT *
        FROM VSX
        WHERE VSX.FV = Obj1_FV) AS Obj1_out
       INNER JOIN
        (SELECT *
         FROM VSX
         WHERE VSX.FV = Obj2_FV) AS Obj2_out
       ON Obj1_out.ts <= Obj2_out.ts + n AND
          Obj2_out.ts <= Obj2_out.ts
 )
 ON Obj1_in = Obj1_out AND Obj2_in = Obj2_out ;
```

This query uses the previous one and computes when the two objects are entering within specified duration and exiting specified duration separately and then checks for their identity. As is clear, the query is complex and somewhat difficult to understand. Also, it is not possible to express chronological pairwise comparison. If the same two objects enter and exit *more than once* within the specified time interval *all combination of pairs* will be listed *not just the chronological ones as expected.*

5.1 Discussion

From the above examples, query formulations (in alternative ways), and identifying the lack of our ability to express a few things in the current SQL, it is clear that the language as well as the representation need to be extended for situation analysis of videos. Either it is difficult to express or it is difficult to understand the expressions or optimization may not be possible in many cases. Hence the need for a richer data representation and query expressiveness.

In order to express queries in **Set 2**, we need to extract and represent events/actions as part of preprocessing. Or we need to compute them using continuous queries and output events corresponding to them using the primitive data extracted from the video. We will discuss the use of continuous query processing for detecting events/actions in Section 6.

The reason for the above exercise is to gain insights into what capabilities are lacking leading to the identifications of capabilities to be added for expressing the above query sets. The goal is to choose a data representation and operators that will help simplify query specification and hopefully their optimization as well.

5.2 Testing SET1 Using Oracle (Release 10.2.0.4.0)

All of the above queries are tested using the Oracle on Omega at UTA. For this, two sample relations were created and populated to mimic the extracted contents of videos from two cameras along the lines of the tables shown in Section 3. Importantly, it contains frame numbers (not all of them), timestamp (as integers), and object numbers assuming they have been re-identified during preprocessing. Bounding box and feature vectors are excluded for the time being as they are vectors. obj_match() function is simulated using a natural join on the obj_id attribute as equality.

Tables camera1 and camera2 were defined in Oracle using the following schema.

```
create table camera1 (
   cameraId1 int not null,
   frame1 int not null,
   objId1  int not null,
   ts1  int,
   primary key (frame1, objId1));

create table camera2 (
   cameraId2 int not null,
   frame2 int not null,
   objId2  int not null,
   ts2  int,
   primary key (frame2,objId2));
```

camera1 relation was populated as follows (actual output of select * from camera1; statement in SQL. Populated data and the result of SET1:Q1 are shown below.

```
SQL> select * from camera1;

 CAMERAID1     FRAME1     OBJID1         TS1
---------- ---------- ---------- ----------
         1          1          1          1
         1          1          2          1
         1          1          3          1
         1          1          4          1
         1          1          5          1
         1         10          2          5
         1         10          3          5
         1         10          4          5
         1         10          5          5
         1         20          2         10
         1         20          3         10

 CAMERAID1     FRAME1     OBJID1         TS1
```

1	20	4	10
1	20	5	10
1	30	3	15
1	30	4	15
1	30	5	15
1	40	6	20
1	40	4	20
1	40	5	20
1	50	6	25
1	50	5	25
1	60	10	30

CAMERAID1	FRAME1	OBJID1	TS1
1	60	11	30
1	60	12	30
1	70	10	35
1	70	11	35
1	70	12	35
1	80	10	40
1	80	11	40
1	80	12	40
1	90	11	45
1	90	12	45
1	100	12	50

CAMERAID1	FRAME1	OBJID1	TS1
1	100	13	50
1	100	14	50
1	100	15	50
1	100	16	50
1	110	14	55
1	110	15	55
1	110	16	55
1	120	14	60
1	120	15	60
1	120	16	60
1	130	14	65

CAMERAID1	FRAME1	OBJID1	TS1
1	130	15	65
1	130	16	65
1	140	14	70

```
         1        140         15         70
         1        140         16         70
         1        150         16         75
         1        150         17         75
         1        150         18         75
         1        160         18         80
         1        170         18         85
         1        180         19         90

 CAMERAID1     FRAME1     OBJID1        TS1
---------- ---------- ---------- ----------
         1        180         20         90
         1        190         19         95
         1        190         20         95
         1        200         20        100

59 rows selected.

SQL>
SQL>
SQL> --Query 1 for SET 1
SQL>
SQL> --How many people entered the building every hour on the hour or
SQL> -- between a specified interval or on a particular day
SQL>
SQL> -- DISTINCT is not allowd in OVER clause for some reason!
SQL>
SQL> SELECT FIRST_VALUE(ts1) OVER (ORDER BY ts1
  2   RANGE BETWEEN 60 PRECEDING AND CURRENT ROW)  start_time,
  3       LAST_VALUE(ts1)   OVER (ORDER BY ts1
  4   RANGE BETWEEN 60 PRECEDING AND CURRENT ROW)  end_time,
  5       COUNT(objId1) OVER (ORDER BY ts1
  6   RANGE BETWEEN 60 PRECEDING AND CURRENT ROW)  NumObjects
  7   FROM camera1
  8   WHERE ts1 >= 1 and ts1 <= 100  ;

START_TIME   END_TIME NUMOBJECTS
---------- ---------- ----------
         1          1          5
         1          1          5
         1          1          5
         1          1          5
         1          1          5
         1          5          9
         1          5          9
         1          5          9
```

```
         1          5          9
         1         10         13
         1         10         13

START_TIME   END_TIME NUMOBJECTS
---------- ---------- ----------
         1         10         13
         1         10         13
         1         15         16
         1         15         16
         1         15         16
         1         20         19
         1         20         19
         1         20         19
         1         25         21
         1         25         21
         1         30         24

START_TIME   END_TIME NUMOBJECTS
---------- ---------- ----------
         1         30         24
         1         30         24
         1         35         27
         1         35         27
         1         35         27
         1         40         30
         1         40         30
         1         40         30
         1         45         32
         1         45         32
         1         50         37

START_TIME   END_TIME NUMOBJECTS
---------- ---------- ----------
         1         50         37
         1         50         37
         1         50         37
         1         50         37
         1         55         40
         1         55         40
         1         55         40
         1         60         43
         1         60         43
         1         60         43
         5         65         41
```

```
START_TIME   END_TIME NUMOBJECTS
---------- ---------- ----------
         5         65         41
         5         65         41
        10         70         40
        10         70         40
        10         70         40
        15         75         39
        15         75         39
        15         75         39
        20         80         37
        25         85         35
        30         90         35

START_TIME   END_TIME NUMOBJECTS
---------- ---------- ----------
        30         90         35
        35         95         34
        35         95         34
        40        100         32

59 rows selected.

SQL>
SQL> -- variant to count distinct objects in an interval
SQL>
SQL> SELECT MIN(ts1) AS start_time, MAX(ts1) AS end_time,
  2   COUNT(DISTINCT objId1)
  3    FROM camera1
  4    WHERE ts1 >= 10 and ts1 <= 50;

START_TIME   END_TIME COUNT(DISTINCTOBJID1)
---------- ---------- --------------------
        10         50                   12

SQL>
SQL> SELECT MIN(ts1) AS start_time, MAX(ts1) AS end_time,
  2   COUNT(DISTINCT objId1)
  3    FROM camera1
  4    WHERE ts1 >= 51 and ts1 <= 100;

START_TIME   END_TIME COUNT(DISTINCTOBJID1)
---------- ---------- --------------------
        55        100                    7

SQL>
```

```
SQL> -- i know there are only 17 distinct objects
SQL>
SQL> SELECT MIN(ts1) AS start_time, MAX(ts1) AS end_time,
  2   COUNT(DISTINCT objId1), COUNT(objId1)
  3    FROM camera1;

START_TIME   END_TIME COUNT(DISTINCTOBJID1) COUNT(OBJID1)
---------- ---------- -------------------- -------------
         1        100                   17            59
```

We populated the table as it would be when videos are preprocessed to extract data. Each frame is likely to have multiple objects and the *same object* will appear in multiple consecutive frames even if one were to eliminate some frames. The table shows about 20 frames starting from frame 1 through frame 100. Similarly, timestamp is also a number from 1 to 100 indicating 100 minutes (or time units) of video.

The query corresponding to SET1:Q1 and its result are shown above. This is the closest one can come to expressing this query in SQL. As noted earlier, there are several problems here: i) same objects are counted multiple times as can be see from the last row – in 60 units of time from 40 to 100, there are 32 objects, but the DISTINCT objects are only 11. **This is because one cannot use the DISTINCT with COUNT as part of the window aggregate!**, ii) window cannot be advanced by more than one tuple. Hence, we get a sliding window which slides by 1 tuples and computes hourly count for each tuple and its previous hour. since there are 59 tuples in the table, you get 59 1 hour windows, and iii) attribute value-based frame specification does not seem to be possible. For example, cannot say that the frame should contain tuples whose timestamp value in in the range 1 hour.

Ranking can be done as shown earlier. Oracle query for regular and dense ranking are shown below with results.

```
SQL>
SQL> --ranking and dense ranking of the count of people for each interval
SQL> -- need to sort in descending order
SQL>
SQL> WITH head_count AS
  2    (SELECT FIRST_VALUE(ts1)
  3         OVER (ORDER BY ts1
  4   RANGE BETWEEN 60 PRECEDING AND CURRENT ROW) start_time,
  5       LAST_VALUE(ts1)
  6         OVER (ORDER BY ts1
  7   RANGE BETWEEN 60 PRECEDING AND CURRENT ROW) end_time,
  8       COUNT(objId1)
  9         OVER (ORDER BY ts1
 10   RANGE BETWEEN 60 PRECEDING AND CURRENT ROW) NumObjects
 11   FROM camera1
 12   WHERE ts1 >= 1 and ts1 <= 100
```

```
 13   )
 14   SELECT start_time as st, end_time as et, NumObjects asnobjs,
 15          RANK() OVER(ORDER BY NumObjects DESC) rank,
 16          DENSE_RANK() OVER(ORDER BY NumObjects DESC) dense_rank
 17   FROM head_count;

        ST          ET    ASNOBJS       RANK DENSE_RANK
---------- ---------- ---------- ---------- ----------
         1          60         43          1          1
         1          60         43          1          1
         1          60         43          1          1
         5          65         41          4          2
         5          65         41          4          2
         5          65         41          4          2
        10          70         40          7          3
         1          55         40          7          3
         1          55         40          7          3
        10          70         40          7          3
         1          55         40          7          3

        ST          ET    ASNOBJS       RANK DENSE_RANK
---------- ---------- ---------- ---------- ----------
        10          70         40          7          3
        15          75         39         13          4
        15          75         39         13          4
        15          75         39         13          4
        20          80         37         16          5
         1          50         37         16          5
         1          50         37         16          5
         1          50         37         16          5
         1          50         37         16          5
         1          50         37         16          5
        30          90         35         22          6

        ST          ET    ASNOBJS       RANK DENSE_RANK
---------- ---------- ---------- ---------- ----------
        30          90         35         22          6
        25          85         35         22          6
        35          95         34         25          7
        35          95         34         25          7
         1          45         32         27          8
         1          45         32         27          8
        40         100         32         27          8
         1          40         30         30          9
         1          40         30         30          9
         1          40         30         30          9
```

```
         1          35          27          33          10

        ST          ET    ASNOBJS        RANK DENSE_RANK
---------- ---------- ---------- ---------- ----------
         1          35          27          33          10
         1          35          27          33          10
         1          30          24          36          11
         1          30          24          36          11
         1          30          24          36          11
         1          25          21          39          12
         1          25          21          39          12
         1          20          19          41          13
         1          20          19          41          13
         1          20          19          41          13
         1          15          16          44          14

        ST          ET    ASNOBJS        RANK DENSE_RANK
---------- ---------- ---------- ---------- ----------
         1          15          16          44          14
         1          15          16          44          14
         1          10          13          47          15
         1          10          13          47          15
         1          10          13          47          15
         1          10          13          47          15
         1           5           9          51          16
         1           5           9          51          16
         1           5           9          51          16
         1           5           9          51          16
         1           1           5          55          17

        ST          ET    ASNOBJS        RANK DENSE_RANK
---------- ---------- ---------- ---------- ----------
         1           1           5          55          17
         1           1           5          55          17
         1           1           5          55          17
         1           1           5          55          17

59 rows selected.

SQL> spool off
```

The query and results for SET1:Q2 is shown below.

```
SQL> @q2
SQL> --data dumped to verify the query results
```

```
SQL>
SQL> --select * from camera1;
SQL>
SQL>
SQL> --Query 1 for SET 1
SQL>
SQL> -- Identify the slowest hour of the day (with respect to
SQL> -- the number of people entering)
SQL>
SQL> WITH Hourly_count AS
  2     (SELECT FIRST_VALUE(ts1)
  3       OVER (ORDER BY ts1 RANGE BETWEEN 60 PRECEDING AND CURRENT ROW)  st
  4     LAST_VALUE(ts1)
  5       OVER (ORDER BY ts1 RANGE BETWEEN 60 PRECEDING AND CURRENT ROW)  en
  6     COUNT(objId1)
  7       OVER (ORDER BY ts1 RANGE BETWEEN 60 PRECEDING AND CURRENT ROW)  Nu
  8       FROM camera1
  9       WHERE ts1 >= 1 and ts1 <= 100 )
 10  SELECT DISTINCT start_time, end_time, NumObjects as Busiest_hour
 11  FROM Hourly_count
 12  WHERE NumObjects IN (SELECT MAX(NumObjects)
 13     FROM      Hourly_count);

START_TIME   END_TIME BUSIEST_HOUR
---------- ---------- ------------
         1         60           43

SQL>
SQL>
SQL> spool off
```

By changing max to min, the slowest hour can be computed. By using the exit relation, same can be computed for exit instead of entry.

For queries 3 onwards, we need both entry and exit relations extracted from the corresponding videos. We have populated camera2 relation (whose schema is the same as camera1 and given earlier) to test the rest of the queries correctness under normal and boundary conditions. We assume that the timestamps (ts) on both the cameras are synchronized. For example, some objects have stayed for more than 1 hour; some objects have entered within n units of each other and exited within n units of each other. SET 1:Q3, its intermediate results anf dinal result are shown below.

```
SQL> @q3
SQL> --data dumped to verify the query results
SQL>
SQL> --select * from camera1;
SQL> select * from camera2;
```

```
 CAMERAID2     FRAME2     OBJID2        TS2
---------- ---------- ---------- ----------
         2          1        101          1
         2          1        102          1
         2          1        103          1
         2        100          2         34
         2        100        101         34
         2        100        102         34
         2        110          2         45
         2        110          3         45
         2        110        101         45
         2        110        102         45
         2        120          4         50

 CAMERAID2     FRAME2     OBJID2        TS2
---------- ---------- ---------- ----------
         2        120         11         50
         2        130         21         60
         2        130         22         60
         2        130         23         60
         2        140         24         70
         2        140         25         70
         2        140         26         70
         2        150          5         80
         2        150          6         80
         2        160          5         90
         2        160          6         90

 CAMERAID2     FRAME2     OBJID2        TS2
---------- ---------- ---------- ----------
         2        170         10        100
         2        170         11        100
         2        170         12        100

25 rows selected.

SQL>
SQL>
SQL> --Query 3 for SET 1
SQL>
SQL> -- list people whon stayed for less than 45 units of time in the buildi
SQL>
SQL> --intermediate query 1 : first frame of entry
SQL>
SQL> SELECT objId1, min(ts1) as enter_time
```

```
  2        FROM camera1
  3        GROUP BY objId1;

    OBJID1 ENTER_TIME
---------- ----------
         1          1
         6         20
        11         30
        13         50
         2          1
        14         50
        20         90
         4          1
         5          1
        17         75
         3          1

    OBJID1 ENTER_TIME
---------- ----------
        18         75
        10         30
        12         30
        15         50
        16         50
        19         90

17 rows selected.

SQL>
SQL> --intermediate query2 : first frame of exit
SQL>
SQL> SELECT objId2, max(ts2) as exit_time
  2        FROM camera2
  3        GROUP BY objId2;

    OBJID2  EXIT_TIME
---------- ----------
        22         60
        25         70
        11        100
         6         90
         2         45
        21         60
        26         70
       102         45
         4         50
```

```
        24         70
         5         90

    OBJID2  EXIT_TIME
---------- ----------
        23         60
       101         45
         3         45
       103          1
        10        100
        12        100

17 rows selected.

SQL>
SQL> --combined query  for less than 45 minutes
SQL>
SQL> WITH VSE as
  2       (SELECT objId1, min(ts1) as enter_time
  3        FROM camera1
  4        GROUP BY objId1),
  5     VSX as
  6       (SELECT objId2, max(ts2) as exit_time
  7        FROM camera2
  8        GROUP BY objId2)
  9  SELECT VSE.enter_time as entry_time, VSX.exit_time as exit_time, VSE.ob
 10  FROM VSE, VSX
 11  WHERE VSE.objId1 = VSX.objId2 AND VSX.exit_time <= VSE.enter_time + 45

ENTRY_TIME  EXIT_TIME     PERSON
---------- ---------- ----------
         1         45          2
         1         45          3

SQL>
SQL>
SQL>
SQL> spool off
```

Only 2 persons 2 and 3 stayed under 45 minutes. if one wants to list all those who stayed more than 45 minutes, the where condition can be changed appropriately.

SET 1:Q4 asks for the average duration a person stayed stayed in a building which requires calculation of intervals for each person and taking the average. As in the previous query, one interval needs to be calculated and used for each person as multiple frames contain the same person both at entry and exit. The Oracle query and its results are given below.

```
SQL> @q4
SQL> --data dumped to verify the query results
SQL>
SQL> --select * from camera1;
SQL> --select * from camera2;
SQL>
SQL>
SQL> --Query 4 for SET 1
SQL>
SQL> -- compute the average duration of stay for those who stayed under 45 u
SQL> -- can be changed to a day or any other interval!
SQL>
SQL> WITH VSE as
  2       (SELECT objId1, min(ts1) as enter_time
  3        FROM camera1
  4        GROUP BY objId1),
  5     VSX as
  6       (SELECT objId2, max(ts2) as exit_time
  7        FROM camera2
  8        GROUP BY objId2)
  9  SELECT avg(exit_time - enter_time)
 10  FROM VSE, VSX
 11  WHERE VSE.objId1 = VSX.objId2 AND VSX.exit_time  <= VSE.enter_time + 45

AVG(EXIT_TIME-ENTER_TIME)
-------------------------
                       44

SQL>
SQL> -- average of those who stayed for more than 45 mins
SQL>
SQL> WITH VSE as
  2       (SELECT objId1, min(ts1) as enter_time
  3        FROM camera1
  4        GROUP BY objId1),
  5     VSX as
  6       (SELECT objId2, max(ts2) as exit_time
  7        FROM camera2
  8        GROUP BY objId2)
  9  SELECT avg(exit_time - enter_time)
 10  FROM VSE, VSX
 11  WHERE VSE.objId1 = VSX.objId2 AND VSX.exit_time  >= VSE.enter_time + 45

AVG(EXIT_TIME-ENTER_TIME)
-------------------------
              69.6666667
```

```
SQL> spool off
```

SET 1:Q5 asks for 2 different people (image or object given) enter and[8] leave the building within n units of each other. We will use 30 units as we know persons 2 and 4 entered and exited within 30 units of each other. no other pairs did. The query and the results are given below.

```
SQL> @q5
SQL> --data dumped to verify the query results
SQL>
SQL> --select * from camera1;
SQL> --select * from camera2;
SQL>
SQL>
SQL> --Query 5 for SET 1
SQL>
SQL> -- indicate when 2 different people (images given) enter AND leave
SQL> -- the building within \textit{n} minutes of each other
SQL>
SQL> --the or version can be treated as two independent queries. otherwise,
SQL> -- a Cartesian product is unnecessarily taken!
SQL>
SQL> --intermediate query 1 : first frame of entry
SQL>
SQL> SELECT objId1, min(ts1) as enter_time
  2        FROM camera1
  3        GROUP BY objId1;

    OBJID1 ENTER_TIME
---------- ----------
         1          1
         6         20
        11         30
        13         50
         2          1
        14         50
        20         90
         4          1
         5          1
        17         75
         3          1

    OBJID1 ENTER_TIME
```

[8]The Or version of this query can be treated as two separate queries. Otherwise, a Cartesian product is unnecessarily taken producing large number of results.

```
---------- ----------
        18         75
        10         30
        12         30
        15         50
        16         50
        19         90

17 rows selected.

SQL>
SQL> --intermediate query2 : first frame of exit
SQL>
SQL> SELECT objId2, max(ts2) as exit_time
  2        FROM camera2
  3        GROUP BY objId2;

    OBJID2  EXIT_TIME
---------- ----------
        22         60
        25         70
        11        100
         6         90
         2         45
        21         60
        26         70
       102         45
         4         50
        24         70
         5         90

    OBJID2  EXIT_TIME
---------- ----------
        23         60
       101         45
         3         45
       103          1
        10        100
        12        100

17 rows selected.

SQL>
SQL> WITH VSE as
  2       (SELECT objId1, min(ts1) as enter_time
  3        FROM camera1
```

```
  4        GROUP BY objId1),
  5     VSX as
  6       (SELECT objId2, max(ts2) as exit_time
  7        FROM camera2
  8        GROUP BY objId2)
  9  SELECT e1.objId1, e1.enter_time, e2.objId1, e1.enter_time, x1.objId2, x
 10  FROM VSE e1, VSE e2, VSX x1, VSX x2
 11  WHERE (e1.objId1 = 2 AND e2.objId1 = 4 AND ABS(e1.enter_time - e2.ente
 12       AND (x1.objId2 = 2 AND x2.objId2 = 4 AND ABS(x1.exit_time - x2.exi

    OBJID1 ENTER_TIME     OBJID1 ENTER_TIME     OBJID2  EXIT_TIME     OBJID2
---------- ---------- ---------- ---------- ---------- ---------- ----------
 EXIT_TIME
----------
         2          1          4          1          2         45          4
        50


SQL>
SQL> -- these two people DID NOt enter AND exit within 4 or less mins of eac
SQL>
SQL> WITH VSE as
  2       (SELECT objId1, min(ts1) as enter_time
  3        FROM camera1
  4        GROUP BY objId1),
  5     VSX as
  6       (SELECT objId2, max(ts2) as exit_time
  7        FROM camera2
  8        GROUP BY objId2)
  9  SELECT e1.objId1, e1.enter_time, e2.objId1, e1.enter_time, x1.objId2, x
 10  FROM VSE e1, VSE e2, VSX x1, VSX x2
 11  WHERE (e1.objId1 = 2 AND e2.objId1 = 4 AND ABS(e1.enter_time - e2.ente
 12       AND (x1.objId2 = 2 AND x2.objId2 = 4 AND ABS(x1.exit_time - x2.exi

no rows selected

SQL> spool off
```

5.3 Proposed Data Representation

Tables 1 and 2 in Section 3 show representation of extracted video data with Vectors for attribute values. Both bounding boxes and feature vectors are represented using a vector although the feature vector requires elements that can be vectors and the size depends on the type of the feature vector. We also indicated the need for ordering and grouping of rows using one or more attributes in order to simplify query expression and its processing. For example, if we group Table 1 on attributes

Camera_id and Frame# and order on ts, we get Table 3. This computation can be expressed using extended SQL as

```
SELECT          *
FROM            VS1
ORDERING ON     ts
GROUP BY        Camera_id, Obj_id
```

The output will use the convention of vectors for attributes that became vectors as a result of this operation. We will drop the *Camera_id* attribute in the future (unless it is needed) from this table as it represents only one camera.

Camera_id	**[Frame#]**	**obj_id**	**[FV]**	**[BB]**	**[ts]**
1	[1,31]	1	[[f1],[f4]]	[][0,0,10,12],[bb4]]	[1,2]
1	[1,31,61,91]	2	[[f2],[f5],[f7],[f10]]	[[bb2],[bb7],[bb10]]	[1,2,3,4]
1	[1]	3	[f3]	[bb3]	[1]
1	[31,61,91]	4	[f5]	[bb6]	[2,3,4]
1	[61,91]	5	[f8]	[bb9]	[3,4]

Table 3: VS1: Video frame representation as a relation with vector attributes

With this representation, we can ask queries about sequences of values in a vector and also compute differences that are relevant to attributes (e.g., ts and may be frame#). It is also possible to use this representation to compute the direction of motion using bounding boxes and overlap among objects as well.

The next step is to compare the two representations and determine what classes of queries are easy to express and understand for each representation. Based on that, we may want to be able to go between these two representations (again using SQL) and choose the one that os appropriate for the query at a hand.

6 Express and Evaluate CQL-VA

Based on the exercise in the previous section, we have identified the following limitations if SQL even with using the OVER clause. They are:

1. Window specifications cannot move the upper and lower bounds arbitrarily
2. Although multiple OVER clauses can be specified on *different relations*, windowing clause is applied to the result of the FROM and WHERE clause, not for each relation *before* joining and applying the WHERE clause.
3. Window-based joins are not allowed under the current specification of the OVE clause
4. Window offsets between joins cannot be specified currently

One of the next steps (perhaps for the extension of this project) is to extend the OVER clause to accommodate the above limitations, define the semantics clearly, and propose a computation model for the proposed semantics.

The other remaining step is to introduce (or refine existing ones) new event operators in order to detect overlap and higher level events such as turning back, getting into a vehicle etc., their semantics and how to use them to combine with the above continuous queries. For this, we may use the queries to generate non-primitive events such as moving forward, going left, going backwards to determine even higher level events such as ”turned back” by composing a sequence of non-primitive events.

7 Optimizing CQL-VA

Once the OVER clause is extended, their efficient computation (or optimization) along with the rest of the operators of the query is the next step. Some of the optimizations that will be considered are : i) establishing mathematical properties of operators (e.g., commutativity, associativity, etc.), ii) re-ordering the sequence in which operators are computed, iii) pushing some predicates below the operators to reduce the amount of intermediate results generated, and iii) measuring the cost of computation using an appropriate metric.

In addition, since we are treating the input as a stream (albeit a stream with a fixed rate), scheduling and load shedding issues become important for real or near real-time requirements.

These will be addressed in the sequence indicated.

8 MavVStream Architecture

In the previous sections, we have established a need for both event processing, stream processing, as well as detecting some of the primitive events (from the extracted contents of videos) needed for composite event detection using stream processing techniques. We will extend the prototype (MavEStream), developed at UT Arlington, which has some of the above capabilities and identify further extensions in terms of new operators (both for event and stream processing), experimentation, and performance evaluation once the operators, their semantics, and situations are expressed using the operators developed in the previous section. There is likely to be additional issues that need to be addressed as part of complex event processing for supporting QoS requirements that are not currently available. The integrated event stream processing architecture (from [90]) shown in Figure 1 has four stages:

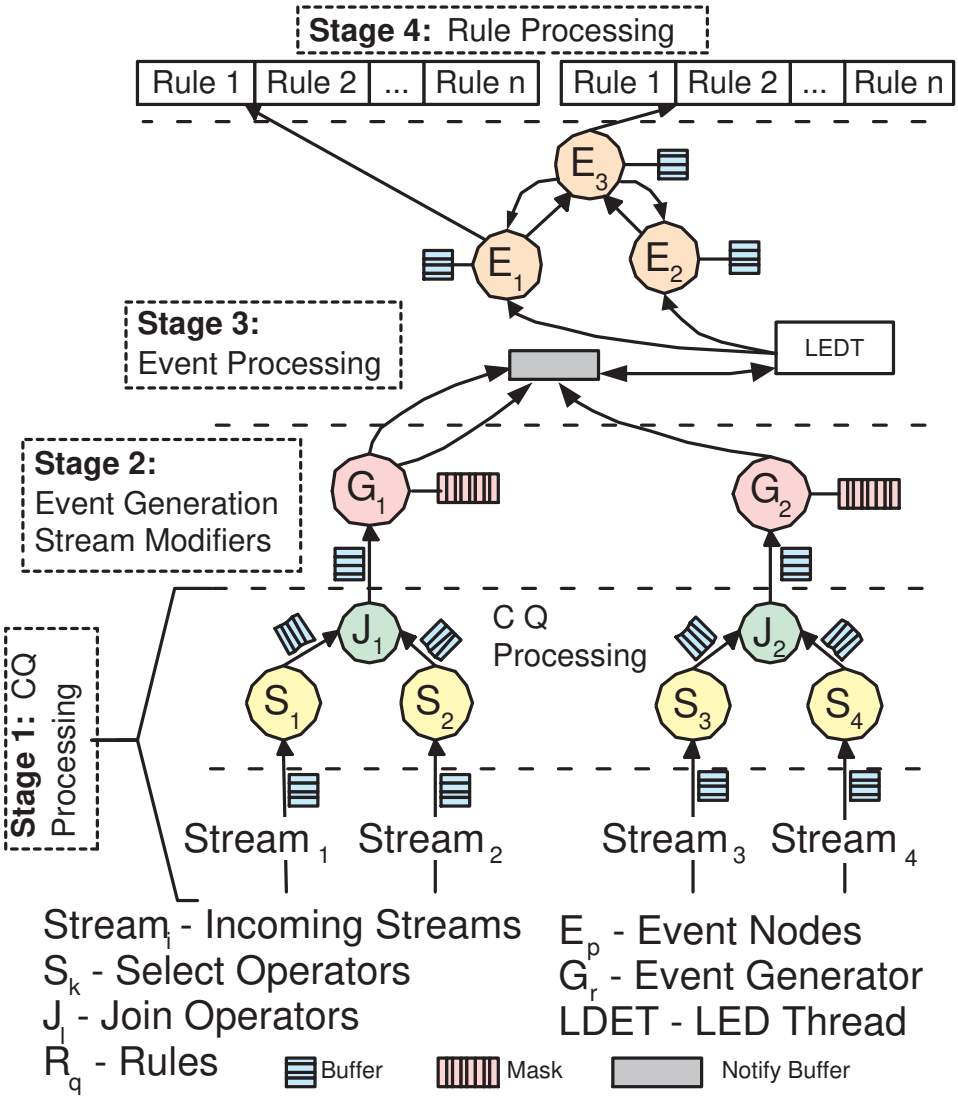

Figure 1: Integrated Event Stream Processing Architecture

Stage 1: In this stage CQs over data streams are processed. This stage processes normal CQs where it takes streams as inputs and gives computed continuous streams. In Figure 1, operators S_1,

S_2, and J_1 form a CQ. CQs can *also* generate meaningful events that can be composed further for detecting higher level events. A CQ may also give rise to multiple events. In Figure 1, primitive events E_1 and E_2 generated from J_1 and J_2, respectively, are composed to form E_3.

Stage 3: In this stage, events are defined and processed using the event detection graphs, and rules are triggered. Event detection graphs (EDGs) [38, 30] can be used to detect events, where leaf nodes represent primitive events and internal nodes represent composite events.

Stage 4: Multiple rules can be associated with an event. When an event is detected in stage 3 and a rule is triggered, conditions are checked, and associated actions are carried out by this stage.

Stage 2: This is the glue that integrates the event and stream processing. In this stage, stream output is coupled with event nodes using an event generator operator. The event detector shown in stage 3 has a common event processor buffer into which all events that are raised are queued. A single queue is necessary as events are detected and raised by different components of the system (CQs in this case) and they need to be processed according to their time of occurrence. This stage adds a new event generator operator to every stream query as the root if an event is associated with that CQ. This operator can take any number of MASKS (or conditions) and for each MASK, a different event tuple/object is created and sent to the notify buffer in stage 3. This operator is activated only when an ECA rule associated with that CQ is enabled. This operator is similar to the select operator except that when it generates an event, it enqueues that event in the buffer. In general, CQs output results in the form of tuples to the *event generator* operator nodes that are attached to the root node of the CQ tree. As shown in Figure 1, nodes J_1 and J_2 are attached to event generator nodes G_1 and G_2, which have MASKS (or conditions on event parameters). Thus, CQ results from J_1 and J_2 are converted to events by G_1 and G_2.

We plan on extending the above prototype to perform video stream analysis based on the operators and semantics presented in this report. The extensions will be incorporated into the event library so that they can be used with other applications. 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.

9 Conclusions

In this report, we have made a case for the need for event composition to detect some higher-level or interesting events that correspond to certain situations. Based on our analysis, we have introduced several event operators, their semantics, detection algorithms, and their integration to an existing event detector. We also made a case for a more expressive window mechanisms for processing video data streams. We have also introduced several extensions to the current window specifications and indicated their usage. Finally, we presented the integrated architecture into which the above can be easily incorporated.

As future work, we need to incorporate the above operators (both event and window) into the architecture for experimentation. In addition to window extensions, we also need additional spatial and temporal operators to make the video querying more expressive. We will be pursuing these as part of suture work.

Bibliography

[1] R. Chellappa, "Frontiers in image and video analysis nsf/fbi/darpa workshop report," in *Workshop*, 2014, p. 120. [Online]. Available: www.umiacs.umd.edu/~rama/NSF_report.pdf

[2] R. Mazzon, S. F. Tahir, and A. Cavallaro, "Person re-identification in crowd," *Pattern Recogn. Lett.*, vol. 33, no. 14, p. 1828–1837, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.patrec.2012.02.014

[3] A. Bedagkar-Gala and S. K. Shah, "A survey of approaches and trends in person re-identification," *Image and Vision Computing*, 2014.

[4] A. Bedagkar-Gala and S. K. Shah, "A survey of approaches and trends in person re-identification," *Image and Vision Computing*, vol. 32, no. 4, pp. 270 – 286, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0262885614000262

[5] D. D. Saur, Y. Tan, S. R. Kulkarni, and P. J. Ramadge, "Automated analysis and annotation of basketball video," in *Storage and Retrieval for Image and Video Databases V, San Jose, CA, USA, February 8-14, 1997*, ser. SPIE Proceedings, I. K. Sethi and R. C. Jain, Eds., vol. 3022. SPIE, 1997, pp. 176–187. [Online]. Available: https://doi.org/10.1117/12.263406

[6] D. Yow, B. lock Yeo, M. Yeung, and B. Liu, "Analysis and presentation of soccer highlights from digital video," 1995, pp. 499–503.

[7] Y. Rui, A. Gupta, and A. Acero, "Automatically extracting highlights for tv baseball programs," in *Proceedings of the Eighth ACM International Conference on Multimedia*, ser. MULTIMEDIA '00. New York, NY, USA: ACM, 2000, pp. 105–115. [Online]. Available: http://doi.acm.org/10.1145/354384.354443

[8] N. Shrestha, A. Nurani Venkitasubramanian, and M.-F. Moens, "Key event detection in video using asr and visual data," in *Proceedings of the COLING Workshop on Vision and Language (VL'14)*, 2014, pp. 46–53.

[9] A. J. Aved and K. A. Hua, "An informatics-based approach to object tracking for distributed live video computing," *Multimedia Tools and Applications*, vol. 68, no. 1, pp. 111–133, 2014.

[10] A. J. Aved, "Scene understanding for real time processing of queries over big data streaming video," Ph.D. dissertation, University of Central Florida Orlando, Florida, 2013.

[11] H. Engstrom, M. Berndtsson, and B. Lings, "ACOOD Essentials," University of Skovde, Tech. Rep., 1997, http://www.ida.his.se/ida/research/tech_reports/reports/tr97/HS-IDA-TR-97-010.ps.

[12] O. Diaz, N. Paton, and P. Gray, "Rule Management in Object-Oriented Databases: A Unified Approach," in *Proceedings of International Conference on Very Large Data Bases*, September 1991, pp. 317–326.

[13] U. Schreier, H. Pirahesh, R. Agrawal, and C. Mohan, "Alert: An architecture for transforming a passive dbms into an active dbms," in *Proceedings of the 17th International Conference on Very Large Data Bases*, ser. VLDB '91. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1991, pp. 469–478. [Online]. Available: http://portal.acm.org/citation.cfm?id=645917.672314

[14] E. Simon and J. Kiernan, "The A-RDL System," in *Active Database Systems: Triggers and Rules For Advanced Database Processing*. Morgan Kaufmann, 1996, pp. 111–149.

[15] E. N. Hanson, "The Design and Implementation of the Ariel Active Database Rule System," *Transactions of Knowledge and Data Engineering*, vol. 8, no. 1, pp. 157–172, 1996.

[16] E. N. Hanson, "Active Rules in Database Systems," in *Ariel*, N. W.Paton, Ed. Springer, 1999, pp. 221–232.

[17] S. Ceri and R. Manthey, "Chimera: a model and language for active DOOD Systems," in *Proceedings of East-West Database Workshop, Workshops in Computing*, 1994, pp. 3–16.

[18] R. Meo, G. Psaila, and S. Ceri, "Composite Events in Chimera," in *Proceedings of Internation Conference on Extending Database Technology*, 1996, pp. 56–76.

[19] S. Ceri, P. Fraternali, S. Paraboschi, and L. Branca, "Active rule management in chimera," in *Active Database Systems - Triggers and Rules For Advanced Database Processing*. Morgan Kaufman Publishers Inc., 1996, ch. 6, pp. 151–176.

[20] N. H. Gehani, H. V. Jagadish, and O. Shmueli, "Composite Event Specification in Active Databases: Model & Implementation," in *Proceedings of International Conference on Very Large Data Bases*, 1992, pp. 327–338.

[21] O. Díaz and A. Jaime, "EXACT: An Extensible Approach to Active Object-Oriented Databases," *VLDB Journal*, vol. 6, no. 4, pp. 282–295, 1997.

[22] U. Dayal, B. T. Blaustein, A. P. Buchmann, U. S. Chakravarthy, M. Hsu, R. Ledin, D. R. McCarthy, A. Rosenthal, S. K. Sarin, M. J. Carey, M. Livny, and R. Jauhari, "The HiPAC Project: Combining Active Databases and Timing Constraints," *SIGMOD Record*, vol. 17, no. 1, pp. 51–70, 1988.

[23] 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." Xerox Advanced Information Technology, Cambridge, Tech. Rep., 1989.

[24] U. Dayal, A. P. Buchmann, and S. Chakravarthy, "The hipac project," in *Active Database Systems: Triggers and Rules For Advanced Database Processing*. Morgan Kaufmann, 1996, pp. 177–206.

[25] C. Collet, T. Coupaye, and T. Svensen, "NAOS efficient and modular reactive capabilities in an object-oriented database system," in *Proceedings of the International Conference on Very Large Data Bases*, September 1994, pp. 132–143.

[26] N. H. Gehani and H. V. Jagadish, "ODE as an Active Database: Constraints and Triggers," in *Proceedings of International Conference on Very Large Data Bases*, September 1991, pp. 327–336.

[27] D. L. Lieuwen, N. H. Gehani, and R. Arlein, "The Ode Active Database: Trigger Semantics and Implementation," in *Proceedings of International Conference on Data Engineering*, March 1996, pp. 412–420.

[28] M. Stonebraker, A. Jhingran, J. Goh, and S. Potamianos, "On rules, procedures, caching and views in data base systems," in *SIGMOD Conference*, 1990, pp. 281–290.

[29] A. Buchman, H. Branding, T. Kudrass, and J. Zimmermann, "Rules in an open system: The REACH rule system," in *Rules in Database Systems.*, September 1993, pp. 111–126.

[30] A. P. Buchmann, J. Zimmermann, J. A. Blakeley, and D. L. Wells, "Building an integrated active oodbms: Requirements, architecture, and design decisions," in *In Proceedings of the 11th International Conference on Data Engineering*. IEEE Computer Society Press, 1995, pp. 117–128.

[31] A. Dinn, N. W. Paton, M. H. Williams, and A. Fernandes, "An Active Rule Language for ROCK & ROLL," in *In Proceedings of British National Conference on Databases (BNCOD)*, 1996, pp. 36–55.

[32] A. Dinn, M. H. Williams, and N. W. Paton, "ROCK & ROLL: A Deductive Object-Oriented Database with Active and Spatial Extensions," in *Proceedings of International Conference of Data Engineering*, 1997, pp. 491–502.

[33] S. Gatziu and K. R. Dittrich, "SAMOS: An Active, Object-Oriented Database System," *IEEE Quarterly Bulletin on Data Engineering*, vol. 15, no. 1-4, pp. 23–26, December 1992.

[34] S. Gatziu and K. R. Dittrich, "Events in an Object-Oriented Database System," in *Proceedings of Rules in Database Systems*, September 1993, pp. 23–39.

[35] E. Anwar, L. Maugis, and S. Chakravarthy, "A New Perspective on Rule Support for Object-Oriented Databases," in *SIGMOD Conference*, 1993, pp. 99–108.

[36] S. Chakravarthy, E. Anwar, L. Maugis, and D.Mishra, "Design of Sentinel: An Object-Oriented DBMS with Event-Based Rules," *Information and Software Technology*, vol. 36, no. 9, pp. 559–568, 1994.

[37] P. Seshadri, M. Livny, and R. Ramakrishnan, "The Design and Implementation of a Sequence Database System," in *Proceedings of International Conference on Very Large Data Bases*, 1996, pp. 99–110.

[38] S. Chakravarthy and D. Mishra, "Snoop: An expressive event specification language for active databases," *Transactions of Data Knowledge and Engineering*, vol. 14, no. 1, pp. 1–26, 1994.

[39] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S. Kim, "Composite Events for Active Databases: Semantics, Contexts and Detection," in *VLDB*, 1994, pp. 606–617.

[40] J. Widom, "The Starburst Rule System: Language Design, Implementation, and Applications," *IEEE Quarterly Bulletin on Data Engineering*, vol. 15, no. 1-4, pp. 15–18, December 1992.

[41] J. Widom, R. J. Cochrane, and . L. B, G, "Implemented Set-Oriented Production Rules as an Extension of Starburst," in *Proceedings 17th International Conference on Very Large Data Bases*, Barcelona (Catalonia, Spain), Sep. 1991, pp. 275–286.

[42] G. Kappel and W. Retschitzegger, "The TriGS Active Object-Oriented Database System - An Overview," *ACM SIGMOD Record*, vol. 27, pp. 36–41, 1998.

[43] W. Retschitzegger, "Composite Event Management in TriGS - Concepts and Implementation," in *Proceedings of International Conference on Database and Expert Systems Applications*, September 1998, pp. 1–15.

[44] A. Kotz-Dittrich, "Adding Active Functionality to an Object-Oriented Database System - a Layered Approach," in *Proceedings of the Conference on Database Systems in Office, Technique and Science*, March 1993, pp. 54–73.

[45] I. Motakis and C. Zaniolo, "Formal Semantics for Composite Temporal Events in Active Database Rules," *Journal of System Integration*, vol. 7, no. 3-4, pp. 291–325, 1997.

[46] I. Motakis and C. Zaniolo, "Temporal Aggregation in Active Database Rules," in *Proceedings of ACM-SIGMOD International Conference on Management of Data*, 1997, pp. 440–451.

[47] J. Widom and S. Ceri, *Active Database Systems: Triggers and Rules*. Morgan Kaufmann Publishers, Inc., 1996.

[48] N. W. Paton, *Active Rules in Database Systems*. Springer, 1999.

[49] U. Jaeger and J. C. Freytag, "An Annotated Bibliography on Active Databases," *SIGMOD Record*, vol. 24, no. 1, pp. 58–69, 1995.

[50] A. J. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, and W. M. White, "Cayuga: A general purpose event monitoring system," in *CIDR*, 2007, pp. 412–422.

[51] L. Brenna, A. J. Demers, J. Gehrke, M. Hong, J. Ossher, B. Panda, M. Riedewald, M. Thatte, and W. M. White, "Cayuga: a high-performance event processing engine," in *SIGMOD Conference*, 2007, pp. 1100–1102.

[52] E. Wu, Y. Diao, and S. Rizvi, "High-performance complex event processing over streams," in *SIGMOD Conference*, 2006, pp. 407–418.

[53] H. Zhang, Y. Diao, and N. Immerman, "Recognizing patterns in streams with imprecise timestamps," *Proc. VLDB Endow.*, vol. 3, pp. 244–255, September 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1920841.1920875

[54] R. Adaikkalavan and S. Chakravarthy, "SnoopIB: Interval-based event specification and detection for active databases," *Transactions of Data Knowledge and Engineering*, vol. 59, no. 1, pp. 139–165, 2006.

[55] R. Adaikkalavan, "Generalization and Enforcement of Role-Based Access Control using a Novel Event-based Approach." Ph.D. dissertation, The University of Texas at Arlington, August 2006.

[56] R. Adaikkalavan and S. Chakravarthy, "Event Specification and Processing for Advanced Applications: Generalization and Formalization," in *DEXA*, 2007, pp. 369–379.

[57] R. Adaikkalavan and S. Chakravarthy, "Events must be complete in event processing!" in *Proceedings, Annual ACM SIG Symposium On Applied Computing*, 2008, pp. 1038–1039.

[58] S. Chakravarthy and R. Adaikkalavan, "Event and Streams: Harnessing and Unleashing Their Synergy," in *International Conference on Distributed Event-based Systems*, July 2008, pp. 1–12.

[59] H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, E. Galvez, J. Salz, M. Stonebraker, N. Tatbul, R. Tibbetts, and S. Zdonik, "Retrospective on aurora," *VLDB Journal: Special Issue on Data Stream Processing*, vol. 13, no. 4, pp. 370–383, 2004.

[60] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: a new model and architecture for data stream management," *The VLDB Journal*, vol. 12, pp. 120–139, August 2003. [Online]. Available: http://dx.doi.org/10.1007/s00778-003-0095-z

[61] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. B. Zdonik, "Monitoring Streams - A New Class of Data Management Applications," in *Proceedings of the International Conference on Very Large Data Bases*, August 2002, pp. 215–226.

[62] S. B. Zdonik, M. Stonebraker, M. Cherniack, U. Çetintemel, M. Balazinska, and H. Balakrishnan, "The aurora and medusa projects," *IEEE Data Eng. Bull.*, vol. 26, no. 1, pp. 3–10, 2003.

[63] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, Y. Xing, and S. B. Zdonik, "Scalable distributed stream processing," in *CIDR*, 2003.

[64] Borealis, "Second Generation Stream Processing Engine," 2003, http://nms.lcs.mit.edu/projects/borealis/.

[65] Y. Xing, S. B. Zdonik, and J.-H. Hwang, "Dynamic Load Distribution in the Borealis Stream Processor," in *Proceedings of International Conference on Data Engineering*, 2005, pp. 791–802.

[66] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. B. Zdonik, "The Design of the Borealis Stream Processing Engine," in *Conference on Innovations in Data Research*, 2005, pp. 277–289.

[67] Y. Ahmad, B. Berg, U. Çetintemel, M. Humphrey, J.-H. Hwang, A. Jhingran, A. Maskey, O. Papaemmanouil, A. Rasin, N. Tatbul, W. Xing, Y. Xing, and S. B. Zdonik, "Distributed operation in the borealis stream processing engine," in *SIGMOD Conference*, 2005, pp. 882–884.

[68] J.-H. Hwang, M. Balazinska, A. Rasin, U. Çetintemel, M. Stonebraker, and S. B. Zdonik, "High-Availability Algorithms for Distributed Stream Processing," in *Proceedings of International Conference on Data Engineering*, April 2005, pp. 779–790.

[69] J.-H. Hwang, S. Cha, U. Çetintemel, and S. B. Zdonik, "Borealis-r: a replication-transparent stream processing system for wide-area monitoring applications," in *SIGMOD Conference*, 2008, pp. 1303–1306.

[70] N. Tatbul and S. B. Zdonik, "Dealing with overload in distributed stream processing systems," in *ICDE Workshops*, 2006, pp. 24–24.

[71] Y. Xing, J.-H. Hwang, U. Çetintemel, and S. B. Zdonik, "Providing resiliency to load variations in distributed stream processing," in *VLDB*, 2006, pp. 775–786.

[72] N. Tatbul, U. Çetintemel, and S. B. Zdonik, "Staying fit: Efficient load shedding techniques for distributed stream processing," in *VLDB*, 2007, pp. 159–170.

[73] StreamBase, "StreamBase Home Page," 2004, http://www.streambase.com.

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

[75] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems." in *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, June 2002, pp. 1–16.

[76] S. Babu and J. Widom, "Continuous queries over data streams," in *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, September 2001, pp. 109–120.

[77] A. Arasu, B. Babcock, S. Babu, J. McAlister, and J. Widom, "Characterizing memory requirements for queries over continuous data streams," *ACM Transactions of Database Systems*, vol. 29, pp. 162–194, 2004.

[78] S. Babu, U. Srivastava, and J. Widom, "Exploiting k-constraints to reduce memory overhead in continuous queries over data streams." *ACM Transactions of Database Systems*, vol. 29, no. 3, pp. 545–580, 2004.

[79] A. Arasu and J. Widom, "A Denotational Semantics for Continuous Queries over Streams and Relations," *SIGMOD Record*, vol. 33, no. 3, pp. 6–12, 2004.

[80] A. Arasu, S. Babu, and J. Widom, "The CQL continuous query language: semantic foundations and query execution," *VLDB Journal*, vol. 15, no. 2, pp. 121–142, 2006.

[81] A. Arasu, S. Babu, and J. Widom, "Cql: A language for continuous queries over streams and relations." in *Database Programming Languages, 9th International Workshop*, September 2003, pp. 1–19.

[82] C. Olston, J. Jiang, and J. Widom, "Adaptive filters for continuous queries over distributed data streams." in *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, July 2003, pp. 563–574.

[83] A. Arasu and J. Widom, "Resource sharing in continuous sliding-window aggregates." in *Proceedings of the Thirtieth International Conference on Very Large Data Bases*, August 2004, pp. 336–347.

[84] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. S. Manku, C. Olston, J. Rosenstein, and R. Varma, "Query processing, approximation, and resource management in a data stream management system." in *Conference on Innovations in Database Research*, 2003.

[85] S. Babu, K. Munagala, J. Widom, and R. Motwani, "Adaptive caching for continuous queries." in *Proceedings of the 21st International Conference on Data Engineering*, 2005, pp. 118–129.

[86] Q. Jiang and S. Chakravarthy, "Anatomy of a Data Stream Management System," in *ADBIS Research Communications*, 2006.

[87] Q. Jiang and S. Chakravarthy, "Data stream management system for MavHome," in *Proceedings, Annual ACM SIG Symposium On Applied Computing*, 2004, pp. 654–655.

[88] Q. Jiang, R. Adaikkalavan, and S. Chakravarthy, "NFM^i: An Inter-domain Network Fault Management System," in *ICDE*, 2005, pp. 1036–1047.

[89] J. Qingchun, "A Framework for Supporting Quality of Service Requirements in a Data Stream Management System." Ph.D. dissertation, The University of Texas at Arlington, August 2005.

[90] Q. Jiang, R. Adaikkalavan, and S. Chakravarthy, "MavEStream: Synergistic Integration of Stream and Event Processing," in *International Conference on Digital Communications*, 2007, pp. 29–29.

[91] D. J. Cook, M. Youngblood, E. Heierman, K. Gopalratnam, S. Rao, A. Litvin, and F. Khawaja, "Mavhome: An agent-based smart home," in *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, March 2003, pp. 521–524.

[92] MavHome, "MavHome Project Home Page, University of Texas at Arlington," 2001, http://cygnus.uta.edu/mavhome/.

[93] Q. Jiang and S. Chakravarthy, "Queueing analysis of relational operators for continuous data streams," in *CIKM*, 2003, pp. 271–278.

[94] Q. Jiang and S. Chakravarthy, "Scheduling Strategies for Processing Continuous Queries over Streams," in *Proceedings of the Annual British National Conference on Databases*, 2004, pp. 16–30.

[95] B. Babcock, S. Babu, R. Motwani, and M. Datar, "Chain: operator scheduling for memory minimization in data stream systems," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, June 2003, pp. 253–264.

[96] B. Babcock, S. Babu, M. Datar, R. Motwani, and D. Thomas, "Operator scheduling in data stream systems," *The VLDB Journal*, vol. 13, no. 4, pp. 333–353, 2004.

[97] Q. Jiang and S. Chakravarthy, "A framework for supporting load shedding in data stream management systems," *TR CSE-2004-19, UT Arlington*, May 2004, http://www.cse.uta.edu/research/publications/Downloads/CSE-2004-19.pdf.

[98] B. Kendai and S. Chakravarthy, "Load Shedding in MavStream: Analysis, Implementation, and Evaluation," in *Proceedings, International British National Conference on Databases (BNCOD)*, 2008, pp. 100–112.

[99] A. Gilani, S. Sonune, B. Kendai, and S. Chakravarthy, "The Anatomy of a Stream Processing System," in *BNCOD*, 2006, pp. 232–239.

[100] S. Chakravarthy and V. Pajjuri, "Scheduling Strategies and Their Evaluation in a Data Stream Management System," in *BNCOD*, 2006, pp. 220–231.

[101] A. Gilani, "Design and Implementation of Stream Operators, Query Instantiator and Stream Buffer Manager." Master's thesis, The University of Texas at Arlington, December 2003.

[102] S. Sonune, "Design and Implementation of Windowed Operators and Scheduler for Stream Data." Master's thesis, The University of Texas at Arlington, December 2003.

[103] V. K. Pajjuri, "Design and Implementation of Scheduling Strategies and their Evaluation in MavStream." Master's thesis, The University of Texas at Arlington, December 2004.

[104] B. Kendai, "Runtime Optimization and Load Shedding in MavStream: Design and Implementation." Master's thesis, The University of Texas at Arlington, December 2006.

[105] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, F. Reiss, and M. A. Shah, "TelegraphCQ: Continuous Dataflow Processing," in *Proceedings of ACM-SIGMOD International Conference on Management of Data*, 2003, p. 668.

[106] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah, "Telegraphcq: Continuous dataflow processing for an uncertain world." in *Conference on Innovations in Database Research (CIDR)*, 2003.

[107] M. A. Shah, J. M. Hellerstein, S. Chandrasekaran, and M. J. Franklin, "Flux: An Adaptive Partitioning Operator for Continuous Query Systems," in *Proceedings of International Conference on Data Engineering*, April 2003, pp. 25–36.

[108] S. Krishnamurthy, S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Madden, F. Reiss, and M. A. Shah, "TelegraphCQ: An Architectural Status Report," *IEEE Data Engineering Bulletin*, vol. 26, no. 1, pp. 11–18, 2003.

[109] C. D. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk, "The Gigascope Stream Database," *IEEE Data Engineering Bulletin*, vol. 26, no. 1, pp. 27–32, 2003.

[110] C. D. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk, "Gigascope: A Stream Database for Network Applications," in *Proceedings of ACM-SIGMOD International Conference on Management of Data*, June 2003, pp. 647–651.

[111] Y. Zhu and D. Shasha, "StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time," in *Proceedings of International Conference on Very Large Data Bases*, August 2002, pp. 358–369.

[112] M. Sullivan and A. Heybey, "Tribeca: A system for managing large databases of network traffic," in *usenix*, June 1998, pp. 13–24.

[113] M. Sullivan, "Tribeca: A stream database manager for network traffic analysis," *Proceedings of the International Conference on Very Large Data Bases*, pp. 594–605, September 1996.

[114] D. B. Terry, D. Goldberg, D. Nichols, and B. M. Oki, "Continuous queries over append-only databases," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM Press, June 1992, pp. 321–330.

[115] D. Goldberg, D. Nichols, B. M. Oki, and D. B. Terry, "Using collaborative filtering to weave an information tapestry." *ACM Communications*, vol. 35, no. 12, pp. 61–70, 1992.

[116] A. Lerner and D. 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*. VLDB Endowment, 2003, pp. 345–356.