MCEP: A Mobility-Aware Complex Event Processing System

BEATE OTTENWÄLDER, BORIS KOLDEHOFE, and KURT ROTHERMEL,
University of Stuttgart
KIRAK HONG, DAVID LILLETHUN, and UMAKISHORE RAMACHANDRAN,
Georgia Institute of Technology

With the proliferation of mobile devices and sensors, complex event proceesing (CEP) is becoming increasingly important to scalably detect situations in real time. Current CEP systems are not capable of dealing efficiently with highly dynamic mobile consumers whose interests change with their location. We introduce the distributed mobile CEP (MCEP) system which automatically adapts the processing of events according to a consumer's location. MCEP significantly reduces latency, network utilization, and processing overhead by providing on-demand and opportunistic adaptation algorithms to dynamically assign event streams and computing resources to operators of the MCEP system.

Categories and Subject Descriptors: C.2.1 [**Computer Communication Networks**]: Network Architecture and Design—*Distributed networks*; C.2.4 [**Computer Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Mobility, complex event processing, migration, moving range queries

ACM Reference Format:
Beate Ottenwälder, Boris Koldehofe, Kurt Rothermel, Kirak Hong, David Lillethun, and Umakishore Ramachandran. 2014. MCEP: A mobility-aware complex event processing system. ACM Trans. Internet Technol. 14, 1, Article 6 (July 2014), 24 pages.
DOI: http://dx.doi.org/10.1145/2633688

1. INTRODUCTION

At present, we face an explosive growth of sensors that become accessible to applications. Billions of users drive vehicles and carry mobile devices (i.e., smartphones and tablets) with a broad variety of on-board sensors. In addition, large-scale sensor deployments are spreading all around the globe, for example, hundreds of cameras in the Atlanta metro area. The ability to monitor and react to events detected with the help of such massive sensor deployment is the key to enable future *mobile situation awareness (MSA) applications* in many domains, such as traffic monitoring, social platforms, and health care. For example, a traffic monitoring application can notify drivers of nearby live road conditions, warning of traffic, accidents, or obstructions on the road. Social platforms can be enriched by incorporating automated live updates about activities of friends, for example, by streaming a video of a friend when she participates in a public sports event.

This work is supported by contract research "CEP in the Large" of the Baden-Württemberg Stiftung.
Authors' addresses: B. Ottenwälder (corresponding author), B. Koldehofe, and K. Rothermel, Institute of Parallel and Distributed Systems, University of Stuttgart; emails: {beate.ottenwaelder, boris.koldehofe, kurt.rothermel}@ipvs.uni-stuttgart.de; K. Hong, D. Lillethun, and U. Ramachandran, College of Computing, Georgia Institute of Technology; email: {khong9, davel, rama}@cc.gatech.edu.

DOI: http://dx.doi.org/10.1145/2633688

Complex Event Processing (CEP) [Luckham 2001] is a key paradigm to detecting events of interest for MSA applications, such as the occurrence of accidents, from basic sensor streams that capture changes in sensor measurements (e.g., vehicle speeds or camera frames). CEP commonly detects events using a set of operators. Each operator takes streams of events as input, processes them in order to detect new events, and produces event streams that include all detected events. More complex events can be detected when operators process the event streams produced by other operators. The event flow is then typically described by connecting operators to each other in a graph. This simple model has a number of advantages towards the scalable detection of events. Event streams of operators can be reused to detect multiple events of interest for applications. Moreover, the operators of the graph can be flexibly placed in a network of processing nodes [Koch et al. 2010] to optimize the bandwidth usage of CEP deployments or other optimization criteria.

Conversely, state-of-the-art CEP systems have significant shortcomings in their support for highly dynamic mobile consumers and sensors in Mobile Situation Awareness (MSA) applications. In order to ensure the consistency of produced event streams, the sensor streams selected as input to the operators need to be statically defined (e.g., incoming event streams are selected with respect to a fixed interest like a predefined road segment). However, the *spatial interest* of MSA applications constantly changes as consumers reacting to events change their location. In MSA applications, the interest in events is commonly defined with respect to the location of a *focal point* (e.g., a consumer is interested in all accidents that happened within the last 30 minutes in a 500-meter radius around its location). At present, changes in the location need to be explicitly handled by the MSA application, which can impose significant inefficiencies, such as a highly redundant processing and streaming of events.

In this work, we combine and extend the findings of our previous work [Koldehofe et al. 2012; Ottenwälder et al. 2013; Hong et al. 2013b] to the *MCEP system*, a middleware that enables dynamic adaptations of the operator graph as mobile consumers and producers change their location. In particular, we present methods that allow the system to dynamically change the event streams selected as input to operators and ensure the consistency of produced event streams. Furthermore, the mobile CEP (MCEP) system provides mechanisms that optimize the deployment in order to reduce the detection latency of events and the bandwidth usage to stream events.

The rest of this article is structured as follows: Section 2 discusses the relevant background and related work. Section 3 describes the architecture of our system. Section 4 covers how we perform the mobility-aware event processing. Section 5 discusses various optimizations. Section 6 covers our evaluation results before we conclude in Section 7.

2. BACKGROUND

In this section, we first analyze the requirements of mobile situation awareness applications and then discuss to what degree existing solutions can already support those applications. We conclude with the challenges tackled by the MCEP system.

2.1. Mobile Situation Awareness Applications

There is a vast number of MSA applications stemming from very different application domains. Traffic applications can analyze recent location information shared by vehicles to detect traffic information, such as accidents or traffic jams, and use this information to enrich navigation systems in vehicles [Koldehofe et al. 2012]. Smart surveillance applications can support police officers by displaying video streams on their smart phones showing suspicious people who were recently nearby [Hong et al. 2011]. Health-monitoring applications can infer the health status of a patient from

sensor data like the heart rate, other vital signs, or sensors in the environment (e.g., monitoring how weather changes affects a patient with severe allergies). Social applications can collect information, like the spatial distance to friends, or provide live video streams of friends.

All of these applications have considerably similar requirements. In contrast to many traditional applications, where a consumer receives a finite set of results from individual one-shot queries (e.g., a street-map), MSA applications have to be supported with continuous queries. Such continuous queries react to changes in sensor data and continuously push detected *situational information* (e.g., information about traffic), to the application. Moreover, they typically detect situational information from sensor data of nearby (mobile) sensors to inform (mobile) consumers in near real time. This *spatial interest* in sensor data typically changes with the location of a consumer. A considerable amount of relevant sensor data is also historical, since situational information is detected over historical and current sensor data (e.g., a downwards trend in the recent average speed of vehicles in a region indicates an upcoming traffic jam). In many cases, even historical situational information is still relevant to the consumer in the future, like a recent accident.

Continuous queries that support MSA applications thus should be able to ensure the following.

—They should continuously detect application-specific situational information on sensor data selected from spatial regions relative to the location of a *focal point* (e.g., a mobile consumer wants to query for recent and nearby accidents). Moreover, they should adapt the spatial interest and the detection of situational information according to the location of the focal points.
—They should ensure the temporal completeness and spatial consistency of situational information. For example, when detecting a downward trend in speed sensor data within the last 15min in a 500m radius, all situational information with respect to that time span and spatial range should be delivered when the spatial interest is adapted to a new location. To avoid false positives, the downward trend should not be detected on a selection of sensor data that contains speed sensor data with respect to both the previous and current spatial interest.
—They should process in a bandwidth and computation efficient way in order to be scalable and thus be able to execute a huge set of heterogeneous queries in parallel. For example, processing sensor data for spatial regions that are not of interest to any consumer should be avoided.
—They should deliver situational information at low latency. For example, an accident that happens in close proximity to a vehicle should be immediately detected and reported to a driver to prevent harm. Moreover, when adapting the processing to a new spatial range, the consumer should be immediately provided with historical situational information that has to be delivered to ensure said completeness.
—They should provide a notion of quality which expresses how closely the delivered situational information matches the actual interest of a consumer. For example, it could be sufficient for a consumer to be informed about traffic with respect to a region that only partially overlaps with the spatial interest if enough sensor data is contained in the overlapping region.

In the remainder, we will focus on two example queries. In most cases, it will be on a traffic scenario: “Retrieve all accidents that happened within the last 30min within 500m around my current location”. At times we will also focus on a social scenario: “Retrieve all videos of my friends that were within 200m around my current location within the last 10min”.

2.2. Continuous Query Processing

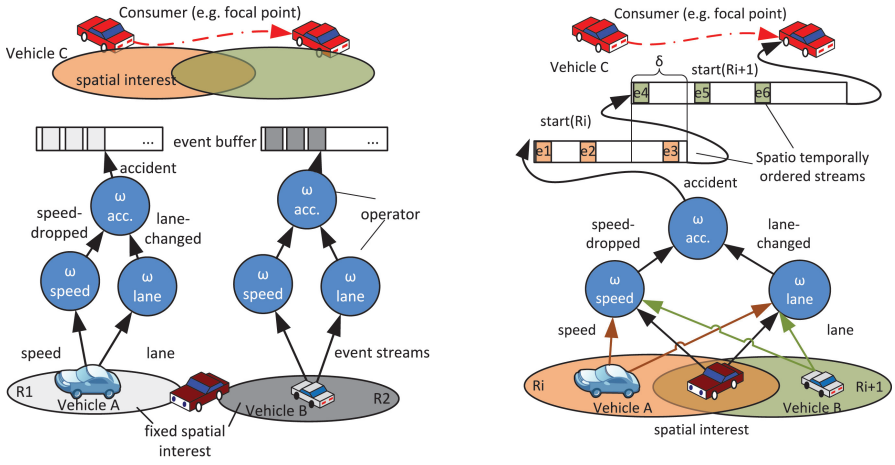
Two types of continuous queries that have been extensively studied in literature already offer support to MSA applications. First, *moving range queries* allow for expressing a dynamic spatial interest in sensor data, yet do not provide the means to detect situational information in the selected data. Second, *event processing queries* detect situational information in selected sensor data, yet are not flexible enough to cope with the dynamics of mobile consumers and sensors.

2.2.1. Moving Range Queries. *Moving range queries* [Xiong et al. 2007; Gu et al. 2009] return (sensor) data in a consumer-specified range as a result and update this result depending on the location of a moving consumer, denoted as a *focal object*. For example, a moving consumer is always updated with the nearest taxis in a 500m radius. These queries can provide filtered and aggregated sensor data [Xiong et al. 2007; Gedik and Liu 2006; Sun et al. 2004] to support MSA applications (e.g., with a count on the number of taxis in the range). On the downside, they yet lack the expressiveness to detect situational information on the selected sensor data. Moreover, range queries can provide spatial completeness of sensor data, but MSA applications also require a temporal completeness on detected situational information. This requires, for example, to detect historical accidents on historical events without missing one. Furthermore, minimizing the number of location updates [Cheema et al. 2010; Xu and Jacobsen 2007] from a mobile device like the taxi to a server has been proposed to reduce the mobile devices energy consumption, but said temporal completeness as well as communication and computing costs of historical sensor streams have been disregarded. Predictive queries [Hendawi and Mokbel 2012] give results describing the future of the focal object (e.g., the taxis that will be in a 500m radius around the consumer 5min from now). Similar methods can be used to opportunistically precompute historical situational information with a low latency. However, due to the inherent uncertainty in future locations, achieving low latency requires combining partially precomputed results with on-demand results at the future location. This is typically not possible for situational information (e.g., an average speed cannot be calculated for a future location without tailoring the combination towards this operation).

2.2.2. Event Processing. *Complex event processing*[1] systems take *atomic events* (e.g., changes in sensor data) that are streamed by *sources* (e.g., sensors) as input, detect events of interest (*complex events*, e.g., situational information), and forward them to a *consumer*. For example, by observing an increased stock value above a threshold, an event can be detected that recommends selling the stock. The detection is performed by operators that process a continuously changing selection of input events (e.g., by sequence operators [Wu et al. 2006] that detect person A entering a room before person B, or custom operations [Neumeyer et al. 2010]). Operators produce zero or more new events as output. Individual operators (nodes in Figure 1), sources, and consumers (vehicles in Figure 1) are connected by event streams (edges in Figure 1) to an *operator graph* [Koch et al. 2010].

Mobility-Driven Adaptation. Sources, operator graphs, and consumers are often statically coupled. In order to support MSA applications, a set of operator graphs has to be deployed that can only process events selected from a fixed spatial interest, as shown in Figure 1(a). The consumer, who also poses the focal point in the example, can fetch the historical situational information from a buffer with respect to a spatial range that bears the most overlap with the spatial interest of this consumer. However, two

[1]Following the results in unified models [Cugola and Margara 2012], we use *stream processing* [Babcock et al. 2002] and *active data bases* [Chakravarthy and Mishra 1994] as synonyms and focus on similarities.



(a) Static deployment example. (b) MCEP query example.

Fig. 1. Accident detection example: Each vehicle is a source and continuously reports its speed and location, including its identifier, to the operator graph(s) that process events with respect to the vehicle's current location. By processing those events, two leaf operators detect a decreased speed and a lane switch of each vehicle. The root operator incorporates the speed and lane information to detect an accident if many vehicles reduced their speed and avoided a specific lane around the same time and location.

operator graphs process and stream events in this example, although only one query is deployed, which is inefficient in terms of communication and computation. Moreover, the quality of the results is unpredictably degraded, since the queries' spatial interest is highly unlikely to match the predefined ranges of the operator graphs.

Dynamic reconfigurations of operator graphs mostly focused on adapting an operator graph to dynamic data rates. Typical solutions are changing the order of operators [Avnur and Hellerstein 2000] or the access pattern on event streams [Dindar et al. 2011]. These solutions therefore do not support the adaptation to new spatial interests. Up to now, Borealis [Abadi et al. 2005; Sheykh Esmaili et al. 2011] offers the best support for MSA applications. The system supports on-the-fly query modifications by altering or replacing processing functions which can be used to adapt an operator to a new location. Furthermore, the system also supports time-travel methods which rewind a continuous query and start it in the past again, as well as dynamic revisions of query results, which means that only deltas (insertions or removals of events) have to be sent when performing a time travel. This can be used to perform optimized adaptations of operator graphs when a focal point updates the location with the system. However, the system does not provide methods to automatically decide how far the system has to rewind, limiting their revision to nonspatial individual results, nor does this work study the implications on the savings in communication costs and the degradation of the quality of events if the revision is not performed.

Completeness and Consistency. Completeness and consistency is typically addressed to make systems robust to failure [Koldehofe et al. 2013; Sebepou and Magoutis 2011] or to ensure event orderings by blocking operators and revising events [Barga et al. 2007]. This does not cover spatial inconsistencies as addressed by this work, which occur, for example, if an individual operator graph for each focal point is used and events of an updated spatial interest are streamed without updating the operator graph. Note that operators keep events in histories [Adi and Etzion 2004], that is, a partial result of a processing step [Pietzuch et al. 2004; Arasu et al. 2003] or events

from their incoming streams. Consider now an operator that detects a critical number of vehicles that reduced their speed, which probably indicates an upcoming traffic jam. To this end, the operator may keep a count on such incidents. This count can vary if the operator dynamically updates its spatial interest.

Latency and Communication-Efficient Processing. Improvements in end-to-end latencies and communication costs are achieved in these systems by placing operators on computing nodes close to sources of high network traffic or delay—typically the sensors [Cugola and Margara 2013; Hummer et al. 2011; Lakshmanan et al. 2008]. Most of them are tailored for scenarios with infrequent changes, that is, where sensors and consumers are not mobile. Therefore either the impact of migration costs on the placement strategy or an amortization of these costs is not considered. Consider a focal object that frequently changes the access point to the system, which would trigger a vast number of migrations to keep the operators near the sensors. Operators can have a large state, for example, they can keep many events in their history or keep a large *immutable state*—the part of the operator that is read-only and fixed in size (e.g., a street map or a database for face recognition). If the state of operators grows to several gigabytes while only a few megabytes of data are streamed to the operator before the next migration is triggered, then the former costs outweigh the latter and the system suffers from a drastic performance decrease. Since migrating operators require first stopping the operator at its old host, then transfer the large state to the migration target, before finally restarting the operator at the target, the consumer also perceives a high end-to-end latency in this scenario.

2.3. Problem Description

The problem addressed with the MCEP system is to efficiently support MSA applications with a middleware that allows querying for temporally complete and spatially consistent, dynamic situational information that yields a consumer-acceptable quality in the result and latency. In addition, the algorithms providing the execution environment for those queries should guarantee low latencies to deploy the operator graphs for new spatial interests, at best with near-zero latency. Moreover, they should ensure low communication and computation costs, that is, streaming and processing as few historical events as possible, and amortize migration costs.

3. ARCHITECTURE

We now detail the systems architecture, including a discussion on the MCEP model, the relevant mobility-aware delivery semantics, and its components.

3.1. MCEP Query Model: Dynamic Interest Queries

Situational information is a set of complex events that are delivered to a consumer. Their detection is modeled by a directed, acyclic operator graph, $G = (\Omega \cup S \cup U, L)$. Ω denotes the set of operators, S the set of sources (e.g., a range query), U the set of consumers, and $L \subseteq (\Omega \cup S \times \Omega \cup U)$ the event streams.

To receive situational information relative to her current location, a consumer registers a *dynamic interest query*, $diq = (G, fo, R, \delta, Sem, QoR)$ with the MCEP system. G denotes the application-specific operator graph, fo the focal point of a consumer, R a spatial interest which is spanned by fo, δ a lifetime parameter that controls the amount of delivered historical situational information, Sem the delivery semantics, and QoR a parameter that allows specifying how closely R should match the actual input of G, denoted as *processing interest*.

In the context of the introduced accident example, a vehicle's head-unit registers a diq with the CEP system for all accidents within a range of 500m around its current

location (see Figure 1(b)). To continuously provide situational information while a consumer is moving, the vehicle provides discrete location updates[2], resulting in a sequence of discrete changing spatial interests $R_1, \ldots, R_m$. The operator graph will then stop processing for the spatial interest R_i and start processing for the consumer's updated spatial interest R_{i+1}, which we further denote as *operator graph switch*. The lifetime parameter δ allows us to detect events that occurred δ time-units before the operator graph switch happens at time $start(R)$. Consider an accident that occurred 30min before the spatial interest includes the location at which the accident happened. This accident is still of interest to the consumer, as it can still block a lane.

MSA applications require a different *delivery semantics* (*Sem*), in terms of *event ordering*, *consistency* of processing state, and *completeness* to account for the dynamically changing spatial interests. Since temporal consistency guarantees and event orderings [Barga et al. 2007] are well known, we focus on the relevant mobility semantics, a *spatiotemporal* event ordering and *spatial consistency* which can be selected by a consumer. In order to distinguish situational information, they are deterministically stamped with the *spatial interest* and the maximal time-stamp from events that are used to detect situational information. In the remainder, we formally define event ordering, spatial consistency, and completeness. For a region R_i, let $[start(R_i), start(R_{i+1})]$ denote the time interval between the location update that leads to R_i and the subsequent one R_{i+1}. Furthermore, let $E(R_i) := (e_{(1,R_i)}, \ldots, e_{(max,R_i)})$ denote a possible sequence of temporally ordered situational information for which G processes events selected from R_i.

(1) A *spatiotemporal ordering* accounts for the sequence of processing interest changes and the temporal ordering within spatial interests. Our system ensures that all situational information with respect to R_i is delivered before situational information with respect to R_{i+1}. Moreover, all situational information $e_{(j,R_i)}$ for R_i with index $j < j'$ are delivered before $e_{(j',R_i)}$. In Figure 1(b), events are delivered in a spatiotemporal order: $e_1, e_2, e_3, e_4, e_5, e_6$.
(2) *Spatial consistency* ensures that any operator $\omega \in G$ only processes a selection of incoming events that are either atomic events stemming from the same spatial interest R_i or are produced as a result of processing such atomic events by their preceding operators. In Figure 1(b), it means that no operator processes a selection of incoming events that comprises events stemming from vehicle A and vehicle B.

If all situational information streamed in a spatiotemporal ordering with respect to a processing interest and lifetime parameter is delivered, the result is said to be *temporally complete*. Naturally, ensuring temporal completeness results in an increased latency for operator graph switches, since a potentially huge number of historical events have to be processed before live events can be processed. We therefore discuss a strict completeness guarantee and a relaxation, which we call best effort completeness.

(1) We say $E(R_i)$ is *temporally complete*, if and only if for all $e \in E(R_i)$, the temporal restriction with respect to the events' time-stamp $t(\mathrm{e}) \in [start(R_i) - \delta, start(R_{i+1})]$ holds and no other sequence $E'(R_i) \supset E(R_i)$ exists, where $t(\mathrm{e}') \in [start(R_i) - \delta, start(R_{i+1})]$ with $\mathrm{e}' \in E'(R_i)$ holds.
(2) A relaxed definition is the *best effort completeness*. The system delivers all situational information $E(R_i)$ with respect to the current spatial interest R_i that can be produced until an operator graph switch occurs. This can result in temporally incomplete data when not enough resources are available to process all (historic)

[2]As surveyed in Leonhardi and Rothermel [2001], there already exist numerous location update strategies.

atomic events (e.g., e_3 in Figure 1(b) could be missing). Yet, consumers are typically only interested in situational information of their current spatial interest.

Recall that not all MSA applications require exactly matching the spatial interest with the actual input to the operator graph. This observation allows us to decouple the actual spatial interest R from the processing interest R_a that selects the atomic events as input to G. The consumer can specify with the *quality of results (QoR)* parameter a relaxation on the overlap of R_a and R using the well-known metrics *precision* and *recall* from the information retrieval domain. The precision describes how noisy the set of selected atomic events is. Recall indicates how relevant the set of selected atomic events is. Formally, if $A(R)$ is the spatiotemporally complete sequence of atomic events selected by R and $A(R_a)$ is the sequence of atomic events selected by R_a, then

$$precision(R, R_a) = \frac{|A(R) \bigcap A(R_a)|}{|A(R_a)|}, \tag{1}$$

$$recall(R, R_a) = \frac{|A(R) \bigcap A(R_a)|}{|A(R)|}. \tag{2}$$

For the example in Figure 1(b), the system might omit the update for R_{i+1} and continue processing events from R_i, reducing the precision and recall to approximately 0.5 if all three sources emit the same amount of events. Note that the overlap of both ranges can be low, but the resulting situations may still be meaningful when precision and recall are close to 1 (i.e., all relevant events for the processing are in this overlap).

3.2. Broker Infrastructure

Operator graphs are deployed on a federation of k distributed brokers $\{b_1 \dots b_k\}$. Similar to typical mobile infrastructures (e.g., GSM networks or location services), brokers are organized in a spatially-partitioned hierarchy [du Mouza et al. 2007]. Such a broker hierarchy can be implemented using virtualized resources in cloud data centers or at the edge of the network.[3] This hierarchy implies that the communication delay to mobile consumers decreases from the edge to the core of the network [Hong et al. 2013a].

Mobile consumers and sensors share event streams over a wireless interface with the broker hierarchy. In order to improve their expected link quality, they greedily connect to the topologically closest broker that hosts the MCEP system, denoted as leaf broker. Since processing tasks are deemed to consume a lot of energy, those mobile devices are only thin clients, leaving all the processing to the infrastructure.

3.3. System Components

Figure 2 shows the logical structure of the MCEP event processing architecture. The main component is the *MCEP server* which runs on each broker. Operators of an operator graph G and a range query for R are executed by the servers *execution environment* (see Section 4). Operators and range queries that are executed in the same execution environment can also share the processing of events if they are the same type by using similar methods to Hong et al. [2009] and Hendawi and Mokbel [2012]. As a result, the consumer is continuously updated with situational information.

The system includes a spatiotemporal *event storage* in which atomic and complex events are stored. By retrieving atomic events from the event storage, our system can also detect historical situational information. In addition, storing complex events allows our system to reduce the latency in detecting historical situational information.

[3]Such in-network virtualization is currently promoted by companies like Cisco [Bonomi et al. 2012].

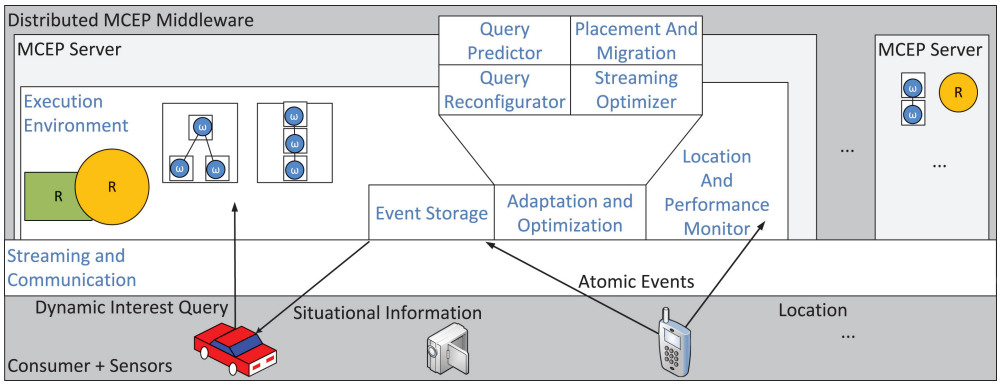


Fig. 2. Architecture overview: Mobile consumers register continuous queries with the MCEP server. The latter performs the processing on their behalf based on sensor data (atomic events) from mobile sources.

All events in the spatiotemporal event storage are indexed by location, time, and type attributes. Events are evicted from the event storage when no operator graph is expected to request them. Moreover, events can be persisted in a database for applications that are interested in long-term analysis of events. A *streaming* component mediates the event streams between the event storage, consumers, operators, and range queries according to the operator graph. Events can, for example, be streamed over a network link, since operators, range queries, and the event storage can be executed on different MCEP servers.

The *location and performance monitor* continuously monitors the location of mobile consumers and sensors, as well as the systems' performance (e.g., current and predicted event rates between operators, streaming latencies, and CPU loads). Changes in those monitored parameters trigger the system's *adaptations and optimizations* (see Section 5).

Location updates trigger the *query reconfigurator* which initiates the operator graph switch. The implemented adaptation algorithms ensure temporally-complete and spatially-consistent results and trade off QoR to communication and computation costs. Deviations in predicted locations and monitored performance metrics trigger the *placement and migration* component. The component decides for each spatial interest which MCEP servers host the operators of G and determines the time when a migration has to be initiated. The implemented algorithms preserve low communication costs by constantly adapting the placement and a low latency migration by opportunistically starting the migration before the operator needs to be active on a new MCEP server. To reduce the communication and computation overhead during an operator graph switch, the *streaming optimizer* prevents the streaming of events that have already been sent for a previous spatial interest. The *query predictor* is triggered with each location update to support the query reconfigurator and to reduce the latency during operator graph switches.

4. MOBILITY-AWARE OPERATOR EXECUTION

This section covers the concepts of the *execution environment* and the *query reconfigurator* [Koldehofe et al. 2012]. We detail in particular how operators process events while ensuring temporal completeness and spatial consistency. In Section 4.1, we introduce the concepts for the operator execution and how the system selects the input to the operator by implementing a window on each incoming stream. In Section 4.2, we then detail the basic mobility-driven adaptation algorithm that uses the dependency

of windows over several levels in the operator graph to ensure temporal completeness. To ensure spatial consistency and a spatiotemporal event ordering, we propose the concept of inducing markers into event streams to isolate the processing of events for each individual processing interest.

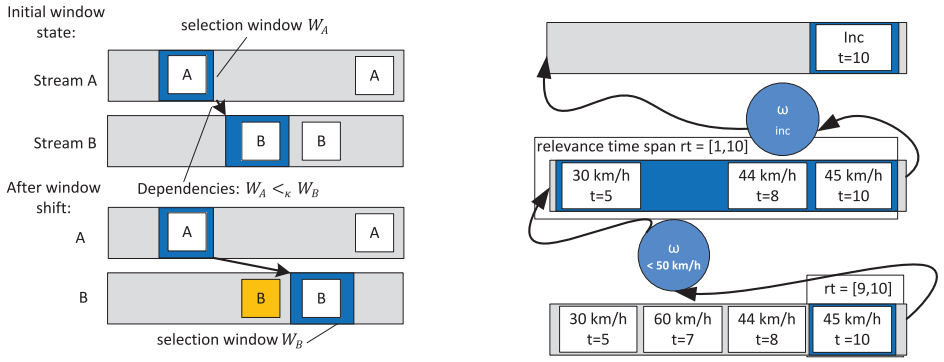
4.1. Operator Execution

The execution of an operator is characterized by performing a sequence of *correlation steps*. In each correlation step, the operator applies its correlation function—which implements the application-specific operator logic—on a selection of events from its incoming event stream to produce a set of zero or more outgoing events. The execution environment of the MCEP system controls these correlation steps by managing the event stream and identifying selections on behalf of the operator. To this end, the MCEP system keeps a buffer for each incoming event stream, which asynchronously adds events delivered from preceding operators.[4] A new correlation step is initiated at the end of the previous correlation step by determining the next selection. An operator is explicitly informed by the MCEP system about the start of each new correlation step, which allows the operator to clean up stale processing state from a previous correlation step. The correlation function can then start processing the events comprised in the selection. When all events of a selection are streamed to the operator, the MCEP system explicitly informs the operator about the end of a selection. The execution environment receives as a result a signal when the operator finished processing the events of a selection, which then triggers an optional eviction of events from the incoming buffers. The MCEP system delivers and stamps the outgoing events produced by an operator with the largest time stamp comprised in the selection and, to avoid ambiguities, with a sequence number. Note that other time-stamp mechanisms are applicable, but for the sake of clarity, we rely on this mechanism for the rest of the article.

To determine a selection in the MCEP system, the programmer is obliged to specify a *selection window* for each incoming stream of the operator, which determines zero or more events that are included in the selection. In particular, the programmer defines a counting or temporal selection window for each incoming stream of the operator. For instance, a window can select a number of subsequent events of the stream bounded by a maximal count or a sequence of all events of the stream, where the difference in the time-stamp between the first and last event included in the window is smaller than a fixed time span. Consider Figure 3(a): it depicts a simple operator that selects events from stream A before an event from stream B, where each stream has its own counting window that selects exactly one event.

A *selection policy* is applied in order to determine a new selection and defines for each selection window individually how it shifts on the event stream, that is, the window shifts for a fixed amount of events, a fixed time, to the first event of the event stream, or to the last event of the event stream. Each shift, however, is bound by a supplemental *temporal dependency*, defined for different incoming streams. These allow a programmer to define causal dependencies between a pair of windows W, W' in form of a relation $<_\kappa$. The relation indicates that all events selected by W have to comprise smaller ($W <_\kappa W'$) or larger time stamps ($W' <_\kappa W$) than the events selected by W'. Cyclical dependencies (e.g., $W <_\kappa W' <_\kappa W'' <_\kappa W'$ over a third window W'') are prohibited. A *consumption policy* allows programmers to explicitly evict events from incoming buffers, that is, it allows to evict a fixed number beginning with the first (or last) of the selected events, a set of events specified in the signal, or all selected events.

[4]To ensure a temporal ordering over all incoming streams, the system relies on FIFO channels between operators, queues incoming events, only inserts queued events in the corresponding incoming buffers if over all FIFO channels events with fresher time stamps arrived, and uses heart beat messages to ensure progress.



(a) Example: Events are selected by a counting window that selects exactly one event. After a processing step the windows shift.

(b) Example: Events depend on historical atomic events, Inc at t=10 depends on 30 km/h at t=5.

Fig. 3. Selection examples.

Additionally, we obligate that programmers define a *relevance time span rt* associated to a selection, a distinct temporal range $[begin(rt), end(rt)]$ on the incoming streams which limits the shift of the selection windows. For example, a selection window of operator ω_{inc} in Figure 3(b) is not allowed to select any event that has a larger time stamp than $t = 10$ and a smaller time stamp than $t = 1$. The relevance time span is updated whenever (i) no window can be shifted anymore within the relevance time span and (ii) when at least one event in the incoming buffers has a larger time stamp than $end(rt)$. The programmer can specify how the relevance time span is shifted when it requires an update. In particular, the relevance time span can shift by one event or it can shift by a fixed amount of time. In the context of the example of Figure 3(b), the next relevance time span can end at $t = 11$ or a fixed time after $t = 10$ when an event with time stamp $t = 11$ arrives. Selection windows are always initialized before a shift of the relevance time span by shifting to the most recent event on their respective buffer, however considering their relation given by $<_\kappa$.

Our system relies on a variety of windows with different consumption and selection policies on each incoming stream for a high expressiveness. Yet, we constraint the expressiveness with a relevance time span. We will show that such a constraint allows our system to ensure temporal completeness.

4.2. Location-Aware Adaptation of Operators

Recall that a consumer's location update triggers the operator graph switch performed by the query reconfigurator. To clarify the basic adaptation steps, we now describe how the operator graph switch is performed without relying on precomputed results from a query predictor. When informed about the new location, the query reconfigurator at the broker to whom the consumer is connected ceases the streaming of the range query for R_i. The query reconfigurator blocks until all events with smaller time stamps than $start(R_{i+1})$, the time associated to the location update, are streamed by the range query to ensure temporal completeness. Subsequently, the processing of events at the operators for the old range R_i is stopped by injecting a marker in each atomic event stream, which allows the system to isolate the streams for each individual processing range and clearly identify the end of a complete stream for R_i. The query reconfigurator then updates the range query, which similarly to Xiong et al. [2007], distributed over all MCEP servers that store relevant atomic events, with the new location. The query

reconfigurator then starts the processing of the operators for R_{i+1} by streaming historical atomic events to the operators from the event storage with respect to R_{i+1}. Each historical atomic event stream starts at a specific time, denoted as the *initialization point*. When all historical atomic events are streamed, the system seamlessly starts streaming live events. Live events that arrive at the MCEP servers during $start(R_{i+1})$ and the time the system switches to the live stream are buffered to avoid event losses.

4.2.1. Finding Initialization Points. Consumers define with *diq* their interest in historical situational information starting with time stamps greater than $start(R_{i+1}) - \delta$. The *initialization point*, however, does not coincide with this time, since situational information itself depends on events that lie even further back in the history. The dependency of complete situational information on historical (atomic) events is shown in Figure 3(b), where $start(R_{i+1}) - \delta$ is $t = 10$. For temporally complete situational information about smoother traffic, all atomic speed events with time stamps $t \geq 5$ are required in order to detect an increase in speed events with values <50 km/h.

Our basic approach to finding initialization points, therefore, considers dependencies on historical atomic events over several levels in the operator graph hierarchy. Due to the dependencies of outgoing events on historical events in selections, the time stamp of the temporally first event in a selection has to decrease in a consumer to source direction. However, a selection comprises only events whose temporal distance is smaller than the relevance time span. We therefore use the relevance time span definitions of the operators as a lower limit. In Section 5.3, we will discuss several approaches to determining an infimum on this lower limit.

For each possible path p through the operator graph from the consumer to an operator that selects atomic events as input, the basic algorithm to finding the initialization point determines an additional time $\Delta(p)$. The additional time $\Delta(p)$ sums the relevance time spans of each operator on p, that is, $\Delta(p) = \sum_{\omega \in p}(end(rt_\omega) - begin(rt_\omega))$. The initialization point for an atomic event stream ae with a path p_{ae}, from this atomic event stream to the root, is then simply calculated as $init(ae) = start(R_{i+1}) - \delta - \Delta(p_{ae})$. In the example, Figure 3(b), $\Delta(\omega_{<50}, \omega_{inc}) = 9 + 1$ and thus $init(speed) = 10 - 10 = 0$.

4.2.2. Updating the Spatial Interest. The *marker messages* inform the execution environments of the operators of G about the time stamp $start(R_{i+1})$ of the operator graph switch and a new spatial interest R_{i+1}. Each marker M implies that from now on, the streams contain only events that are selected and processed relative to R_{i+1}. Note that at this point in time, the operator may still need to produce events with respect to the old region to ensure temporal completeness (e.g., when the operator awaits events with a smaller time stamp than $start(R_{i+1})$ on a second incoming stream). The execution environment therefore prevents selection windows from shifting if they would select a marker M in the next correlation step. If no selection window can be shifted without selecting any marker M, the execution environment will produce a marker for all of its outgoing streams to initiate the reconfiguration of successors in G. Subsequently, all events of R_i are consumed, the operator is informed about R_{i+1}, and processing on the events of R_{i+1} is started by initializing the relevance time span at the initialization point of this operator. Note that best effort completeness allows our system to immediately cease processing for a range R_i when at least one stream comprises a Marker for R_{i+1}.

5. ADAPTATION AND OPTIMIZATION

The adaptation and optimization components of the MCEP system are targeted to minimize the communication and computation costs while preserving low end-to-end latencies and high Quality of Resultss (QoRs).

5.1. Location Monitor

To reduce the resource requirements of the system in streaming and processing historical events, our system can ignore processing interest updates. This savings, however, comes at the cost of a reduced accuracy with which a processing interest matches the actual spatial interest. By employing additional update conditions that regulate how often a location monitor informs the query reconfigurator about a location update of a consumer, the MCEP system can decide to trade-off system resources against QoR for each operator graph.

(1) The *basic update condition* performs the operator graph switch as soon as the focal point reports a new location. It maintains a high accuracy, as it reflects all location changes, but at the risk of many unnecessary reconfigurations if the underlying set of atomic event streams does not change.
(2) A *timed or spatial update condition* initiates the operator graph switch after some predefined time has passed or when the consumer has moved a predefined distance.
(3) A *QoR update condition* initiates the operator graph switch if the QoR calculated on the most recent atomic events in the spatial and processing interest is below the threshold given by the *diq*.

5.2. Streaming Optimizer

Note that two subsequent spatial interests R_i and R_{i+1} have a spatial overlap $R_i \bigcap R_{i+1}$ as well as a temporal overlap (at least) for events produced within $[start(R_{i+1}) - \delta, start(R_{i+1})]$. This can lead to *duplicates* after an operator graph switch, that is, the same event is streamed between operators multiple times. To deal with such overlapping event sequences, we distinguish between two classes of operators and present corresponding approaches that minimize the bandwidth that is required to stream historical events. First, *locality preserving operators* guarantee the following: For any pair of incoming event sequences E and E' which yields $E \subset E'$, all events produced with respect to E are also produced with respect to E'. For example, an attribute filter is part of that class. Second, for *non-locality preserving operators*, not all events produced with respect to E are necessarily also produced with respect to E'. This class includes aggregation operators. Furthermore, detected overlapping event sequences allow our system to reuse a correlation step with respect to R_{i+1} that has already been performed with respect to R_i in order to reduce the number of performed correlation steps.

Optimizations Using the Spatial Overlap. If $\omega \in G$ is a locality preserving operator, it indicates in a marker message to all its succeeding operators that events with location-stamps in $R_i \bigcap R_{i+1}$ can be reused until the event processing catches up with live events. To this end, the incoming stream of R_i is buffered in the event storage. Note, that for non-locality preserving operators, the optimization of the spatial overlap cannot be used. Consider that an operator counts the average number of cars in R_i within a time span of 10min on the selection e_1, e_2, e_4 and produces e_o. If at least one new event e_3 is integrated during the operator graph switch, that is, the selection is now e_1, e_2, e_3, e_4, the output is now changed to e'_o, and therefore e_o cannot be reused at the succeeding operator, although all events that caused it are within the new region R_{i+1}.

Optimizations Using the Temporal Overlap. To exploit the temporal overlap for non-locality preserving operators, we also buffer produced events in the local event storage. Each $\omega \in G$ verifies for each produced event whether the event has already been sent to its succeeding operator by looking it up in its outgoing buffers. If an event was already produced, we log it and forward the log to the succeeding operator after a time span t_s has passed. Within t_s, more events may be found, so the successors of ω can be notified by sending a message containing the log for those events in a single batch. An operator

receiving a log from its preceding operator will retrieve these events from the event storage.

5.3. Query Reconfigurator

The basic approach to ensuring temporal completeness (see Section 4) makes it possible that historical events can be processed unnecessarily. In Figure 3(b), the relevance time span of ω_{inc} can include many events with time stamps smaller than $t = 5$, which can lead to situational information with smaller time stamps than $t = 10$ (i.e., an over-provisioning of historical situational information). Our solution to overcoming this problem is to (i) also utilize the selection window definitions beyond the relevance time span to determine the initialization point of the atomic event stream *ae* and (ii) further relax temporal completeness—situational information at the beginning of the stream is only delivered with best effort.

The core idea is to keep an initialization point for each individual incoming stream of an operator based on its selection window W. The system can then determine the initialization points in the atomic event stream *ae* similar to the basic approach by determining an additional time $\Delta(p_{ae})$ for each path p_{ae} from the atomic event stream to the consumer. We therefore first determine a window-dependent time $\Delta(W_\omega)$, instead of $(end(rt_\omega) - begin(rt_\omega))$, for each individual operator ω on the path. The additional time is then determined as $\Delta(p_{ae}) = \sum_{\omega \in p_{ae}} \Delta(W_\omega)$. For temporal windows, this relates to their fixed time span. For counting windows, the system maps the count to an estimated time span. Therefore, the system monitors the interarrival times of recent events. To account for the dependencies of W_ω on several other windows W' through W^n with $W <_\kappa W' \cdots <_\kappa W^n$, the system increases $\Delta(p_{ae})$ by $\Delta(W_\omega) + \Delta(W') + \cdots + \Delta(W^n)$.

This approach reduces the number of streamed events, especially in scenarios with highly selective operators and many temporal dependencies. The approach often provides temporal completeness, but if, for example, the selectivity of operators and the event rates of the incoming streams are bursty, some situational information could be missed.

5.4. Query Predictor

We now discuss an algorithm for the query predictor [Hong et al. 2013b] that reduces the latency incurred by (i) switching the operator graph and (ii) processing a possibly huge number of historical events. To explain the basic principles of the algorithm, we first describe how the system uses predicted future locations of a focal point to configure operator graphs in advance. We then extend the mechanism to trade off computing resource usage against QoR. To this end, the system precomputes situational information for multiple future locations at each future point in time.

Predicting Operator Graph Deployments. With each location update triggered from the location monitor, our system predicts a future processing interest at a discrete time. The location monitor thus implements an additional location predictor (e.g., using a linear dead-reckoning algorithm [Leonhardi and Rothermel 2001]), which returns a predicted future location of the focal point. Our system deploys a copy of the operator graph and a range query for the future processing interest. The start of streaming for the future processing interest is scheduled with respect to the initialization point of the future processing interest.

Few changes have to be made to the query reconfigurator to deal with future processing interests. If the future processing interest is expected to maintain a QoR above the consumer defined threshold, calculated on the most recent atomic events, the system switches to this operator graph instead of adapting the current operator graph. The resources of the old location's range query and operators are then freed after processing

```
1: Query diq //consumer specific query
2: function determine_interests(Locations P_L)
   F ← ∅ //set of future processing interests
3:  //set of resource requirements per location
   S ← calc_initial_costs(P_L, R(diq))
4:  s_tot ← 0 //expected resource requirements
5:  while P ≠ ∅ do
6:      p ← select_and_remove_next(P, p, F, diq)
7:      if ∄f ∈ F :
          precision(p, interst(diq, f)) > QoR(diq)
          ∧recall(p,interst(diq, f)) > QoR(diq)
        then
8:          if s_tot + S[p] ≤ s_max then
9:              F ← F ⋃{interest(diq, p)}
10:             s_tot ← s_tot + S[p]
11:         end if
12:     end if
13:  end while
14:  return F
15: end
```

```
1: stable_plan, neighbor_plans, TG ← ∅
2: upon generate_plan( )
   //create time graph with nearby broker and
   //recent estimations on latency and bandwidth
   TG ← create_time_graph(TG, neighbor_plans)
3:  path ← determine_shortest_path(TG)
4:  //determine placements from vertices of path
   P ← create_migration_plan(path)
5: //a plan deviates iff placements differ
    if deviates_from(P, stable_plan) then
6:      stable_plan ← P
7:      trigger send plan_updated_msg(P)
          to all neighbors ∈ G
8:  end if
9: end
10: upon timeout(regular time interval)
11:  if check_monitoring() then
12:      trigger generate_plan()
13:  end if
14: end
```

Fig. 4. Algorithms for the query predictor and the migration and placement component. The function determine_interest() selects future processing interests to achieve a high QoR while preserving resource constraints. The function generate_plan() generates a plan by finding a shortest path in a time graph.

and sending the marker message. When all resources are freed, the query reconfigurator connects the opportunistically deployed future operator graph to the consumer. The system then delivers all precomputed historical situations from the event storage that have been detected so far and afterwards delivers situational information produced by the new operator graph.

Increasing the Expected QoR. Since a single future location is subject to high location uncertainties, that is, the location predictor is not a perfect oracle, the query reconfigurator can often encounter a future processing interest which produces situational information with a QoR that does not satisfy the threshold given by a *diq*. The system therefore deploys multiple operator graphs that simultaneously process atomic events selected by slightly different processing interests to increase the chance of finding a future processing interest that provides a suitable QoR.

The best result for the QoR is achieved when deploying an infinite set of future processing interests for each discrete future time, which is not practically feasible. We therefore deploy only a discrete number of processing interests around the predicted future location of the focal point. A set of equidistant locations that span processing interests with a high spatial overlap will provide a good QoR for a spatial uniform event distribution. Nevertheless, processing events from all those processing interests can still lead to significant resource usage. Moreover, events are typically not uniformly distributed, which can lead to redundant future processing interests. In particular, for each future spatial interest that is provided with an acceptable QoR by a redundant processing interest, another processing interest exists that provides an acceptable QoR.

Our greedy algorithm (see determine_interests() in Figure 4) therefore selects a minimal set of processing interests F from a predefined large number of locations P_L around the predicted location of the focal point. These processing interests cover many spatial interests with a QoR above the threshold which is defined by the consumers' query *diq*. Moreover, the algorithm keeps the expected computing resource usage below a system-defined limit s_{max}. The computing resource usage is estimated for each processing interest spanned by a location in P_L based on the recent atomic event load (Line 3); the system therefore expects that operator graphs are profiled using different atomic event loads before deployment. The algorithm iterates over all locations of the input

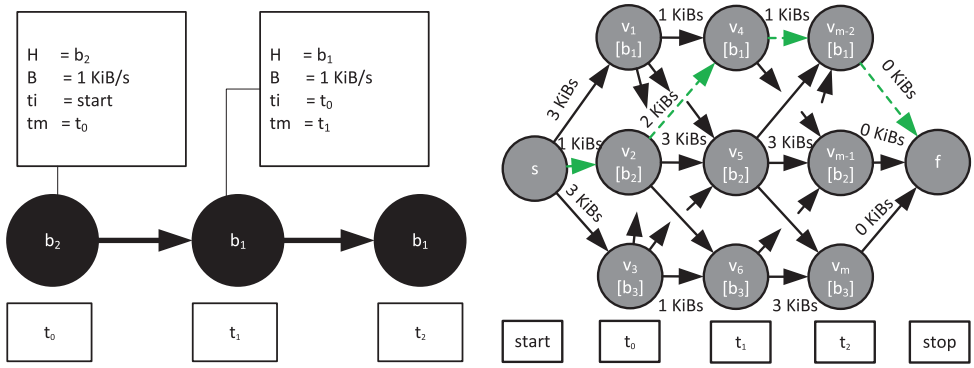
according to a policy that a programmer specified while developing the MSA application (Line 6). The *greedy policy* borrows the selection method of the greedy set cover algorithm [Johnson 1973]. With this policy, the algorithm always selects the next location that spans a processing interest that covers the largest uncovered region of recent atomic events with respect to already selected processing interests in F. Another policy is one that favors processing interests that are close to the actually predicted location, assuming that these are the most likely future locations of focal points (*location policy*). A processing interest that is spanned by the selected location is then inserted to the result set F if it satisfies the following two conditions: (i) a processing interest is not suspected to be redundant, that is, another overlapping processing interest $f \in F$ is already selected that satisfies the consumers requirements on precision and recall (Line 7), and (ii) the processing on events from the selected processing interest should not lead to an expected computing resource usage above the system defined limit s_{max} (Line 8).

5.5. Placement and Migration

The placement and migration component [Ottenwälder et al. 2013] reduces network utilization by constantly adapting the placement of an operator depending on the location of a consumer. In this context, it considers two subgoals: (i) reducing the latency incurred by transferring state and not being operative during a migration from one MCEP server to another and (ii) reducing the communication costs incurred by migrations and event streams. The algorithm favors minimizing the latency and bandwidth incurred by the migration over optimized streaming costs. However, it can easily be extended to trade off both optimization criteria against each other.

The system opportunistically selects MCEP servers as future migration targets for future operator graphs and transfers state before actually switching to the future operator graph. In order to perform the opportunistic migration, we assign a dynamically updated live *migration plan (plan)* to each operator of the operator graph. The plan defines a set of future migration targets for an operator, a deadline by when an operator needs to be ready to execute at its migration target, and a time when the migration will be initiated such that the migration deadline can be met. At the time when a migration needs to be initiated for an operator according to the plan, the migration component will begin to copy the immutable state. In the meantime, the operator will continue to execute at its original placement until the transfer of state has been completed. Finally, when an operator graph switch occurs at a time that matches the deadline indicated in the plan, events are streamed to the new placement, and the resources taken up by an operator at its original location are released. Since operators are stateless in between operator graph switches, no other state has to be transferred for the migration.

Migration Plan and Time Graph. Figure 5(a) depicts for ω_{lane} of the accident example the plan which provides expected placements of an operator for a sequence of discrete time steps $t_i \in TS = \{t_0, \ldots, t_{max}\}$. An expected placement is a quadruple $ep = (H, B, t_i, t_m)$, where H is the expected hosting MCEP server of the operator, B the average expected bandwidth of the operator's outgoing stream, t_i is the time at which the transfer of immutable state has to be initiated, and t_m the deadline after which the operator starts processing at the host. Since consumers and sources are constantly associated with MCEP servers, we can use a similar abstraction to model their movements. The only difference for range queries with respect to spatial interests is that they can span several leaf MCEP servers, which requires extending ep by capturing H and B as a set of hosts and bandwidths.



(a) Basic migration plan: An operator will migrate from the MCEP server b_2 to b_1 at time t_1. Immutable state is already transferred at t_0.

(b) Time graph: Vertices represent migration targets. Edges represent migrations, weighted with *bdp* × *time spanned by edge*.

Fig. 5. Examples for data structures of the placement and migration component.

In order to build these plans, we use a data structure called a *time-graph* $= \{V_{tg}, D_{tg}\}$ (see Figure 5(b)) that allows for each individual operator ω to identify costs and durations of future migrations and placements. The time graph for a single $\omega \in \Omega$ comprises migration targets, which are suitable for fulfilling the constraints of a placement. Note that even not performing a migration imposes a cost by streaming events over the in- and outgoing streams.

Each vertex $v_{tg} \in V_{tg}$ represents the possible placement of ω at a MCEP server b, starting at time step $t_i \in TS$ and ending at the next time step $t_{i+1} \in TS$. For example, vertex v_1 in Figure 5(b) is labeled with b_1 at t_0, which represents the placement at MCEP server b_1 starting at t_0. Two special vertices which do not indicate future placements, labeled as s and f, represent the start and end of all possible sequences of migrations and placements.

Each directed edge $d_{tg} = (v_j, v_k) \in D_{tg}$ represents the migration between MCEP servers or no migration if d_{tg} connects the same MCEP server. Furthermore, an edge indicates that a migration starting at time step t_j of v_j is expected to be completed at time step t_k of v_k. For example, when the migration from b_1 to b_2 is started at time t_0, then all state is expected to be transferred by t_1 and ω can start processing.

The edge weight $w_{j,k}$ of $(v_j, v_k) \in D_{tg}$ represents the costs that are expected to occur in the time interval $[t_j, t_k)$—the average bandwidth-delay products weighted with that interval. This comprises the expected migration and streaming costs at the MCEP server of v_j during $[t_j, t_k)$. Since ω still processes events at the previous MCEP server, while the migration happens in the background, streaming costs are also considered if an edge represents a migration. The edge weight of 2KiBs between v_2 and v_4 comprises the streaming costs of 1KiBs at b_2 and migration costs of 1KiBs.

Migration Plan Creation. We now detail the basic algorithm for the creation of the plan for an individual operator ω (see generate_plan() in Figure 4), executed at the operator's current MCEP server. The algorithm finds a shortest weighted path in a time graph (*TG*) of ω and generates an executable plan (*stable_plan*) from this path. The time graph is maintained using the set of plans from all neighbors in the operator graph (*neighbor_plans*) and the anticipated event loads given by the expected placements in these plans.

The first plan is created after the deployment of an operator. Afterwards, the plan creation is triggered concurrently when the predictions on the outgoing bandwidth and

collected latencies deviate beyond a threshold from previous predictions (Lines 10–14), or the plan of the neighboring operators changes.

In the first step of the plan generation, the *time graph* is created, which models the expected placement and migration costs for this operator for the next *TS* time steps (Line 3). In a subsequent step, an initial plan is determined by traversing the shortest path in the time graph. This path contains a sequence of migration targets and times to migrate between them, where the expected overall network utilization is low (Line 4). We replace a previous stable plan with the newly found plan only if this new plan deviates in the expected placements at any time step (Line 6). However, before implementing the new plan, it is consolidated by coordinating it with the neighboring operators (Line 7). Neighbors adapt their own plan with the plan that is to be consolidated. Another iteration of the plan generation is triggered when the plan of at least one neighbor changed due to the plan that is to be consolidated. The consolidation phase is repeated until all neighbors agree on the plan of ω, that is, do not change their own plan.

6. EVALUATIONS

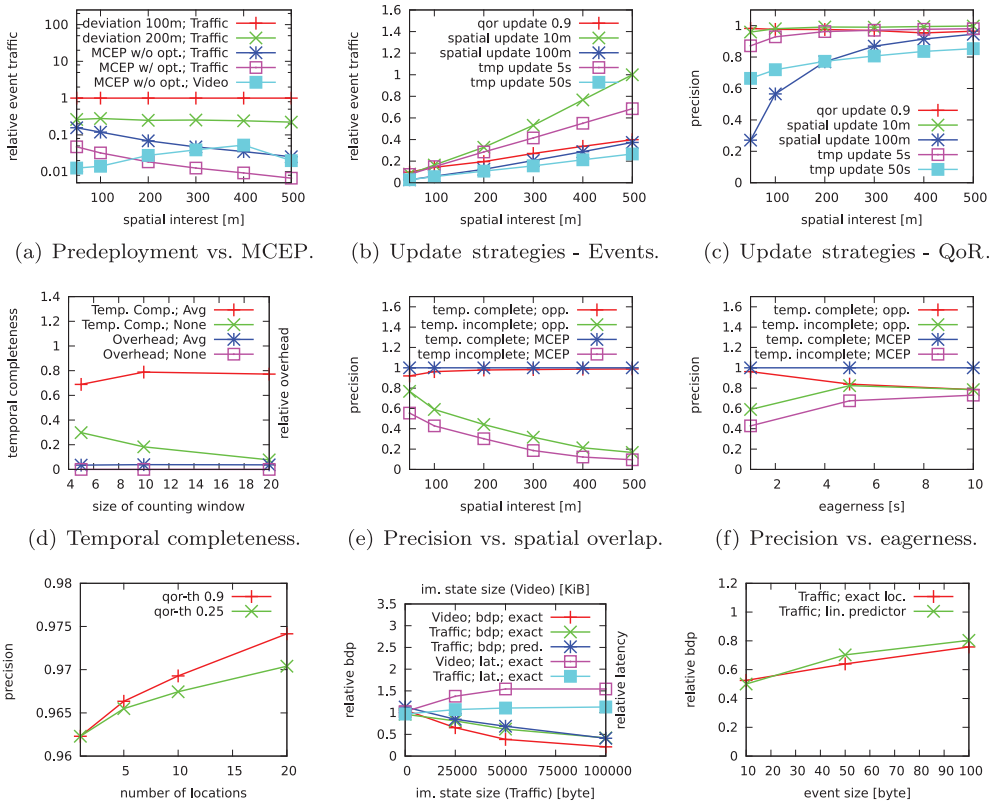
We evaluated the MCEP system using the Omnet++ network simulator [Varga and Hornig 2008]. To model realistic traffic patterns of vehicles on an OpenStreetMap graph [Haklay and Weber 2008], we used the traffic simulation package SUMO [Behrisch et al. 2011].

A dynamic set of brokers were connected in a hierarchical tree data-structure. Latencies between directly connected brokers in the hierarchy were similar. Each leaf broker was assigned a dedicated spatial region and, as a result, was connected to vehicles within this region. Over the course of 500 to 1,000 simulated seconds, approximately 2,500 to 5,000 vehicles drove in an area of the size 7.7km $\times$ 3.5km. We repeated the experiments approximately ten times with different sets of trajectories.

Our simulations cover three MSA applications. First, an *accident detection* application similar to the one detailed in Section 2, where the accident is detected when a sudden speed drop in an area is followed by a high number of lane changes. Each vehicle published, on average, one event per second in this scenario. Second, a *video friend finder* application (see Section 2), where the simulated cameras were equidistantly deployed in the service area and filmed the traffic in a square-sized area to find vehicles of friends. Third, a *speed detection* application similar to Figure 3(b), where the average speed was calculated over the most recent speed events that were above a predefined minimal speed. The spatial interest of consumers was described in all scenarios by rectangular shapes with varying edge lengths.

6.1. Evaluation of Query Reconfigurator

The first set of experiments shows the benefits in the resource usage of the MCEP approach (*MCEP without optimizations*) and the streaming optimizer (*MCEP with optimizations*) compared to the resource usage of an approach with a fixed amount of statically deployed overlapping processing interests (*pred.*). Those benefits were evaluated for the accident scenario (*Traffic*) and the video friend finder (*Video*). To increase the number of selected events per processing interest, we varied over several experiments the edge length of the spatial and processing interests. To allow consumers to match the spatial interest with the statically deployed processing interests with different accuracy, we also varied the spatial distance between centers of predeployed interest on a grid layout (*deviation*). Figure 6(a) depicts the number of streamed events (*event traffic*) relative to a fixed set of operator graphs with a deviation of 100m for different sizes of the spatial interest, varying from 50m to 500m. It shows that our approach of dynamically adapted processing interests, with and without the streaming



(a) Predeployment vs. MCEP. (b) Update strategies - Events. (c) Update strategies - QoR.

(d) Temporal completeness. (e) Precision vs. spatial overlap. (f) Precision vs. eagerness.

(g) Inc. num. of operator graphs. (h) Migration with fixed event size of 10 byte (10 KiB for Video). (i) Migration with fixed immut able state size 75,000 byte.

Fig. 6. Evaluation of MCEP System.

optimizations of Section 5.2, reduces the number of streamed events for a single *diq* in both scenarios. The savings of our approach are achieved, since only events of a subpart of the geographic space, selected by the spatial interest, have to be processed. Deploying more *diq* linearly increases the results.

We also evaluated the benefits of the update strategies presented in Section 5.2 in terms of the event traffic and their effect on the QoR for the accident example. In particular, we performed operator graph switches when a new reported location of a consumer deviates a fixed distance from the previous location, when more than a predefined time passed since the last operator graph switch, or when omitting the next operator graph switch is expected to drop the QoR, represented by the precision, below 0.9. The results depicted in Figure 6(b) show the relative event traffic with the maximal number of streamed events as baseline over several spatial interests, while Figure 6(c) shows the corresponding impact on the actual measured QoR. The results show that the update strategies significantly influence the number of streamed events, since performing switches to a new operator graph at a lower frequency requires the streaming of fewer historical events. However, low streaming costs due to a lower frequency of operator graph switches come at the price of a low QoR, since omitting many updates results in a high mismatch between the processing interest and the spatial interest.

We further analyzed the impact of several approaches for selecting the initialization point on the *temporal completeness* and the *overhead* of detecting unnecessary historical situational information when $\delta = 0$ and performing an operator graph switch every second. In this set of experiments, we used the speed detection scenario with a variable size of the corresponding counting window and a length of the relevance time span of 25s. In particular, we tested the basic approach, the window approach that uses an average over the last interarrival times of events (*Avg*), and a naive approach that did not stream any historical events (*None*). Figure 6(d) shows the fraction of detected events and overhead on the y-axis for the latter two approaches relative to the the basic approach versus the number of events selected by the counting window. Our average approach purges many unnecessary processing steps, which leads to a low overhead of less than 10% for any size of the counting window. However, this comes at the cost of a decreased average temporal completeness of around 80%. Not streaming historical atomic events leads to a temporal completeness of 20%.

6.2. Evaluation of Query Predictor

We analyzed the effect of the query predictor with respect to the selected speed events of the accident detection application. In particular, we evaluated its effect on precision (the results for recall are similar), completeness, and latency for the operator graph switch. To study these effects, we varied three control parameters. First, the *eagerness* that indicates how long before the actual operator graph switch occurs the query predictor deploys the operator graph(s). Second, a *number of random locations* around the predicted future location. Third, the edge length which represents the size of the *spatial interest*. Note that in realistic scenarios, processing latency for historical events depends on the number of events and complexity of operators. However, we used the processing latency as a control variable to measure its impact on the QoR, that is, 5ms in the presented results. The location prediction relied on a linear dead-reckoning predictor. We tested four deployment strategies. On the one hand, we deployed future operator graphs (*opp.*). On the other hand, we deployed operator graphs on-demand after the operator graph switch (*MCEP*). Both methods were tested either with the strict temporal completeness (*complete*) or the best effort completeness (*incomplete*).

The results in Figure 6(e) show the effects of different edge lengths for the spatial interest on the precision. We only maintained one future operator graph with an eagerness of 1s for this experiment. The temporal complete result for the on-demand deployment is always at 1, since the system waits for the deployment until the operator graph switch is performed and does not stop until all relevant events are processed, at the cost of late results (i.e., up to 10s). Opportunistically deploying future operator graphs that produce temporally complete results only incurs spatial uncertainties from the predicted location, thus the results are close to the optimum. Since more events are selected with larger spatial interests, more processing has to be done. This can lead to incomplete situational information, since not all processing steps can be finished until the next operator graph switch is performed.

The results in Figure 6(f) show the effects of varying the eagerness on the precision and the temporal completeness. With a higher eagerness, the incompleteness can be reduced, since the system can start processing earlier and waits longer until the next operator graph switch is performed. However, the spatial uncertainty of the predicted spatial interest for the opportunistically deployed operator graphs increases with the eagerness, since the location predictor has to provide consumer locations that are further in the future.

For the results in Figure 6(g), we increased the number of initially selected locations and applied the location selection algorithm depicted in Figure 4 with the location policy. For these experiments, we kept the eagerness at one second and the spatial

interest at 500m. It shows that using more predicted locations increases the precision, since the consumer can choose between more deployed operator graphs at the future location. Moreover, we tested several QoR thresholds (*qor-th*). To achieve a higher QoR, more operator graphs are maintained and more resources are invested, which results in a higher precision.

6.3. Evaluation of Placement and Migration Component

To evaluate the migration approach, we used the video friend finder (*Video*) and accident application (*Traffic*). Since our placement and migration approach is designed to consider the immutable state size and event size for an optimal sequence of migrations, we studied their influence on the the network utilization (*bdp*) and the streaming latency. We compared two migration approaches: (i) the *greedy* approach that greedily selected the broker with the best placement cost every few seconds, and (ii) our approach, called *MigCEP*. The operators were initially placed at the closest MCEP server to the consumer.

In the first experiment, we gradually increased the immutable *state size* of one leaf operator from 0 byte to 100,000 bytes for the accident scenario, and from 0KiB to 100,000KiB for the video scenario. Note that with the greedy approach in a realistic scenario with 100,000KiB state, the operator is not operational for over a minute during the migration. We tested the approaches with an exact knowledge about the future location of a consumer (*exact*) and a predicted location (*pred.*). The number of brokers were fixed in a tree with one broker as root and 40 brokers as children. The y-axis in Figure 6(h) depicts the results of the average bandwidth-delay product (*bdp*) for streaming and migration relative to the greedy approach. Moreover, it also depicts the streaming *latency* relative to the greedy approach. The x-axis depicts the immutable state size. While the resulting migration and streaming costs of the MigCEP approach are comparable to the greedy approach when the size of immutable state is low, the benefit in terms of bdp increases the larger the immutable state is, since our approach amortizes the migration costs. The average streaming latency, however, is lower for the greedy approach, since it rigorously places operators close to the sources.

For the second experiment, we varied the event size from 10 bytes to 100 bytes for the traffic scenario. The results, depicted in Figure 6(i), show the bdp relative to the greedy approach over the event size. Since higher streaming costs outweigh the immutable state size, the benefits of our approach decrease with larger event sizes.

7. CONCLUSION/FUTURE WORK

Recent investments in the industry—Google purchased the navigation software maker Waze for $1.03 billion [Federman and Rosenthal 2013]—confirm the growing importance of MSA applications. The mechanisms behind the MCEP system can improve the ability of those applications to detect and react to events in the presence of a very large number of mobile consumers and producers. Our findings have shown how dynamic operator reconfigurations can be be efficiently performed in order to leverage consistency for processing on spatiotemporal event streams in the context of three different use cases. For instance, in the context of a traffic scenario, resource savings of up to 99% are achieved. At the same time, we have shown how to minimize the resource usage by proposing methods for the opportunistic deployment and migration of operators. This way, additional resource savings of up to 40% are possible.

In future and ongoing work, we are investigating the potential for further optimization. For instance, processing events on mobile devices offers the potential to save resources whenever the detection of events requires access to sensors hosted on the mobile devices themselves. Moreover, overlapping operator graphs—as they occur in

densely populated spatial interests—offers the potential of saving redundant event processing. Therefore, we are currently working on energy-efficient operator placement as well as methods to reuse event streams in overlapping operator graphs.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers and R. Mayer for their helpful comments.

REFERENCES

Daniel J. Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag S. Maskey, Alexander Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stan Zdonik. 2005. The design of the borealis stream processing engine. In *Proceedings of the 2nd International Conference on Innovative Data Systems Research (CIDR'05)*. 277–289.

Asaf Adi and Opher Etzion. 2004. Amit - The situation manager. *VLDB J.* 13, 2 (2004), 177–203.

Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Rajeev Motwani, Itaru Nishizawa, Utkarsh Srivastava, Dilys Thomas, Rohit Varma, and Jennifer Widom. 2003. STREAM: The Stanford stream data manager. *IEEE Data Eng. Bull.* 26, 1 (2003), 19–26.

Ron Avnur and Joseph M. Hellerstein. 2000. Eddies: Continuously adaptive query processing. In *Proceedings of the ACM SIGMOD Interntional Conference on Management of Data (SIGMOD'00)*. 261–272.

Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. 2002. Models and issues in data stream systems. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'02)*. 1–16.

Roger S. Barga, Jonathan Goldstein, Mohamed Ali, and Mingsheng Hong. 2007. Consistent streaming through time: A vision for event stream processing. In *Proceedings of the 3rd Biennial Conference on Innovative Data Systems (CIDR'07)*. 363–374.

Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. 2011. SUMO - Simulation of urban mobility: An overview. In *Proceedings of the 3rd International Conference on Advances in System Simulation (SIMUL)*. 63–68.

Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog computing and its role in the internet of things. In *Proceedings of the 1st MCC Workshop on Mobile Cloud Computing (MCC'12)*. ACM, 13–16.

Sharma Chakravarthy and Deepak Mishra. 1994. Snoop: An expressive event specification language for active databases. *Data Knowl. Eng.* 14, 1 (1994), 1–26.

M. A. Cheema, L. Brankovic, Xuemin Lin, Wenjie Zhang, and Wei Wang. 2010. Multi-guarded safe zone: An effective technique to monitor moving circular range queries. In *Proceedings of the IEEE 26th International Conference on Data Engineering (ICDE'10)*. 189–200.

Gianpaolo Cugola and Alessandro Margara. 2012. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.* 44, 3, Article 15 (2012).

Gianpaolo Cugola and Alessandro Margara. 2013. Deployment strategies for distributed complex event processing. *Springer Comput.* 95, 2 (2013), 129–156.

Nihal Dindar, Peter M. Fischer, Merve Soner, and Nesime Tatbul. 2011. Efficiently correlating complex events over live and archived data streams. In *Proceedings of the 5th ACM International Conference on Distributed Event-Based System (DEBS'11)*. 243–254.

Cédric du Mouza, Witold Litwin, and Philippe Rigaux. 2007. SD-Rtree: A scalable distributed rtree. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE'07)*. IEEE, 296–305.

Josef Federman and Max J. Rosenthal. 2013. Waze sale signals new growth for Israeli high tech. USA Today, 12, 2013.

Bugra Gedik and Ling Liu. 2006. MobiEyes: A distributed location monitoring service using moving location queries. *IEEE Trans. mobile comput.* 5 (2006), 1384–1402.

Yu Gu, Ge Yu, Na Guo, and Yueguo Chen. 2009. Probabilistic moving range query over RFID spatio-temporal data streams. In *Proceedings of the 18th ACM International Conference on Information and Knowledge Management (CIKM'09)*. 1413–1416.

Mordechai Haklay and Patrick Weber. 2008. OpenStreetMap: User-generated street maps. *IEEE Perv. Comput.* 7, 4 (2008), 12–18.

Abdeltawab M. Hendawi and Mohamed F. Mokbel. 2012. Panda: A predictive spatio-temporal query processor. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems (SIGSPATIAL'12)*. ACM, 13–22.

Kirak Hong, David Lillethun, Umakishore Ramachandran, Beate Ottenwälder, and Boris Koldehofe. 2013a. Mobile fog: A programming model for large-scale applications on the internet of things. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Mobile Cloud Computing (MCC'13)*. 15–20.

Kirak Hong, David Lillethun, Umakishore Ramachandran, Beate Ottenwälder, and Boris Koldehofe. 2013b. Opportunistic spatio-temporal event processing for mobile situation awareness. In *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems (DEBS'13)*. 195–206.

Kirak Hong, Stephen Smaldoney, Junsuk Shin, David Lillethun, Liviu Iftodey, and Umakishore Ramachandran. 2011. Target container: A target-centric parallel programming abstraction for video-based surveillance. In *Proceedings of the 5th ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC'11)*. 1–8.

Mingsheng Hong, Mirek Riedewald, Christoph Koch, Johannes Gehrke, and Alan Demers. 2009. Rule-based Multi-query Optimization. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology (EDBT'09)*. ACM, 120–131.

Waldemar Hummer, Philipp Leitner, Benjamin Satzger, and Schahram Dustdar. 2011. Dynamic migration of processing elements for optimized query execution in event-based systems. In *Proceedings of the Confederated International Conference on the Move to Meaningful Internet Systems (OTM'11)*. Springer-Verlag, Berlin Heildelberg, 451–468.

David S. Johnson. 1973. Approximation algorithms for combinatorial problems. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing (STOC'73)*. 38–49.

Gerald G. Koch, Boris Koldehofe, and Kurt Rothermel. 2010. Cordies: Expressive event correlation in distributed systems. In *Proceedings of the 4th ACM International Conference on Distributed Event-Based Systems (DEBS'10)*. 26–37.

Boris Koldehofe, Ruben Mayer, Umakishore Ramachandran, Kurt Rothermel, and Marco Völz. 2013. Rollback-recovery without checkpoints in distributed event processing systems. In *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems (DEBS'13)*. 27–38.

Boris Koldehofe, Beate Ottenwälder, Kurt Rothermel, and Umakishore Ramachandran. 2012. Moving range queries in distributed complex event processing. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems (DEBS'12)*. 201–212.

Geetika T. Lakshmanan, Ying Li, and Rob Strom. 2008. Placement strategies for internet-scale data stream systems. *IEEE Internet Comp.* 12, 6 (2008), 50–60.

Alexander Leonhardi and Kurt Rothermel. 2001. A comparison of protocols for updating location information. *Cluster Comput.* 4, 4 (2001), 355–367.

David C. Luckham. 2001. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

Leonardo Neumeyer, Bruce Robbins, Anish Nair, and Anand Kesari. 2010. S4: Distributed stream computing platform. In *Proceedings of the IEEE International Conference on Data Mining Workshops (ICDMW'10)*. 170–177.

Beate Ottenwälder, Boris Koldehofe, Kurt Rothermel, and Umakishore Ramachandran. 2013. MigCEP: Operator migration for mobility driven distributed complex event processing. In *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems (DEBS'13)*. 183–194.

Peter Pietzuch, Brian Shand, and Jean Bacon. 2004. Composite event detection as a generic middleware extension. *IEEE Network* 18, 1 (2004), 44–55.

Zoe Sebepou and Kostas Magoutis. 2011. CEC: Continuous eventual checkpointing for data stream processing operators. In *Proceedings of the 41st International Conference on Dependable Systems Networks (DSN)*. 145–156.

Kyumars Sheykh Esmaili, Tahmineh Sanamrad, Peter M. Fischer, and Nesime Tatbul. 2011. Changing flights in mid-air: A model for safely modifying continuous queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'11)*. 613–624.

Jimeng Sun, D. Papadias, Yufei Tao, and Bin Liu. 2004. Querying about the past, the present, and the future in spatio-temporal databases. In *Proceedings of the 20th International Conference on Data Engineering (ICDE'04)*. 202–213.

András Varga and Rudolf Hornig. 2008. An overview of the OMNeT++ simulation environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (Simutools'08)*. ICST, 1–10.

Eugene Wu, Yanlei Diao, and Shariq Rizvi. 2006. High-performance complex event processing over streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'06)*. 407–418.

Xiaopeng Xiong, H. G. Elmongui, Xiaoyong Chai, and W. G. Aref. 2007. PLACE*: A distributed spatio-temporal data stream management system for moving objects. In *Proceedings of the International Conference on Mobile Data Management (MDM'07)*. 44–51.

Zhengdao Xu and Arno Jacobsen. 2007. Adaptive location constraint processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'07)*. 581–592.

Received October 2013; revised April 2014; accepted April 2014