Opportunistic Spatio-temporal Event Processing for Mobile Situation Awareness

Kirak Hong, David Lillethun,
Umakishore Ramachandran
College of Computing
Georgia Institute of Technology
Atlanta, Georgia, USA
{khong9, davel, rama }@cc.gatech.edu

Beate Ottenwälder, Boris Koldehofe
Institute of Parallel and Distributed Systems
University of Stuttgart
Stuttgart, Germany
{beate.ottenwaelder,
boris.koldehofe}@ipvs.uni-stuttgart.de

ABSTRACT

With the proliferation of mobile devices and sensors, mobile situation awareness is becoming an important class of applications. The key requirement of this class of applications is low-latency processing of events stemming from sensor data in order to provide timely situational information to mobile users. To satisfy the latency requirement, we propose an opportunistic spatio-temporal event processing system that uses prediction-based continuous query handling. Our system predicts future query regions for moving consumers and starts processing events early so that the live situational information is available when the consumer reaches the future location. In contrast to existing systems, our system provides timely information about a consumer's current position by hiding computation latency for processing recent events. To evaluate our system, we measure the quality of results and timeliness of live situational information with various query parameters. Our evaluation shows that we can achieve highly meaningful query results with near-zero latency in most cases.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Distributed networks; C.2.4 [**Distributed Systems**]: Distributed applications

Keywords

mobility; complex event processing; situation awareness

1. INTRODUCTION

The explosive growth of sensors in the environment is enabling many future applications. Vehicles and mobile communication devices (i.e., smartphones and tablets) have a variety of sensors that they lacked only a few years ago [5]. Large scale camera deployments around transportation infrastructure, such as airports, seaports, and highways are becoming common as well. For example, there are hundreds of cameras monitoring the highways in the Atlanta metro area [1]. These sensors could enable applications such as one that notifies drivers of live road conditions near them, warning of traffic, accidents, or obstructions on the road. Such information could also be used in live, adaptive traffic routing applications. Furthermore, urban sensor deployments can enhance community safety, for example, by notifying police officers of suspicious individuals based on their behaviors.

These applications, often classified as mobile situation awareness, provide situational information to mobile users based on the events from various sensors that are widely deployed in the environment. Mobile situation awareness is naturally event-based, processing primitive events from sensors using application-specific algorithms to generate high-level events including situational information, and finally either reporting the situational information to a human or using it for automated decision making. In this context, events are spatio-temporal in nature – they occur at a particular place and time – and mobile users typically make continuous queries about their surroundings, i.e., situational information based on recent, nearby events.

As a specific example, consider a user who is driving from New York to Los Angeles. The user may have a vehicular application to automatically detect driving conditions along the route (e.g., traffic, accidents, road obstructions, or construction) and reroute the user around those problems. However, it is not practical to process the events along this whole cross-country route for the entire trip. Furthermore, an accident along the route near LA may not be relevant if it happens while the user is only just leaving NY. Therefore, the application should only process events in the vicinity of the user, according to some reasonable, application-defined range. Timely delivery of events is critical for this application, however, since there is no point in notifying the user's vehicle of an accident or bad traffic after the user passes the last exit on the highway before the problem – it is too late for the user to do anything useful with that information. This is an example of an application where an approximate result is better than a late result.

Complex Event Processing (CEP) is a well-known paradigm to generate such situational information. To generate situational information, CEP applications use operator graphs consisting of multiple operators that perform online processing of events. Online processing of events allows asynchronous, low-latency generation of situational information since events are processed as they come. In a traditional, infrastructure-based CEP application, a single operator graph would take input from all the sensors in the infrastructure and perform continuous computation to generate live events.

However, for a mobile situation awareness application, it is not scalable to continuously perform live computation on all of the sensors everywhere. Furthermore, mobile users are typically only interested in events occurring within a certain area around them. Computing events outside that area results in wasted computation. The reduction in needed computation when only processing events in a range around the mobile user, vs. processing all events, is substantial [19]. A naive solution would be to start the operator graph anew in different locations as the mobile user moves to those locations. However, processing of some historical events is typically necessary before live event processing can begin. Often the user is interested in recent events, not only ones happening right now. It also may be the case that some historical context is needed to correctly process new events. Events must be processed in temporal order, so there is a processing delay to deal with the historical events before live event processing begins. Therefore, if the operator graph is started in a location only when a user moves to that location, the user will then be in a different location by the time the operator graph has finished processing the historical events. This could potentially lead to a situation where the operator graph is constantly trying to "catch up" to the user, resulting in processing events in inappropriate locations and never actually getting to process any "live" events.

We propose to address this problem by processing the historical events for a location before the mobile user arrives at that location, so that live event processing begins at the moment the user arrives, if not before. Two existing technologies enable this just-in-time computation: future location predictions for the mobile user placing the query, and processing time estimations for the CEP algorithms. Several location prediction algorithms already exist [32], and profiling techniques can be used to estimate processing time. Our system treats both of these as black boxes, allowing different location prediction and processing time estimation algorithms to be plugged in. However, two important challenges still remain. First, if the speed of the mobile user is too fast compared to the processing time, the historical event processing may take longer than the user takes to get to the new location. To mitigate this, we propose using parallel resources to enable pipeline processing of future locations in several time- steps look-ahead. Second, the location predictor may not give a single, accurate location prediction. To mitigate the imperfection of location prediction algorithms, we propose taking several predictions for each time-step look-ahead and opportunistically compute the events for all of those locations, as our parallel resources allow. When the user arrives at that time, the prediction among those that is closest to truth (the user's actual position) will be selected and its events returned.

Our research contributions include 1) a system architecture that enables spatio-temporal event processing for mobile situation awareness applications; 2) methods for (a) starting event processing at predicted future locations in advance of a mobile user's arrival, (b) pipelining multiple future prediction points to allow completion of event processing by the time the mobile user arrives, and (c) selecting multiple location points from a predictor to opportunistically compute events at future locations; 3) metrics for assessing the quality of results and timeliness of mobile situation awareness applications; and 4) an experimental evaluation of our system and methods.

The remainder of this paper is structured as follows: Section 2 discusses related work. Section 3 presents the system architecture. Section 4 discusses our opportunistic event processing method. Section 5 evaluates our system and Section 6 discusses future work and concludes.

2. RELATED WORK

Many complex event processing systems [25, 20, 2, 7, 18] provide methods to efficiently detect interesting patterns on various sensor data for situation awareness applications. To reduce latency for processing events, some CEP systems exploit parallelism [8, 13] while others support adaptive placement of operators [28, 24]. However, these systems are designed to support CEP on a fixed infrastructure of sources. In contrast, our system supports CEP for mobile devices, such as smart vehicles, which requires migrating an operator graph to a new region when a user moves. Koldehofe et al. [19] propose a system that supports CEP on moving ranges. However, they do not address the latency of processing historical events when an operator graph moves to another region. We use this work as a baseline for our approach to prove the efficacy of opportunistic event processing in terms of timeliness and quality of results.

Research in spatio-temporal databases has developed various representations of spatio-temporal objects and methods for querying and storing spatio-temporal objects [9, 23, 10]. These spatio-temporal databases are complementary to our system since they can serve as a spatio-temporal event storage module in our system. Predictive query handling on spatio-temporal databases [16, 12] allows answering queries about the future locations of mobile objects, e.g., the ten nearest neighbors after five minutes, based on location prediction of mobile objects. In contrast, our system delivers just-in-time situational information about the recent state of the current location. Hendawi et al. [12] precompute query results to improve scalability and reduce the latency of query handling. However, their work combines on-demand query results with precomputed query results to provide complete results with various best-effort latencies. Our system provides query results with near-zero latency, although query results may not be complete due to inaccurate location predictions.

Mobile publish-subscribe systems adapt streaming based on consumer location changes [15, 22]. Cilia et al. [6] support pre-subscriptions to new locations before the consumer arrives to improve the bootstrapping process of mobile event-based applications. Low-latency event delivery is achieved through exploiting parallelism [31] and adaptation of routing paths [30]. However, the above work does not deal with the computational latency for delivering situational information since they focus on delivering individual events rather than complex events that are generated by processing the individual events.

Large-scale situation awareness [27] is receiving increased attention due to the proliferation of sensors and advances in analytics, such as computer vision. Target Container [14] is a distributed system providing a high-level, handler-based programming abstraction that helps domain experts write large-scale surveillance applications on camera networks. SLIP-

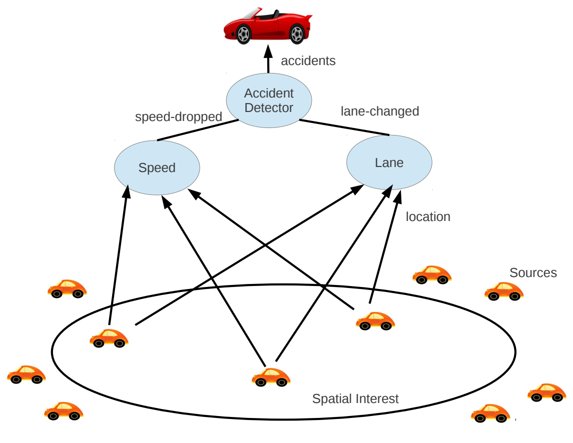

Figure 1: Example operator graph for mobile situation awareness

stream [26] offers a stream-oriented programming model that ensures automatic scalability of interactive perception applications. However, these systems focus on processing live streams from cameras, which potentially requires a huge amount of system resources in the large scale. In contrast, our system performs analysis of sensor data based on user queries and caches the query results for later use, resulting in efficient use of system resources.

Meissen et al. [21] present an approach to predict the future context of a user and efficiently deliver information based on the knowledge discrepancy between the situation-awareness system and the user. However, they focus on identifying knowledge discrepancy based on various operations, including set and relational algebra. In contrast, we focus on reducing computational latency with opportunistic event processing.

3. SYSTEM ARCHITECTURE

In this section, we discuss the details of our system architecture including the spatio-temporal event and query model, the logical modules for event processing, and the fog-based distributed architecture of our system.

3.1 Spatio-temporal Query Model

To support mobile situation awareness, our system collects various types of sensor data called events from widely deployed heterogeneous sensors such as smartphones, connected vehicles, and traffic monitoring cameras. Every event must have a set of required properties including a type, a location, and a timestamp. A location and timestamp property can be either a point or a range, regarding the type of event. The required properties are set by a producing sensor, specifying where and when a certain type of event is generated. Besides the required properties, each event may have optional application-specific properties for sensor data, which can be either structured (e.g., integer, string) or unstructured (e.g., audio or video).

Using events collected from various sensors, our system provides situational information to mobile users. Situational information is generated by executing an application-specific operator graph on the collected events. An operator graph consists of multiple operators, where each represents a piece of computation. Each operator takes one or more input types of events and produces an output type of event. Connecting edges between operators define the logical flow of events. Take Figure 1 as an example of using an operator graph for mobile situation awareness. In the example, each car continuously reports location events, including its identifier and geographical position, to the operator graph. By consuming speed and location events, two leaf operators in the operator graph detect speed-patterns and lane switches of each car. The root operator is an accident detector, which incorporates the speed and lane information to detect an accident if many cars reduced their speed and avoided a specific lane at the same time. The final outcome of this operator graph, accidents, are the situational information that our system delivers to mobile users. Note that a real application may also include heavy-weight operators, such as computer vision algorithms, and sensor streams may be unstructured, such as video.

To receive situational information, a mobile user registers a continuous query with a number of parameters listed in Table 1. An *operator graph* as the application-specific situation awareness logic that generates situational information from sensor events. The *spatial interest* and *temporal interest* defines a space and time range based on a user's current location and current wallclock time that define the selection of interesting events as an input to the operator graph. For example, a navigation system may want to display all recent accidents at nearby locations. Detecting such recent events requires processing both live and historical events, which require a mobile user to specify her temporal interest based on the duration from the current wallclock time. The user also specifies her spatial interest based on her current location to receive most relevant information.

A mobile user also sets a *location sensitivity* that indicates a distance threshold for updating the spatial scope of the situation awareness based on the user's current location. For example, a car navigation system has to show up-to-date information while a car is moving. It can set the location sensitivity at 100 meters so it can receive updated situational information for a new region whenever the user moves more than one 100 meters.

Temporal Ordering

Note that it's crucial that our system injects events to operator graphs in a temporal order. If events are not temporally ordered, the application logic of each operator has to go through past events that are already processed, and perform its algorithm again to find certain patterns using updated sequence of events. Take for example an operator that detects a linear drop in speed values over the last 30 seconds. If delivered in the right temporal order, e.g., $\{speed, t_1, l_1, 30\frac{km}{h}\}$, $\{speed, t_2, l_2, 20\frac{km}{h}\}$, $\{speed, t_3, l_3, 10\ \frac{km}{h}\}$, it's a sequential process to just compare the last two events with every new event arrival. Delivered in the wrong temporal order $\{speed, t_3, l_3,\ 10\frac{km}{h}\}$, $\{speed, t_2, l_2, 20\frac{km}{h}\}$, $\{speed,\ t_1,\ l_1,\ 30\frac{km}{h}\}$, means that the operator either detects an increase in the speed or has to sort the events and look at each event again to check if it is now a linear drop. Our ordering model assumes that live-events from the same sensor arrive in temporal order over a FIFO channel, so that a local time-stamp in the arrival order according to a local clock allows for a total ordering of events for the processing of the operator. Historical events that are injected

Query Parameter	Description	Example
spatial interest	a mobile user's interested region, in terms of distance from the user's current location	500 meters from here
temporal interest	a mobile user's interested time duration, in terms of duration from current wallclock time	recent 5 minutes
operator graph	operator graph implementing situation awareness logic	Figure 1
location sensitivity	distance threshold to update the scope of situation awareness (i.e., Our system switches to a new operator-graph if the user moves more than this threshold from the previous location.)	100 meters

Table 1: Query Parameters for Mobile Situation Awareness

to create historical situations are buffered before being collected by the operator and deterministically time-stamped at that buffer.

Operator Graph Switch

When a user moves to another region, our system needs to provide updated situational information for the region. Following the results of [19], it is beneficial to create and initialize an operator graph on demand based on user mobility, instead of deploying a vast amount of operator graphs for all possible regions. Consider again the example that a consumer is interested in accidents that happened in the last hour in a 5 km perimeter. The current operator graph already processes up-to-date events from a previous region. Integrating historical events from the new region that were not previously detected means that the current operator graph cannot process these events in the previously established temporal order. For example location l_1 was not part of the previous region, suddenly this wrong temporal ordering $\{speed, t_2, l_2, 20\frac{km}{h}\}$, $\{speed, t_3, l_3, 10\frac{km}{h}\}$, $\{speed, t_1, l_1, 30\frac{km}{h}\}$ is possible. To continuously provide situational information while a user is moving, our system creates a new operator graph for the user's updated spatial interest and terminates the previous operator graph. The new operator graph starts processing historical and live events for a new region and asynchronously delivers situational information to the user. After processing all historical events, the operator can provide live situational information by processing live events from a specific region. In order to distinguish results, they are stamped with the *spatial interest* and the interval of the maximal and minimal time-stamp from events that are used to generate the situational information.

3.2 Logical Modules for Event Processing

Figure 2 shows the logical structure of our spatio-temporal event processing architecture. Each client has a *query subscriber* through which a mobile user registers a continuous query with a server. The registered query is handled by the server's *query processor* that creates an operator graph and executes it by injecting relevant spatio-temporal events matching the continuous query. While running, the operator graph generates high-level events for situational information such as car accidents on highways, which are then asynchronously delivered to the current user through the query subscriber.

Our system includes a *spatio-temporal event storage* that stores primitive events from sensors as well as high-level events that are generated from operator graphs to serve future queries without redundant computation. All events in the spatio-temporal event storage are indexed by location, time, and type properties. An administrator given TTL value specifies a lifecycle of the events stored in the event storage. After an event is expired based on its TTL value and timestamp, our system removes the event from the spatio-temporal event storage. By removing old events, our system efficiently uses its storage space by keeping events that are more effective for situation awareness.

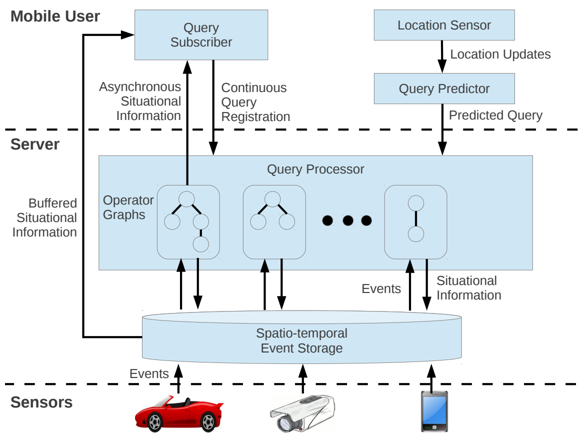

Figure 2: Conceptual view of our system architecture: Each component represents a logical module that is distributed over multiple physical nodes.

A *query predictor* is a key contribution in our system that allows opportunistic event-processing based on location predictions. The query predictor runs on the client-side and sends requests for creating and running operator graphs on a user's future locations. Detailed mechanisms of this component are discussed in Section 4.

3.3 Fog-based Distributed Event Processing

To serve a large number of continuous queries while each query potentially involves processing a large number of event streams, we need to run operator graphs on distributed computing resources. One design choice can be using the cloud, since it provides virtualized system resources on demand. Using the cloud, our system can elastically scale up and down regarding the number of users and sensor streams for event processing. However, there are two critical drawbacks for the cloud-based approach. The first problem is the network latency to deliver situational information from the cloud to mobile devices. If our system is running in the cloud, it collects events and delivers situational information through the Internet. Satyanarayanan, et al. [29], show that WAN latencies can be high and that these latencies interfere with interactive applications. We argue that sensor applications are vulnerable to the same issue due to the sense-process-actuate loop, as would be any other applications with feedback loops. In short, WAN latencies prevent our system from providing timely information to mobile users. Another problem of using the cloud is the core network traffic. Although network traffic is invisible to mobile users, using geographically centralized cloud can potentially burden the underlying network infrastructure.

In order to alleviate these problems, we propose deploying our system on computational resources located near the edge of the network. *Fog computing* is a new resource paradigm proposed by Cisco [4] that supports large–scale, latency–sensitive applications. The key idea is to maintain highly available computing and storage resources in the middle of the network infrastructure, providing resources from the core to the edge. In contrast to the cloud, these hierarchical and geographically distributed resources allow applications to perform low–latency processing near the edge. Although no commercial fog products are deployed in a production environment yet, we envision our system running in the edge network to which mobile users can communicate with low latency. This may be accomplished using smart routers, middleboxes, or simply a local cluster, and we shall henceforth refer to any computational resources at the edge of the network as “fog nodes”. On each of the distributed nodes, we run an instance of the spatio-temporal event storage and query processor to store events and execute operator graphs based on the events.

To allow fog-based distributed event processing, sensors send events to nearby fog nodes after receiving a connection endpoint (e.g., IP address) from a well-known name server. The name server maintains a directory including each fog node and their responsible spatial regions. Similarly to the sensors, a mobile user can find a nearby fog node through the name server to register a continuous query. Once an operator graph is created at the fog node, the mobile user receives situational information from the fog node that is running the operator graph. If the operator graph’s region overlaps the region of multiple fog nodes, events are asynchronously aggregated from the multiple fog nodes to the node running the operator graph. When a user moves away from the current operator graph, our system finds a new fog node to create a new operator graph for a new region.

To select the best fog node to run an operator graph, our system uses a simple heuristic based on the current and expected workload of candidate fog nodes. At runtime, our system monitors the workload at each fog node in terms of processor utilization, per-event processing time for each individual operator graph, and event rate for different event types. The latter is done by monitoring the average event rates on fine-grained regions in individual grids for different event types that span the region of a fog node.

Upon a request for creating a new operator graph, our system finds candidate fog nodes whose responsible regions overlap with the new operator graph’s region. Among the candidate nodes, our system first selects a node with highest event rate for the operator graph by summing up event rates for fine-grained regions within the overlap between the fog node’s region and the operator graph’s region. Once a fog node is selected, our system estimates how the processor utilization would change at the selected fog node if it starts running the new operator graph. The estimated workload is calculated based on the average per-event processing time of the operator graph and expected event rate for the entire region of the new operator graph. If the estimated processor utilization stays below a certain threshold, our system selects the fog node to run the new operator graph. Otherwise, our system selects the fog node with the next highest event rate for the operator graph and estimates workload again. The intuition behind this approach is to find a fog node that minimizes the network utilization for aggregating events, while ensuring that the node has enough computational capacity to run the operator graph.

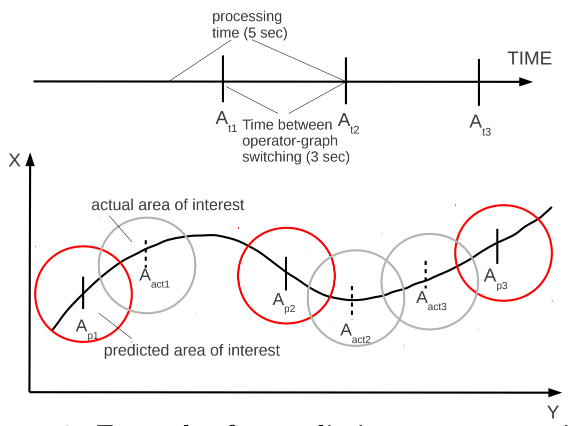

Figure 3: Examples for predictive query execution: operator graphs do not match with predicted locations and may require more time to execute than it takes to switch from one operator graph to the next

4. OPPORTUNISTIC EVENT PROCESSING

This section details the problem of processing latency for historical events when switching to a new operator graph and our solution to minimize the latency using opportunistic event processing mechanism.

4.1 Problem Description

If a mobile user moves to a new location that is farther than its distance threshold (location sensitivity), our system creates a new operator graph with an updated region and starts processing historical events matching to the mobile user’s temporal interest (e.g., recent 5 minutes). Although situational information is asynchronously generated and delivered to the user while processing the historical events, the mobile user can only receive live situational information after processing all historical events in temporal order. Because of the processing latency of historical events, switching to a new operator graph causes a delay before receiving live situational information. Since low-latency is a key requirement in mobile situation awareness, such a delay can be a significant problem. Another problem caused by the latency of processing historical events is the meaningfulness of situational information. By the time when recent situational information is delivered, a mobile user may already moved away from the previous location, which makes some situational information meaningless as they are outside of the mobile user’s current spatial interest.

4.2 Solution Overview

To give timely situational information, an operator graph must have processed all historical events when a user switches to this operator graph. In the ideal case, there may be no historical events for the operator graph's region and therefore is the processing latency for historical events zero. However, if the region contains historical events matching to the user's temporal interest, we should start the operator graph earlier before the user switches to the operator graph, giving enough time to process all historical events. To start operator graphs earlier, our system performs opportunistic computing based on the prediction of a mobile user's future locations. A location predictor based on a well-known prediction techniques, e.g., a linear dead-reckoning or locations from the navigation system, provides a generic location model for the predicted future locations. This is a set of predicted locations associated with a probability that this location represents the actual location of a consumer. Using the predicted locations, our system creates operator graphs for each future location and starts running the operator graphs before a mobile user reaches one of the future locations.

The quality and timeliness of the resulting situations detected by the predicted operator graph highly depends on where and when an operator graph is initialized. The location prediction only gives uncertain future locations, which makes it highly probable that the spatial interest does not match with the predicted operator graph' region. Consider, for example, in Figure 3 the predicted spatial region A_{p1} deviates from the actual spatial interest A_{act1} and the circles that indicate those interests only partially overlap with each other. Since processing of historical events takes time, it is also possible that the consumer moves faster to new locations than the operator graph requires to process all historical events. Consider again Figure 3, the temporal axis shows that a consumer moves within 3 seconds from A_{act1} to A_{act2} while the processing takes 5 seconds.

In this context we discuss a quality metric that allows a user to decide if the results of a predicted operator graph are meaningful, in spite of the partial overlap with the actual spatial interest. Furthermore, we discuss how the processing time of an operator graph can be anticipated. Both metrics are based on the number of events that are contained in this overlapped region (see Subsection 4.3). To maximize the probability that the consumer receives timely results, we describe the basic query prediction mechanism. To avoid the problem of late processing in face of fast movements we extend that algorithm by pipelining future operator graphs for not only the next predicted location, but for more subsequent locations (see Subsection 4.4). We also provide a method to maximize the probability that a user gets results with a desired quality from the system, based on over-provisioning of operator graphs (see Subsection 4.5).

4.3 Metrics

Quality of Results

We break the quality of results down to two metrics, the *completeness* and the *effectiveness*. The completeness indicates how many events covered by the spatial interest of a consumer are not included in the processing of results, and therefore describes the event-loss. The effectiveness indicates how many events are included in the processing but

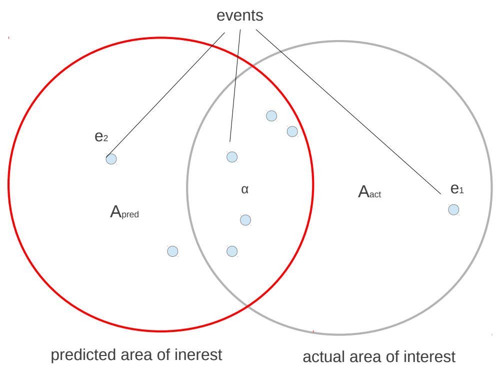

Figure 4: Quality of Results

not covered by the spatial interest of a consumer, and therefore describes the noisiness.

Since imprecise location predictions lead to overlaps in the actual spatial interest and the predicted operator graph's region, not all events that lie in the spatial interest are included in the resulting situation. For example event e_1 in Figure 4 is not included in the area of the predicted operator graph. An interesting observation, however, is that most of the relevant events in this example lie within the overlap. Our metric captures such an inhomogeneous event distribution by giving a value close to one if most of the relevant events are in the overlap, and close to zero if most of the events are not in the overlap. More formally, let V_{ov} be the number of events in the overlap and V_i be the number of events in the spatial interest of the consumer:

$$completeness = \frac{V_{ov}}{V_i} \tag{1}$$

The effectiveness considers that events can be included in the processing of resulting situations of an operator graph that are not relevant to the consumer. For example, event e_2 in Figure 4. The effectiveness is therefore represented by a value of the domain $[0, 1]$, where 1 is the ideal case where all events that are processed lie within the overlap. More formally, let V_{ov} be the number of events in the overlap and V_p be the number of events in the predicted operator graph's region:

$$effectiveness = \frac{V_{ov}}{V_p} \tag{2}$$

The involved actual spatial interest of the consumer and the operator graph's region can have a low overlap, but the resulting situations may still be meaningful to the consumer when the effectiveness and completeness are close to 1.

We can estimate the effectiveness and completeness for two circular regions A_{q_1}, A_{q_2} under the assumption of spatially evenly distributed events. Let α be the overlap of those areas, and er be the average event rate. Observe that the estimation is independent of the event rate:

$$E(completeness) = \frac{er * \alpha}{er * A_{q_1}} = \frac{\alpha}{A_{q_1}} \tag{3}$$

and

$$E(effectiveness) = \frac{er * \alpha}{er * A_{q_2}} = \frac{\alpha}{A_{q_2}} \tag{4}$$

```
1:  $P \leftarrow \emptyset$ // set of predicted locations
2:  $q_{curr} \leftarrow initial\_query$ // current operator graph
3:  $Q \leftarrow \emptyset$ // set of predicted operator graphs
4:  **upon** locationUpdate($Location\ currentLoc$ )
5:    **if** currentLoc - previousLoc > location_sensitivity **then**
6:        stopOperatorGraphs($q_{curr}$)
7:        switchToNewOperatorGraph($currentLoc$,$Q$)
8:        P ← getNextPredictedLocations($currentLoc$)
9:        Q ← generateQueries($P$)
10:       startOperatorGraphs($Q$)
11:       $previousLoc \leftarrow currentLoc$
12:   **end if**
13: **end**
14: **function** switchToNewOperatorGraph($Location\ currentLoc$, $Set\text{-}of\text{-}operator\text{-}graphs\ Q_L$)
15:   $q_{curr} \leftarrow selectNext(Q_L)$
16:   $discardOperatorGraps(Q_L - q_{curr})$
17:   $deliverHistoricEvents(q_{curr})$
18:   $initiateLiveNotification(q_{curr})$
19: **end**
```

Figure 5: Basic Query Prediction

Timeliness

The time an operator graph takes to process historical events depends on the complexity of the algorithm that is realized with the operator graph, the available resources of the executing platform and the number of input events. In order to approximate the processing time for a given platform and number of events the operator graph can be profiled on different platforms. The time in between those sampling points can then be interpolated. Let $C_i()$ be the function to determine this interpolated time, T be the temporal interest, R be the spatial region of the operator graph and $ev(R)$ be an average value of events per second in the spatial range estimated from the historical events already available in that area, then the anticipated processing time computes T_c to:

$$T_c = C_i(ev(R) * T) \tag{5}$$

Note that if another operator graph for the same spatial region is already deployed for another consumer, this result can be reused. Reusing effectively reduces the processing time on historical events, for the consumer specific operator graph that reuses results, to zero.

4.4 Query Prediction

We now outline an algorithm for the predictive query system. To explain the basic principles of the algorithm, we first describe how the system predicts future locations and initializes the next operator graph each time the mobile user moves farther than the *location sensitivity* user-defined parameter (Algorithm 5). Thereafter, we extend the algorithm to initialize operator graphs for destinations that are farther away, to deal with fast moving consumers (Algorithm 6).

Predicting the next operator graph

An initial operator graph is deployed when the consumer initiates its query at its initial location (*initial_query*). Henceforth, the system keeps track of the operator graph that currently processes events on behalf of the consumer Q_{curr} and set of future operator graphs Q that already process events for the next expected location of the consumer.

With every location update from the consumer, the query processor compares the current location ($currentLoc$) of the consumer with the location that was reported the last time the consumer switched to a new operator graph ($previousLoc$). If these locations deviate more than the $location\_sensitivity$, the system stops the processing for the current operator graph and releases its resources (Line 5-6).

In the next step (Line 7) the system selects one operator graph from the set of future operator graphs. Policies to select the next operator graph q_{curr} — specified by the domain expert that designed the operator graph — are to switch to the operator graph which has process on an area of interest that has the highest overlap with the current spatial interest of the consumer, the highest completeness, or effectiveness. Operator graphs that are not selected are stopped and their resources are freed (Line 15-16). The system delivers then all historical situations from the spatio-temporal event storage that have been detected so far and afterward delivers live-situations detected by this operator graph (Line 17-18).

At last the system selects and initializes future operator graphs (Line 8-9). The system retrieves a set of future locations from the location predictor, representing possible locations for the center of the future region of the operator graph the consumer will switch to, and initialize the operator graph at each of that locations (Line 15-17). The number of retrieved locations is a system parameter that is set by a system administrator prior to the deployment of an operator graph. In Section 4.5 we address a more sophisticated solution for the operator graph selection (Line 18).

Pipelining

When the consumer moves with a high frequency from one location to another it can happen that the processing of historical events takes longer than moving further than the threshold *location_sensitivity*. In such a case it is useful to pipeline more than one set of operator graphs. This means that operator graphs are not only processing for the next possible switch of operator graphs due to an movement beyond the threshold *location_sensitivity*, but also for a sequence of movements to new operator graphs that are further away.

Algorithm 6 presents the extended algorithm to consider this issue. The basic idea is to maintain an *eagerness* parameter that dictates how many predicted locations we look ahead. This can be set by a system administrator or dynamically adjusted according to the expected processing time T_c (see Equation 5).

The basic difference to Algorithm 5 is that iteratively more operator graphs are added (Line 11-19). With every new step of this iterative process, the location predictor is called, however, with the predicted locations of the previous prediction as input. The intuition is that those locations represent the next actual locations (Line 11). Note, that this exponentially increases the number and uncertainty of predicted locations for a high *eagerness*, which is traded-off for a timely delivery of results.

For each step until the eagerness is reached the operator checks, if an operator graph is already deployed due to a previous location update. In such a case (Line 13), the system selects and initializes a new set of operator graphs at each of the predicted location (Line 16-17). Moreover, if an operator graph is already deployed through a previous location update, the system checks if the predicted locations deviate beyond a threshold (e.g., $100m$) from the previous

```
 1: $P \leftarrow \emptyset$ // set of predicted future locations
 2: $q_{curr} \leftarrow initial\_query$ // current operator graph
 3: $Q[] \leftarrow \emptyset$ // set of predicted operator graphs
 4: $currStep \leftarrow 0$

 5: upon locationUpdate($Location\ currentLoc$ )
 6:   if currentLoc - previousLoc > location_sensitivity then
 7:       stopOperatorGraphs($q_{curr}$)
 8:       switchToNewOperatorGraph($currentLoc$,$Q[currStep]$)
 9:       P ← currentLoc
10:       for $step \in [currStep + 1, currStep + eagerness]$ do
11:           P ← getNextPredictedLocations(P)
12:           if $notExists(Q[step])$
13:               $\vee$ $locationDeviates(Q[step], P)$ then
14:               stopOperatorGraphs($Q[step]$)
15:               $Q[step] \leftarrow generateQueries(P)$
16:               startOperatoGraphs($Q[step]$)
17:           end if
18:       end for
19:       inc(currStep)
20:   end if
21: end
```

Figure 6: Opportunistic Query Prediction

predictions. In this case the operator graphs that already process for that step will be stopped and their resources will be freed (Line 14), before a new set of operator graphs is initialized.

Reuse of Results

Situational information of operator graphs can be reused, since it is buffered in the spatio-temporal event store. Which applies if multiple consumer deploy the same operator graph for the same spatial-interest. The main idea is to use a reference-counting mechanism for operator graphs that match in the spatial interest and overlap in the temporal interest. When initializing operator graphs (Line 8 of Algorithm 6) the system has then to check if other operator graph for the same spatial interest already exist. If this is the case a reference associated with the trees is increased. Moreover, instead of instantly releasing resources of operator graphs, the system decides according to the reference count if other consumers still require the operator graph.

4.5 Opportunistic Prediction

We now focus on the opportunistic algorithm to select a set of operator graphs for each predicted location. Its task is to select future operator graphs, s.t. the probability is maximized that the next time the *location_sensitivity* is exceeded an operator graph can be found that ensures a consumer-defined completeness and effectiveness. The method of choice to achieve the desired qualities is an over-provisioning of operator graphs. This means, that instead of only one operator graph, multiple-operator graphs process at the same time on historical data for slightly different areas of spatial interests. Which raises the problem that, depending on the number of over-provisioned operator graphs, the required resources and the overall time to process all operator graphs increases. Therefore it's crucial to stay below a specific maximal resource cap s_{max}, which express the maximum tolerable number of events that can be processed in parallel.

Selecting all predicted locations as centers for the areas of interest of operator graphs is one opportunity. However,

```
 1: function generateQueries( $Locations\ P_L$ )
 2:   $Q_L \leftarrow \emptyset$
 3:   $s_{tot} \leftarrow 0$
 4:   $S \leftarrow$ calcInitialCosts($P_L, S$)
 5:   while $P_L \neq \emptyset$ do
 6:       $p \leftarrow$ selectAndRemoveNext($P_L$)
 7:       if $\nexists q \in Q_L : canExecute(OperatorGraph(p))$ then
 8:           if $s_{tot} + S[p] \leq s_{max}$ then //stop if resource cap
 9:               $Q_L \leftarrow Q_L \bigcup \{OperatorGraph(p)\}$
10:               $s_{tot} = s_{tot} + S[p]$
11:           end if
12:       end if
13:   end while
14:   return $Q_L$
15: end
```

Figure 7: Query Generation

dependent on the number of retrieved queries, the overhead in terms of processing all events can be large.

To find a good approximation that selects a set of operator graphs from that set of predicted locations and stays below a resource limit s_{max}, we'll first discretize the problem and then reduce it to a set coverage problem. For the discretization we assume that only the predicted locations are valid future locations of the consumer, which is true if the number of predicted locations is infinite. The universe U for the set coverage problem is the set of events that is covered by all interest areas of the operator graph selected by the simple approach. The interest areas of those operator graphs represent the subsets $G \subset U$. The problem is then to select a minimum number of subsets $G' \subset G$, s.t., all elements of U are covered. However, we have to consider two additional constraints to the classical set coverage problem: i) all areas of not selected subsets $N \subset G$ must overlap with areas of selected subsets $G' \subset G$, s.t., completeness and effectiveness for all sets in N are expected to be ensured, and ii) the overall expected resource usage s_{tot} stays below s_{max}. The first constraint strives to ensure the quality, while the second one strives to ensure the resource limits.

Constraint Set Cover Algorithm

We adapted the greedy Johnson's Algorithm [17] to solve the set coverage problem (see Algorithm 7) to consider our constraints. It computes a set of operator graphs Q that have a high probability to ensure the results quality. It does this on the basis of a set of selected probabilistic locations P and a map S where the resource requirement of individual operator graphs, with the area of interest centered at each $p \in P$ is maintained. The algorithm starts out by receiving a set of predicted locations P_L and estimating the resource requirement of the operator graphs as if at each of those locations we center the area of interest of an operator graph and only this operator graph would process historical events (Line 2-3).

Henceforth, the algorithm tries to add sequentially for each predicted location $p \in P_L$ a new operator graph (Line 4-14). Which p is selected next depends on a policy that an domain-expert specified while developing the situation aware application with the operator graph. Choosing the next predicted location of P_L with the highest probability, will favor operator graphs that have a high probability for an overlap with the actual location, possibly only selecting few operator graphs. Choosing the operator graph with the lowest expected cost $S[p]$, will deploy many operator graphs, possibly at unlikely locations. Choosing the lowest cost factor $(1-p)*S[p]$, will trade-off the benefits of the previously described policies (Line 5).

An operator graph is said to be executable at the next selected location p if no other operator graph $q \in Q$ is already selected that satisfies the consumers requirements on completeness and effectiveness, which can be checked according to Equation 1 and Equation 2 (Line 6-7). The algorithm stops if the overall expected costs s_{tot} exceed the limit s_{max} or all point $p \in P$ are covered.

5. EVALUATION

In this section, we evaluate our system by measuring the timeliness and quality of results, based on realistic spatio-temporal events and user mobility. Timeliness is a delay between switching to a new operator graph and receiving live situational information after processing historical events. Our system predicts future locations and starts running operator graphs on the locations before user arrival, therefore provides near-zero latency for receiving live situation updates when switching to the new operator graphs. Quality of results are measured in terms of completeness and effectiveness as defined in Section 4. In our approach, the quality of results can degrade if location predictions are inaccurate. To compensate such prediction errors, we opportunistically deploy multiple operator graphs for future locations, increasing chances to find a better operator graph with higher quality of results. As provided in the subsequent subsections, our system outperforms on-demand query processing that starts operator graph after user arrival both in terms of timeliness and quality of results.

5.1 Experimental Setup

To evaluate our system, we conducted simulations using SUMO [3], a well-known traffic simulator that generates realistic mobility patterns of vehicles on a real road network. Our simulations monitored the traffic in the downtown area (3.791 km x 2.872 km) of Atlanta for 20 minutes based on the road network obtained from Open Street Map [11]. We originally simulated 1000 vehicles for each simulation but the random trip generator of SUMO automatically pruned out some invalid routes after generating trips, which resulted in 884 vehicles per simulation on average. For each simulation, we observed 282,951 events on an average, meaning each vehicle reported about 320 events on an average.

During the simulation, we recorded each vehicle's geographical location at every second, which is used in two different ways in subsequent experiments. In one case, we treated the location reports as anonymous spatio-temporal events that are used by operator graphs to generate situational information. In the other case, we use the trajectory of individual vehicles to simulate mobile users receiving situational information through continuous queries.

When measuring the quality of results and timeliness, we use four different event processing mechanisms, namely *zero*, *eager-oracle*, *eager*, and *lazy*. *Zero* represents an ideal case for both opportunistic event processing and on-demand event processing, which assumes zero computing cost for event processing. In this case, *location sensitivity*, a user-given parameter for the operator graph switch, is ignored and an operator graph is created upon every fine-grained location update. At each location, the created operator graphs provide up-to-date situational information immediately since the cost to process historical events is zero, resulting in perfect quality of results at any given time and location.

Eager-oracle is an ideal case for our opportunistic event processing that assumes an oracle predictor that knows the exact future locations of all mobile users. Since our system knows the exact future locations, it can run operator graphs on the exact locations before user arrival. However, unlike the *zero* case, operator graphs are created at coarse-grained locations defined by *location sensitivity* since the computing cost is not zero. While a mobile user is keep moving, the user's spatial interest may be slightly different from the current operator graph region at each moment, resulting in degrades in quality of results. In the following experiments, *zero* provides perfect completeness and effectiveness while *eager-oracle* provides an upper bound quality of results for our opportunistic event processing mechanism.

Eager represents opportunistic event processing with a realistic location predictor, called *dead-reckoning*. Dead-reckoning is a process of predicting future locations based on the current location, estimated speed, and estimated direction. We used simple linear dead-reckoning, using the last ten location histories to estimate future direction and speed, and predict future locations based on this estimate. To compensate for decreased completeness due to the erroneous nature of location prediction, we create four opportunistic operator graphs for each prediction.

The last case, *lazy*, represents on-demand event processing that creates an operator graph and starts processing historical events upon user arrival. Because of processing latency for historical events, a user cannot immediately receive up-to-date situational information, which results in degradation of both timeliness and quality of results.

5.2 Quality of Results Comparison

This section compares different event processing mechanisms by measuring the quality of results. We measured two metrics, completeness and effectiveness, at each fine-grained location of individual vehicles based on the overlapping events between the mobile user's actual spatial interest and the current operator graph's region. Although both metrics are measured, we only present completeness in this paper since they show the same trend. We also vary user-given parameters as well as processing latency for historical events in order to investigate each parameter's impact on the quality of result. For each experiment, the following default parameters are used, except one parameter that is selected as a control variable.

Location Sensitivity	=	100 meters
Spatial Range	=	500 meters
Temporal Range	=	60 seconds
Processing Latency	=	4 seconds

Figure 8 shows completeness for different event processing mechanisms while varying location sensitivity. In this experiment, smaller location sensitivity means more fine-grained, and more frequent operator graph switches while a user is moving. As shown in the figure, both *eager* and *eager-oracle* provide better quality of results as location sensitivity decreases because the distance between two successive operator graphs depends on the location sensitivity, and the quality of service degrades as operator graphs are spread

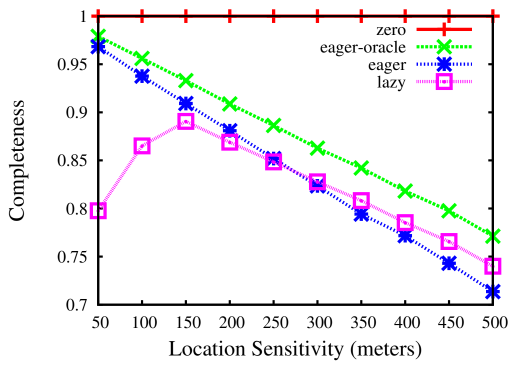

Figure 8: Completeness of query result with different location sensitivity

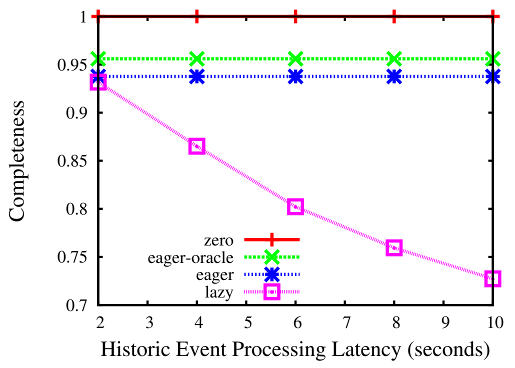

Figure 9: Completeness of query result with different processing latency

farther apart. *Eager* shows a steeper decrease in quality of results since the prediction error also increases when predicting far-away future locations. In contrast to opportunistic event processing mechanisms, on-demand event processing, or *lazy*, shows that it achieves peak completeness at 150 meters of location sensitivity. At small location sensitivities, we observed that an operator graph cannot catch up to the fast-moving vehicles since the vehicles moves away from the operator graph's region before the operator graph finishes processing historical events. On-demand event processing cannot handle such fast-moving vehicles with a small location sensitivity, thus showing the necessity of opportunistic event processing. At large location sensitivities, *lazy* shows a similar trend to *eager-oracle* because the quality of results decreases between two subsequent operator graphs while there is no uncertainty of future location involved.

Figure 9 presents completeness while varying historical event processing latency. When a user switches to a new operator graph, the operator graph should process recent historical events to provide up-to-date situational information. In realistic scenarios, processing latency for historical events depends on the number of events and complexity of operators. However, we used processing latency as a control variable in this experiment to show the impact of processing latency on the quality of results. As shown in Figure 9,

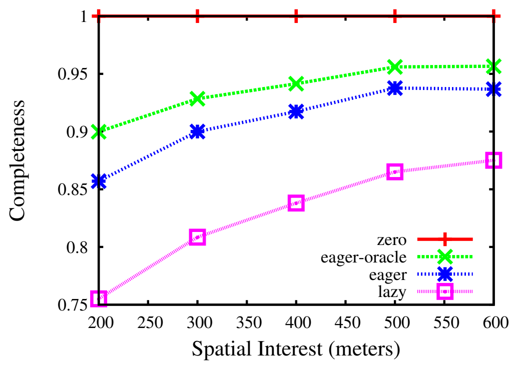

Figure 10: Completeness of query result with different spatial interest

eager and *eager-oracle* are not affected by historical event processing latency because they process events in advance through our opportunistic event processing mechanism. The difference in quality of results between the two is due to the imperfect location predictions in *eager*. However, *lazy* shows decreasing completeness when historical event processing latency increases. The decreased completeness is caused by user mobility, since a requesting mobile user can be far away from the requested location by the time a new operator graph is ready to begin "live" processing.

Figure 10 shows changes in the completeness when the range of a mobile user's spatial interest changes. In this experiment, all event processing mechanisms show the same trend that wider spatial interest yields a better quality of results. Although quality of service decreases between two subsequent operator graphs because of missing events and unnecessary events, the number of events that are missed or unnecessarily included can be negligible at large spatial interest.

5.3 Timeliness Comparison

This section compares the timeliness of situational information between opportunistic event processing and on-demand event processing. We compare timeliness by measuring the delay between a mobile user's arrival at a new location and receiving live situational information about that location. In on-demand event processing, the delay is exactly the same as the historical event processing latency since live situational information is only available after processing all the historical events. In the opportunistic event processing mechanism, however, the new operator graph might already be created and have processed all the historical events. In this case, the delay for live situational information is zero since the mobile user can immediately receive the information.

In this experiment, we used a dummy operator graph that takes a uniform random latency from one to three seconds to process historical events, while both the server and client modules are running on the same local machine. In a realistic scenario, timeliness will be also affected by network latency between a mobile user and a server that is running an operator graph. For opportunistic event processing, we use the *dead-reckoning* predictor to predict future locations

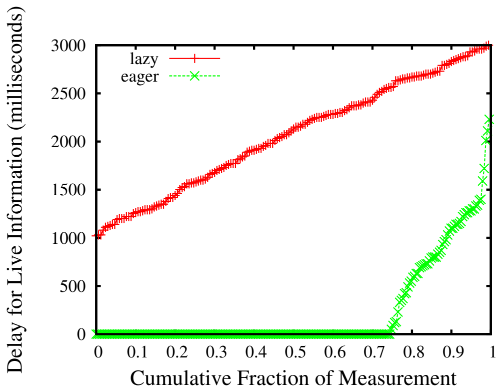

Figure 11: Timeliness of Opportunistic and On-demand Event Processing

while four operator graphs are opportunistically created at each prediction.

Figure 11 shows the cumulative distribution function (CDF) for the timeliness of both on-demand and opportunistic event processing mechanisms. On-demand event processing, labeled as *lazy*, creates an operator graph after user arrival and therefore it suffers from poor timeliness caused by the historical event processing latency. The delay for receiving live situational information is uniformly distributed between one to three seconds, following the distribution of event processing latency. However, our approach of opportunistic event processing (labeled as *eager*) provides zero latency in more than 70% of time because the operator graphs have already been created and the historical events processed when the user arrives at the future location. In few cases, our system has to create operator graphs on demand, which causes the same amount of delay with the on-demand event processing. Another cause of delay in opportunistic event processing is that a vehicle moved too fast and therefore the operator graph did not process all historical events yet.

6. CONCLUSION

The abundance of sensors in the environment enables many exciting new applications. Mobile situation awareness applications are one such class of applications that are furthermore spatio-temporal event-based as well as latency-sensitive. Event-based applications are typically programmed using a complex event processing (CEP) model, however there are challenges when the system includes mobile consumers with an interest in nearby, recent situational information that must be addressed.

It is necessary to update the spatio-temporal range of operator graphs as mobile users move through time and space in order to keep the results relevant, as well as to allow the application to scale and avoid wasted computation. However, mobile situation awareness applications often require recent historical events as well as current, live events. The processing of these recent historical events each time the location of the query changes causes a delay to delivering query results and processing live events.

In this paper, we have proposed a system and method to address the problems caused by this delay. The metrics of interest that we improved are the timeliness and quality of results for mobile spatio-temporal queries. Our contributions include: 1) the system architecture, 2) the method for eager computation of historical events, including a pipelining method to look several steps into the future and an opportunistic computing method to compensate for partially inaccurate location prediction results, and 3) a simulation evaluation that shows that our system and method achieve the near-zero latency goal, as measured by the timeliness metric, while improving the quality of results over an on-demand computation method.

There are two significant areas of future work. The first is to develop an actual mobile situation awareness application and use its operator graph to simulate our method using real algorithms and workloads. The second, as fog computing technology develops and testbeds become available, is to create a real implementation of our system using a fog computing platform.

Acknowledgment

This work is supported by contract research "CEP in the Large" of the Baden-Württemberg Stiftung. The authors would like to thank the reviewers and K. Rothermel for their helpful comments.

7. REFERENCES

[1] Georgia Navigator. `http://www.511ga.org`, 2013.

[2] Adi, A., and Etzion, O. Amit - the situation manager. *The VLDB Journal 13*, 2 (May 2004), 177–203.

[3] Behrisch, M., Bieker, L., Erdmann, J., and Krajzewicz, D. Sumo-simulation of urban mobility-an overview. In *SIMUL 2011, The Third International Conference on Advances in System Simulation* (2011), pp. 55–60.

[4] Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the 1st MCC workshop on Mobile Cloud Computingputing* (New York, NY, USA, 2012), MCC '12, ACM, pp. 13–16.

[5] Campbell, A. T., Eisenman, S. B., Lane, N. D., Miluzzo, E., Peterson, R. A., Lu, H., Zheng, X., Musolesi, M., Fodor, K., and Ahn, G.-S. The rise of people-centric sensing. *IEEE Internet Computing 12*, 4 (2008), 12–21.

[6] Cilia, M., Fiege, L., Haul, C., Zeidler, A., and Buchmann, A. P. Looking into the past: enhancing mobile publish/subscribe middleware. In *Proceedings of the 2nd international workshop on Distributed event-based systems* (New York, NY, USA, 2003), DEBS '03, ACM, pp. 1–8.

[7] Cugola, G., and Margara, A. Tesla: a formally defined event specification language. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems* (New York, NY, USA, 2010), DEBS '10, ACM, pp. 50–61.

[8] Cugola, G., and Margara, A. Low latency complex event processing on parallel hardware. *J. Parallel Distrib. Comput. 72*, 2 (Feb. 2012), 205–218.

[9] du Mouza, C., Litwin, W., and Rigaux, P. SD-Rtree: A Scalable Distributed Rtree. In *Proc. of IEEE 23rd International Conference on Data Engineering* (Apr. 2007), ICDE 2007, pp. 296–305.

[10] Güting, R. H., Böhlen, M. H., Erwig, M., Jensen, C. S., Lorentzos, N. A., Schneider, M., and Vazirgiannis, M. A Foundation for Representing and Querying Moving Objects. *ACM Trans. Database Syst. 25*, 1 (Mar. 2000), 1–42.

[11] Haklay, M., and Weber, P. Openstreetmap: User-generated street maps. *Pervasive Computing, IEEE 7*, 4 (oct.-dec. 2008), 12 –18.

[12] Hendawi, A. M., and Mokbel, M. F. Panda: A Predictive Spatio-Temporal Query Processor. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems* (New York, NY, USA, 2012), SIGSPATIAL '12, ACM, pp. 13–22.

[13] Hirzel, M. Partition and compose: parallel complex event processing. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems* (New York, NY, USA, 2012), DEBS '12, ACM, pp. 191–200.

[14] Hong, K., Smaldone, S., Shin, J., Lillethun, D. J., Iftode, L., and Ramachandran, U. Target container: A target-centric parallel programming abstraction for video-based surveillance. In *ICDSC* (2011), pp. 1–8.

[15] Jayaram, K., Jayalath, C., and Eugster, P. Parametric subscriptions for content-based publish/subscribe networks. In *Middleware 2010* (2010), I. Gupta and C. Mascolo, Eds., vol. 6452 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 128–147.

[16] Jeung, H., Yiu, M. L., Zhou, X., and Jensen, C. S. Path prediction and predictive range querying in road network databases. *The VLDB Journal 19*, 4 (Aug. 2010), 585–602.

[17] Johnson, D. S. Approximation algorithms for combinatorial problems. In *Proceedings of the fifth annual ACM symposium on Theory of computing* (New York, NY, USA, 1973), STOC '73, ACM, pp. 38–49.

[18] Koch, G. G., Koldehofe, B., and Rothermel, K. Cordies: Expressive event correlation in distributed systems. In *Proceedings of the 4th ACM International Conference on Distributed Event-Based Systems* (New York, NY, USA, 2010), DEBS '10, ACM, pp. 26–37.

[19] Koldehofe, B., Ottenwälder, B., Rothermel, K., and Ramachandran, U. Moving Range Queries in Distributed Complex Event Processing. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems* (New York, NY, USA, 2012), DEBS '12, ACM, pp. 201–212.

[20] Li, G., and Jacobsen, H.-A. Composite subscriptions in content-based publish/subscribe systems. In *Middleware '05: Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware* (New York, NY, USA, 2005), Springer-Verlag New York, Inc., pp. 249–269.

[21] Meissen, U., Pfennigschmidt, S., Voisard, A., and Wahnfried, T. Resolving Knowledge Discrepancies in Situation-aware Systems. *Int. J. Pervasive Computing and Communications 1*, 4 (2005), 327–336.

[22] Muthusamy, V., Petrovic, M., Gao, D., and Jacobsen, H.-A. Publisher mobility in distributed publish/subscribe systems. In *Proceedings of the Fourth International Workshop on Distributed Event-Based Systems (DEBS) (ICDCSW'05) - Volume 04* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 421–427.

[23] Pfoser, D., Jensen, C. S., and Theodoridis, Y. Novel Approaches in Query Processing for Moving Object Trajectories. In *Proceedings of the 26th International Conference on Very Large Data Bases* (San Francisco, CA, USA, 2000), VLDB '00, Morgan Kaufmann Publishers Inc., pp. 395–406.

[24] Pietzuch, P., Ledlie, J., Shneidman, J., Roussopoulos, M., Welsh, M., and Seltzer, M. Network-aware operator placement for stream-processing systems. In *Proceedings of the 22nd International Conference on Data Engineering* (Washington, DC, USA, 2006), ICDE '06, IEEE Computer Society, pp. 49–.

[25] Pietzuch, P. R., Shand, B., and Bacon, J. Composite Event Detection as a Generic Middleware Extension. *Network, IEEE 18*, 1 (jan/feb 2004), 44 – 55.

[26] Pillai, P. S., Mummert, L. B., Schlosser, S. W., Sukthankar, R., and Helfrich, C. J. Slipstream: scalable low-latency interactive perception on streaming data. In *Proceedings of the 18th international workshop on Network and operating systems support for digital audio and video* (New York, NY, USA, 2009), NOSSDAV '09, ACM, pp. 43–48.

[27] Ramachandran, U., Hong, K., Iftode, L., Jain, R., Kumar, R., Rothermel, K., Shin, J., and Sivakumar, R. Large-Scale Situation Awareness With Camera Networks and Multimodal Sensing. *Proceedings of the IEEE 100*, 4 (april 2012), 878 –892.

[28] Rizou, S., Diirr, F., and Rothermel, K. Fulfilling end-to-end latency constraints in large-scale streaming environments. *IEEE International Performance Computing and Communications Conference 0* (2011), 1–8.

[29] Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. The case for VM-based cloudlets in mobile computing. *IEEE Pervasive Computing 8*, 4 (October - December 2009).

[30] Tariq, M. A., Koch, G. G., Koldehofe, B., Khan, I., and Rothermel, K. Dynamic publish/subscribe to meet subscriber-defined delay and bandwidth constraints. In *Proceedings of the 16th international Euro-Par conference on Parallel processing: Part I* (Berlin, Heidelberg, 2010), EuroPar'10, Springer-Verlag, pp. 458–470.

[31] Wang, Z., Zhang, Y., Chang, X., Mi, X., Wang, Y., Wang, K., and Yang, H. Pub/sub on stream: a multi-core based message broker with qos support. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems* (New York, NY, USA, 2012), DEBS '12, ACM, pp. 127–138.

[32] Wolfson, O., Sistla, A. P., Chamberlain, S., and Yesha, Y. Updating and querying databases that track mobile units. *Distrib. Parallel Databases 7* (July 1999), 257–387.