Mobility-awareness in Complex Event Processing Systems

Von der Fakultät für Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Beate Ottenwälder

aus Esslingen

Hauptberichter:	Prof. Dr. rer. nat. Dr. h.c. Kurt Rothermel
Mitberichter:	Prof. Dr. Umakishore Ramachandran
	Prof. Dr.-Ing. habil. Bernhard Mitschang
Tag der mündlichen Prüfung:	13.04.2016

Institut für Parallele und Verteilte Systeme (IPVS)
der Universität Stuttgart

2016

Acknowledgments

First of all, I would like to thank my doctoral advisor, Prof. Kurt Rothermel, who has made this work possible. I am grateful for the guidance and support he provided. I also like to thank Prof. Umakishore Ramachandran for being my co-advisor and his ever-helpful counseling. Many thanks also to Prof. Bernhard Mitschang for being my second co-advisor. Special thanks go to my project supervisor Boris Koldehofe for his encouragement and many inspiring research discussions.

I would like to thank my colleagues from the Distributed Systems group at the University of Stuttgart for the good working environment. I also want to thank the members of the Embedded Pervasive Lab at the College of Computing of the Georgia Institute of Technology which kindly hosted me for five months. Their encouraging, constructive, and valuable feedback helped me a lot to improve my work. Especially, I want to mention Ruben Mayer, Marius Wernke, Sukanya Bhowmik, Frank Dürr, Kirak Hong, and David Lillethun.

Special thanks go to Annemarie Rösler. I would have been lost without her constant and reliable assistance in administrative matters.

I would like to thank the Baden-Württemberg Stiftung GmbH for partially funding this research through the "CEP in the Large" project. This funding, which enabled my research in the first place, allowed me to present my results to the international research community.

Last but not least, I want to express my sincere gratitude to my family and my friends. This work would not have been possible without their constant support, encouragement, and patience.

Beate Ottenwälder, April 23, 2016

Contents

Abstract

The proliferation and vast deployment of mobile devices and sensors over the last couple of years enables a huge number of Mobile Situation Awareness (MSA) applications. These applications need to react in near real-time to situations in the environment of mobile objects like vehicles, pedestrians, or cargo. To this end, Complex Event Processing (CEP) is becoming increasingly important as it allows to scalably detect situations "on-the-fly" by continously processing distributed sensor data streams. Furthermore, recent trends in communication networks promise high real-time conformance to CEP systems by processing sensor data streams on distributed computing resources at the edge of the network, where low network latencies can be achieved. Yet, supporting MSA applications with a CEP middleware that utilizes distributed computing resources proves to be challenging due to the dynamics of mobile devices and sensors. In particular, situations need to be efficiently, scalably, and consistently detected with respect to ever-changing sensors in the environment of a mobile object. Moreover, the computing resources that provide low latencies change with the access points of mobile devices and sensors.

The goal of this thesis is to provide concepts and algorithms to i) continuously detect situations that recently occurred close to a mobile object, ii) support bandwidth and computational efficient detections of such situations on distributed computing resources, and iii) support consistent, low latency, and high quality detections of such situations.

To this end, we introduce the distributed Mobile CEP (MCEP) system which automatically adapts the processing of sensor data streams according to a mobile object's location. MCEP provides an expressive, location-aware query model for situations that recently occurred at a location close to a mobile object. MCEP significantly reduces latency, bandwidth, and processing overhead by providing on-demand and opportunistic adaptation algorithms to dynamically assign event streams to queries of the MCEP system. Moreover, MCEP incorporates algorithms to adapt the deployment of MCEP queries in a network of computing resources. This way, MCEP supports latency-sensitive, large-scale deployments of MSA applications and ensures a low network utilization while mobile objects change their access points to the system. MCEP also provides methods to increase the scalability in terms of deployed MCEP queries by reusing event streams and computations for detecting common situations for several mobile objects.

Zusammenfassung

Heutzutage sind viele lokationsabhängige Daten für Anwendungen zugänglich. Milliarden an Nutzern fahren Autos oder tragen mobile Kommunikationsgeräte mit sich (bspw. Smartphones), welche ein breites Spektrum an Sensoren bereitstellen. Zusätzlich verbreiten sich weltweit weiträumig ausgelegete Sensornetzwerke, beispielswiese sind mehrere hundert Kameras in der Innenstadt von Atlanta aufgestellt. Die Möglichkeit auf Ereignisse zu reagieren ist für zukünftige mobile Situationsbewusstseinsanwendungen in vielen Anwendungsbereichen wie Verkehr, soziale Netzwerke und medizinische Überwachung unabdingbar. Eine Verkehrsanwendung kann beispielsweise Fahrer über nahegelegene, aktuelle Verkehrslagen informieren.

Die Verarbeitung komplexer Ereignisse (engl. Complex Event Processing (CEP)) ist ein Schlüsselparadigma um interessante Ereignisse (bspw. Unfälle) für mobile Situationsbewusstseinsanwendungen zu erkennen. Diese Ereignisse werden in Sensorströmen erkannt welche Änderungen in Sensormessungen beinhalten (bspw. Geschwindigkeiten von Fahrzeugen). Dazu wird eine Menge von Operatoren auf die Sensordatenströme angewandt. Jeder Operator nimmt Ströme an Ereignissen entgegen und verearbeitet diese um neue Ereignisse zu erkennen, welche dann in einem Ausgangsstrom von Ereignissen enthalten sind. Komplexere Ereignisse können erkannt werden indem Operatoren Ereignisströme anderer Operatoren verarbeiten. Der Ereignisfluss wird üblicherweise als Opratorgraph dargestellt, welcher alle Operatoren verbindet. Dieses einfache Modell hat eine Menge an Vorteilen bei der skalierbaren Verarbeitung von Ereignissen. Ereignisströme von Operatoren können wiederverwendet werden um mehrere Ereignisse von Interesse für Anwendungen zu erkennen. Außerdem können Operatoren des Operatorgraphen flexibel in einem Netzwerk von Rechenknoten ausgebracht werden um die Bandbreitenausnutzung von CEP Systemen oder andere Kriterien zu optimieren.

Überaschenderweise haben moderne CEP Systeme Schwächen, wenn es um die Unterstützung von hoch-dynamischen mobilen Konsumenten und Sensoren in mobilen Situationsbewusstseinsanwendungen geht. Um die Konsistenz zu bewahren sind die Sensorströme, welche als Eingabe des Operatorgraphen dienen, häufig fest zugeordnet. Eingangsströme eines Verkehrsoperators können beispielsweise mit Bezug auf ein Straßensegment ausgewählt sein. Das räumliche Interesse von mobilen Situationsbewusstseinsanwendungen ändert sich allerdings ständig mit der Lokation von mobilen Objekten, in deren Nähe Situationen geschehen. In solchen Anwendungen wird das Interesse meist über einen Brennpunkt ausgedrückt. Ein Konsument könnte so ein Brennpunkt sein und Interesse an allen Unfällen, die in den letzen 30 Minuten in einem 500 Meter Umkreis geschehen sind, haben. Für viele Situationen, die erkannt werden können, muss heutzutage hierzu die Änderungen der Lokation explizit von der Anwendung behandelt werden,

was typischerweise hohe Ineffizienzen zur Folge haben kann. Beispielsweise könnte eine der Folgen eine hohe redundante Verarbeitung von Ereignissen sein.

Das Ziel dieser Arbeit ist es Konzepte und Algorithmen für ein mobiles CEP System bereitzustellen—das MCEP System. Dazu wird ein Operatorgraph kontinuierlich und dynamisch an die Lokation eines Brennpunkts angepasst.

Als ersten Beitrag stellen wir hierzu ein Verarbeitungsmodell für Situationen, die mittels einer sich ständig ändernden Menge an Sensoren erkannt werden, vor. Dieser Beitrag beinhaltet ein allgemeingültiges, fenster-basiertes Operatormodell, welches effizient Anpassungen des Operatorgraphen an geänderte Lokationen eines Brennpunktes ermöglicht. Ein weiterer Teil dieses Beitrags ist eine Menge an Algorithmen, welche bandbreitendschonend und berechnungsschonend Anpassungen des Operatorgraphen bezüblich einer geänderten Lokation durchführen. Diese Algorithmen nutzen räumliche und zeitliche Überlappe des Interesses von Anwendungen aus.

Als zweiten Beitrag stellen wir einen Algorithmus vor, welcher es ermöglicht mit geringer Latenz auf historische Situationen zuzugreifen, sobald Situationen bezüglich einer geänderten Lokation benötigt werden. Dieser Algorithmus nutzt vorhergesagte, zukünftige Lokationen von Brennpunkten um relevante Situationen in der Nähe der vorhergesagten Lokation zu erkennen, bevor ein mobiles Objekt diese Lokation erreicht. Da zukünftige Lokationen großer Unsicherheiten unterworfen sind, werden hierzu parallel Situationen bezüglich mehrere zukünftige Lokationen berechnet.

Als dritter Beitrag wird ein Ressourceneffizienter Platzierungs- und Migrationsalgorithmus vorgestellt. Der Algorithmus bestimmt wann ein Operator auf welchem Rechenknoten in einem verteilten System ausgeführt wird. Dieser Algorithmus nutzt Wissen über die Operatoren aus und basiert auf Lokationsvorhersagen um Migrationen zu planen. Der Plan erlaubt es MCEP Migrationskosten zu amortisieren, indem Platzierungen gefunden werden, welche eine geringe erwartete Netzwerkauslastungen erreichen.

Als vierter Beitrag wird ein Mechanismus für die fein-granulare Wiederverwendung von Berechnungen und Strömen zwischen Operatoren vorgestellt um die Resourcenausnutzung des MCEP Systems zu verbessern. Dazu werden zwei Charakteristiken von Anwendungen ausgenutzt. Erstens müssen häufig ähnliche Situationen bezgülich mehrer mobiler Objekte, die nahe beiander sind, erkannt werden. Zweitens benötigen Applikationen häufig keine akkuraten Sensordaten um bedeutsame Ereignisse zu erkennen.

1 Introduction

At present, we face an explosive growth of location-specific data that becomes accessible to applications. Billions of users drive vehicles and carry mobile devices (i.e., smartphones and tablets) that are capable of monitoring their environment with a broad variety of on-board sensors [ZHBM07, CEL+08, PSA+13]. In addition, large-scale sensor deployments are spreading all around the globe, e.g., a vast amount of video data is produced by hundreds of cameras in the Atlanta metro area [Bla12]. The ability to monitor and react to events detected with the help of such data is the key to enable future *Mobile Situation Awareness applications (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.

Complex Event Processing (CEP)[1] systems [Luc01, BK09, AGKW14] offer tremendous support for large-scale MSA applications by detecting meaningful events (e.g., the occurrence of accidents) from low-level data streams (e.g., streams that capture changes in sensor measurements like vehicle speeds or camera frames). Consumers can register a continuous query with the system that describes the events of interest. In return, the CEP system will notify the consumers about any detected event that matches the consumer's query until the query is unregistered. In contrast to traditional *Relational Database Management Systems (RDBMSs)*, where data is typically pulled from a disk and processed as a result of issuing a query, changes in low-level data streams are pushed to the continuous query and processed "on-the-fly" [CCD+03]. This way, CEP systems can detect events of interest immediately when they occur. Since many existing applications, e.g., business processes [AKF+14], algorithmic trading [Hir12], or social networks [Twi14a], are event-driven, CEP systems have gained a wide-spread interest by industry [Ora15] and academia [AAB+05]. This interest has given rise to an abundance of optimizations [CM12a, HSS+14], e.g., bandwidth and latency efficient deployments of continuous CEP queries in a network of nodes [LLS08, Riz13].

Surprisingly, state of the art CEP systems have significant shortcomings in their support for highly dynamic mobile objects such as consumers and data sources of MSA applications. Typically, in order to ensure the consistency of produced event streams, the data streams selected as input to the processing of a continuous query are statically defined, e.g., incoming sensor streams are selected with respect to a fixed interest like a predefined road segment. However, the interests of MSA applications in situations

[1] Following the results in unified models [CM12b], *stream processing* [BBD+02] and *active data bases* [CM94] are used as synonyms to CEP in this thesis (its focus is on their similarities).

constantly changes with the locations of mobile objects. In MSA applications, the *spatio-temporal interest* in events is commonly defined with respect to the location of a *focal object*, e.g., a consumer is interested in all accidents that happened within the last 30 minutes in a 500 metre radius around its location. At present, changes in the location of focal objects need to be explicitly handled by the MSA applications, for many situations of interest. This can impose significant inefficiencies, such as a highly redundant processing and streaming of events.

The goal of this thesis is to provide concepts and algorithms for a *Mobile CEP (MCEP) system*, this is, a middleware that provides autonomous and dynamic adaptations of continuous CEP queries as mobile objects (consumers, data sources, and focal objects) change their location. In particular, we present methods that allow the system to dynamically change the data streams selected as input to the continuous MCEP query and ensure the consistency of produced event streams. Moreover, the system provides mechanisms that optimize and adapt the deployment of a MCEP query in order to reduce the detection latency of events and the bandwidth usage to stream events. Furthermore, we present methods to reuse partial detections of events to answer several MCEP queries in order to save system resources.

A more detailed view on the motivation for the work presented in this thesis as well as the background of mobile applications and CEP is given in Section 1.1. Section 1.2 will shed more light on the research focus and goals of the work. Finally, Section 1.3 presents the contributions and outlines the structure of this thesis.

1.1 Motivation and Background

This section covers the motivation and background for our MCEP system. Section 1.1.1 discusses the recent advances in mobile and utility computing that build the foundation for MCEP. In Section 1.1.2 we discuss a specific class of applications that can be supported by the MCEP system, MSA applications, and derive challenging requirements that stem from this class of applications. Based on these requirements, we assess in Section 1.1.3 to what degree existing middleware solutions can already support those applications and what challenges still need to be addressed by MCEP systems. We conclude in Section 1.1.4 by giving a brief overview over the CEP in the Large (CEPiL) project that tackles the unaddressed challenges and in whose context this thesis has been conducted.

1.1.1 Trends and Technologies

Large-scale MSA applications tremendously benefit from two major technological trends: First, the rise of connected mobile devices (e.g., smartphones, wearables, and tablets) as well as of sensors (e.g., GPS receivers and cameras); this allows applications to access sensor data that monitor mobile objects (e.g., people, vehicles, and shipments) virtually anywhere. Second, utility computing, e.g., cloud [AFG+10] and fog computing [BMZA12], eases the deployment of large-scale middleware solutions.

Connected Mobile Devices and Sensors

Nowadays, feature rich mobile applications, like navigation, can rely on powerful mobile devices with immense computing and sensing capabilities. For example, the state-of-the-art smartphone Sony Xperia Z4 has a 2 GHz Quad-Core processor and 3 GB of main memory. Moreover, it comprises a multitude of sensors: a microphone, a GPS receiver, a compass, a proximity sensor, an accelerometer, an ambient light sensor, and more [XP115]. Such mobile devices are widely deployed. Nearly a billion smartphones and over 250 million tablets were sold in 2013, which leads to over 1.9 billion smart devices in use [RvdM14, vdMR14]. Even a vast number of vehicles provide on-board computing capabilities. For example, nearly 13 million on-board navigation systems with GPS receivers were deployed in 2012 in North America, which is expected to rise to 56 million until 2019 [Eis12].

Another recent trend is that large-scale sensor deployments are spreading all around the globe. For example, hundreds of cameras are deployed in the Atlanta metro area [Bla12]. Hewlett-Packard (HP) aims with its Central Nervous System of the Earth (CeNSE) project to deploy a trillion sensors that observe the entire earth (e.g., to collect seismic data) [Pac09]. The Bluetooth low energy (BLE) standard raised the interest for the widespread deployment of cheap BLE devices, e.g., iBeacons [ibe15] that broadcast simple location dependent information.

Due to evolving and widely deployed wireless technologies (WLAN, LTE,... [Sch03]) and cheap flatrate billing options, mobile devices and sensors are always connected. This means, mobile devices and sensors are capable of communicating with any server in the Internet at nearly any time and from nearly any place over a nearby access point (e.g., a base station). Even the majority of vehicles is expected to be connected in the near future [Cis13].

Following the vision of the Internet of Everything (IoE) [Cis14], future mobile applications and middleware systems can easily access and integrate data from connected mobile devices and sensors—from virtually everywhere in the world. Within the IoE, virtually everything is connected, in particular applications, sensors (and other physical objects), data, and people.

The rise of connected mobile devices has drastically propelled the phenomenon of *Big Data* [Lan01]. This phenomenon refers to the vast amount and variety of data that is continuously produced or recorded by computer systems at a high velocity. Vehicles produce three to four gigabyte sensor data per minute [Kre15]. Every second, several thousand messages are produced by users of the microblogging service Twitter [Twi14b], while at the same frequency one hour of video is uploaded to the video-streaming platform YouTube [Smi14]; a dominating fraction of this data is already produced or accessed by mobile devices [Lun13].

Utility Computing

Cloud computing [Hay08, AFG+10] has emerged as an attractive solution for deploying large-scale middleware systems, like a CEP system, due to the simplified infrastructure maintainance for the middleware provider and the cloud's elasticity. In particular, administrators of a large-scale middleware can acquire on-demand computing or storage resources from an external cloud provider, rather than maintaining their own dedicated hardware. The external cloud providers are then responsible for many quality aspects like the availability of the leased resources. Moreover, cloud computing allows middleware systems to dynamically adapt the number of required computing resources to the current workload. For example, a traffic monitoring system that counts the number of cars on highways (i.e., that pass light barriers) can acquire more resources during the rush hour and release unused resources during the night when noone is driving on a road. Amazon's EC2 [Ama14] or Google's Compute Engine [Goo14] are prominent examples of cloud computing services.

Yet, cloud computing cannot support latency-sensitive MSA applications because clouds use data center resources that are geospatially centralized and typically far away from users in terms of network distance. Moreover, data is typically transferred over many hops and network resources like routers to cross the network distance between many users and clouds. To this end, the resources of the whole network are strained when large amounts of data are aggregated at cloud data centers.

Recent *fog computing* models, like the one proposed by Cisco Systems Inc. [BMZA12], the Edge Cloud [CHML14], or cloudlets [SBCD09] present the idea of using resources at the edge or in the middle of the network. This includes routers and dedicated computing resources. The resources are heterogeneous, geophysically distributed, and hierarchical since the resources are deployed from the edge to the core network. In contrast to the cloud, fog computing allows applications that require low network latencies to perform processing on computing resources near the edge of the network. Latency-tolerant, large-scope processing can still be efficiently performed on powerful resources at the core of the network, such as cloud data centers.

1.1.2 Mobile Applications and Requirements

One class of applications that tremendously benefits from highly accessible location-specific data is *Mobile Situation Awareness (MSA)*. These applications [Hon14, RHI+12, SA11, MPVW05] are real-time applications that deal with timely decision making based on knowledge that is extracted from sensed data. Note that we refer to soft real-time applications where it is undesirable, but not critical to miss deadlines. A vast amount of applications from various domains, such as traffic monitoring, social media, advertising, entertainment, and health care, can be classified as MSA application. The rest of this section will detail concrete *traffic and transportation*, *monitoring and tracking*, and *smart environment* scenarios from those domains, in order to assess the application characteristics and their requirements.

Scenarios for MSAs.

- **Traffic and Transportation:** A traffic application can notify drivers of nearby live road conditions, warning of traffic, accidents, or obstructions on the road. As a specific example, consider a user who is driving from San Francisco to Los Angeles. The user may have a vehicular application to automatically detect driving conditions along the route and reroute the user around those problems. Recent investments in the industry—Google purchased the navigation software provider Waze for $1.03 billion [FR13]—confirm the growing importance of such traffic applications. Waze allows users to share real-time traffic and road information, e.g., a map can show alerts about police, which are reported by other users. A transportation application can inform train drivers about vehicles or people that recently moved onto the rail-road ahead of the train's location and not left the rail-road yet.
- **Monitoring and Tracking:** Another example for situation awareness applications is a smart surveillance application [HSS+11] that can support police officers by displaying video streams on their smartphones showing suspicious people who were recently nearby. Social platforms can be enriched by incorporating automated live-updates about activities of friends, e.g., by streaming a video of a friend when she participates in a public sports event or when she was recently nearby and entered a specific location. Health monitoring applications [SXSS14] 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., sensors monitoring weather changes that affect a patient with severe allergies).
- **Smart Environments:** A smart super market can change or highlight the content of shelves based on the needs of a nearby customer, e.g., a shelf can highlight a product that has been pointed at by a sales clerk while the customer was looking in the other direction. A smart bus can change the language of its signs or attached advertisements depending on the language skills of people walking towards the bus.

Characteristics. A multitude of the aforementioned applications has considerably similar characteristics. 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 data and continuously push *situational information* like the occurrence of accidents to the application. Consumers typically have a *spatial interest* in situational information, this means that the situational information stems from sensor data in a confined spatial region. Sensors and consumers of situational information are predominantly highly dynamic, mobile devices.

For consumers, plain sensor data is often not expressive enough to represent situational information. *Events* detected in the data are often of higher value and help to make informed decisions. For example, the locations of all vehicles in the proximity of the user

of a traffic monitoring application are not relevant to her. Yet, if a particular movement pattern in the trajectories that are spanned by these locations indicate an accident on the road segment ahead of her current location, she can react by selecting another route.

The spatial interest in sensor data and situational information is typically dynamic and changes with the location of a *focal object*. For example, a vehicular navigation system can display accidents that occurred in San Francisco when starting a trip from San Francisco to Los Angeles and update its display with accidents that occurred close to the location of the vehicle during the trip. Users, like the driver of the example's vehicle, are often interested in *spatially consistent* and *spatially complete* situational information. Otherwise, some accidents are not reported (false negatives) or accidents that are not of interest are reported (false positives) in the example. Situational information is spatially consistent iff it is detected using only sensor data that matches a distinct spatial interest, e.g., accidents that occurred in San Francisco are not reported after arriving in Los Angeles. Situational information is spatially complete iff all sensor data with respect to a spatial interest is considered for the detection of an event, e.g., a sensor data that would indicate an accident in the vehicle's proximity is not missed.[2]

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 consumers even have a *temporal interest* in historical situational information, e.g., a recent accident is still relevant to the consumer in the future when it is still blocking the road. In particular, consumers are interested in *temporally consistent* and *temporally complete* situational information, to avoid false positives and negatives that would be detected otherwise. Situational information is temporally consistent iff for a sequence of temporally ordered sensor data a distinct situational information is detected. For example, the same accidents are detected in recorded sensor data streams after the user of a navigation system arrived at her destination and live while driving. Moreover, it is temporally complete [BGAH07] iff all situational information that can be detected within a consumer-defined historical time span is delivered to the consumer, i.e., all accidents that occurred in the last 10 min are reported.

Another important characteristic is that for many MSA applications approximate results are often sufficient [BCD03]. For example, if accidents are reported to a consumer that are detected in sensor data streams with respect to a spatial interest that overlaps in large parts with the consumer's spatial interest, the consumer is still informed about the closest and most relevant accidents.

Requirements. Based on the previous observations regarding mobile devices, sensors, utility computing, and MSA applications it is possible to generalize the following requirements for systems that support MSA applications—in particular MCEP systems. Those systems should ensure the following:

[2] Note, that we will formally define consistency and completeness in context of MSA applications in Chapter 2.

- They should continuously detect *arbitrary situational information* on arbitrary sensor data. A vision domain expert should be able to detect faces on video streams while a vehicular domain expert should be able to detect traffic situations.
- They should continuously *detect application-specific situational information* on sensor data selected from spatial regions relative to the location of a *focal object*, e.g., a mobile consumer wants to query for recent and nearby accidents. Moreover, it has to *adapt the spatial interest* and the detection of situational information according to the location of the focal object.
- They should ensure an application-specific *delivery semantic*. This requires a notion of *completeness* and *consistency* of situational information. For example, when detecting a downward trend in speed sensor data within the last 15 min in a 500 m 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 a previous and the current spatial interest.
- 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 can 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.
- 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.

1.1.3 Continuous Query Processing

Two types of continuous queries that have been extensively studied in literature already offer to support MSA applications. First, *Event Processing Queries* detect situational information in selected data, yet are not flexible enough to cope with the dynamics of mobile consumers and sensors. Second, *Moving Range Queries* allow expressing a dynamic spatial interest in data, yet do not provide the means to detect situational information in the selected data. We briefly discuss those continuous queries and then discuss their combined support for MSA applications.

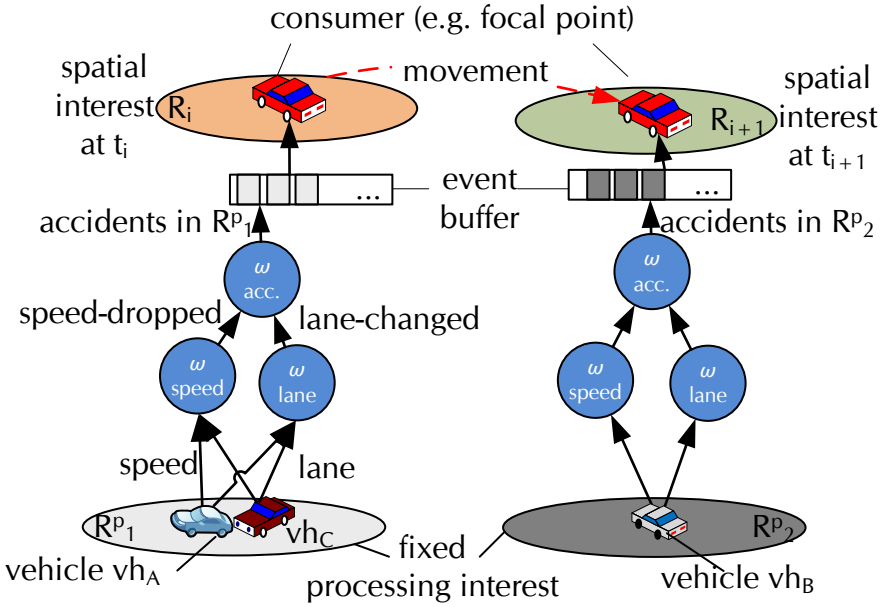


Figure 1.1: Accident detection example: Each vehicle (vh_A,vh_B,vh_C) is a source and continuously reports its speed and location, including its identifier, to the operator graph(s) that process events with respect to the vehicle's current location. By processing those events, two leaf operators detect 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.

Event Processing Queries

Complex Event Processing (CEP) [Luc01, BK09, CM12b] is nowadays a well established paradigm to continuously detect events of interest from basic data streams. CEP 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*.

The detection is performed by a set of operators [PSB04, WDR06, CEB+09, NRNK10, ZDI14]. Operators are typically considered as *black boxes* that process a continuously changing *selection* of input events. Operators detect zero or more new events by applying an arbitrary correlation function on the selected events and produce event streams as output that include all detected events. For example, if a source is providing a stream of location changes, then an operator can select the changed locations that happened within the last 10 min as input and detect if the trajectory of the source indicates a movement towards a landmark [TSBV05]. This operator produces an output stream with information about landmarks towards which the source is moving. To consistently indicate when a complex event occurs, it is stamped based on the times of the occurrences of the corresponding atomic events, e.g., using a temporal interval or the maximal timestamp [AC05]. The event flow is typically described by connecting operators to each

other in an *operator graph* [ACc+03, AAB+05, GAW+08, KKR10] that connects individual operators (nodes in Figure 1.1), sources, and consumers (vehicles in Figure 1.1) through event streams (edges in Figure 1.1).

Moving Range Queries

Moving range queries (MRQs) [XECA07, GYGC09, ZJDR10, ARIC12] return mobile objects like sensors in a consumer-specified range as a result and update this result depending on the location of a moving focal object. For example, a moving consumer is always updated with the nearest taxis in a 500 m radius. These queries can provide filtered and aggregated data [XECA07, GL06, SPTL04] to support MSA applications, e.g., with a count on the number of taxis in the range. In a nutshell, MRQs are highly adaptive to the location of a focal object, however, they lack the expressiveness to detect arbitrary situational information on the data from selected sensors.

Basic Support for MSA

In the remainder of this section we discuss how MSA applications can be supported by continuous queries. First, a *static range query* approach that predeploys a fixed number of operator graphs, each processing atomic events from a dedicated spatial region denoted *processing interest*. Second, a *moving range query* approach, which defines a moving range query as processing interest to an individual operator graph; sources selected by the moving range query push streams to the operator graph.

Predeployment of Static Range Queries. Sources, operator graphs, and consumers are often statically coupled. Therefore, a set of operator graphs has to be deployed that can only process events of sources that are selected by a static range query which covers a predefined spatial region, the so-called processing interest. This way, sources are decoupled from operator graphs as shown in Figure 1.1. The consumer, who also poses the focal object in the example, can fetch historical and live situational information from a buffer. This buffer comprises detected situational information from an operator graph whose processing interest bears the highest overlap with the spatial interest of the consumer.

In such predeployment scenarios, the quality of situational information is unpredictably degraded, since the consumer's spatial interest is highly unlikely to match the processing interests of the operator graphs. Moreover, two operator graphs process and stream events in this example although only one query is deployed, which is inefficient in terms of communication and computation. In general, depending on the required quality, the predeployment of operator graphs can cause a significant amount of unnecessary processing where none of the produced events are of interest to a consumer. For example, consider a traffic monitoring system which guarantees, for each processing interest, an accuracy of 100 m maximum deviation from the actual spatial interest whereas each processing interest covers a region of size 1 km^2. Then approximately 23,100,000 overlapping processing

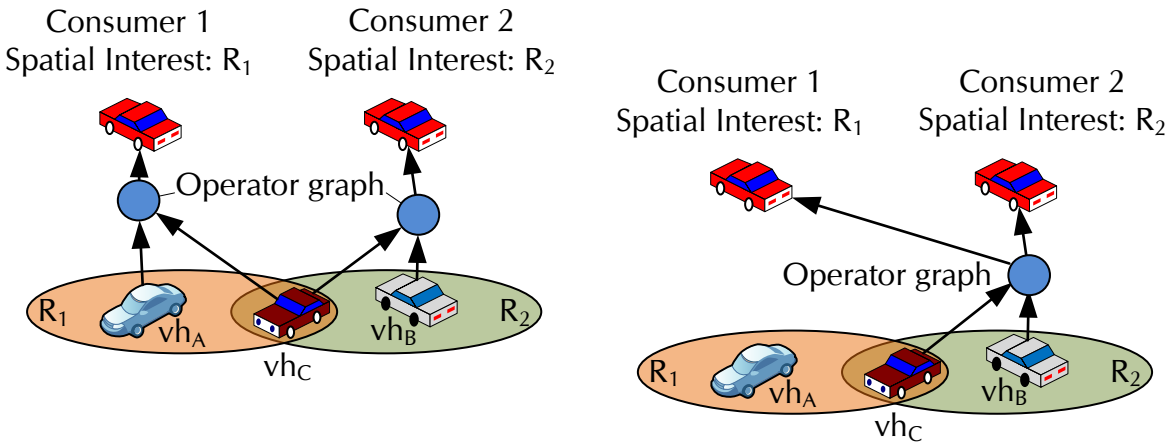
interests (as well as corresponding operator graphs and buffer) have to be predeployed to cover the core road network of Germany. However, due to the large variations in traffic—from 5,000 upto 180,000 cars per day for the main road segments—we would expect 40% of all buffers for processing interests to be idle and not answer queries 95% of their deployment time.[3]

Moving Range Query Approach. As a second approach, MSA applications may choose to deploy a single operator graph and couple it with a moving range query (MRQ) which continuously covers the application's spatial interest. The MSA application then informs sensors that are currently selected by a moving range query to provide their data streams as input to the operator graph, while not selected sensors are informed by the MSA application to stop streaming. As a result, operator graphs are only processing and streaming events when they match the spatial interest, this way avoiding the inefficiencies towards the scalable detection of events of predeployed static range queries. However, although operator graphs and MRQs have been individually well researched, there are still drawbacks when coupling these techniques. In particular it can lead to spatial inconsistencies, temporal incompleteness, high bandwidth and computational costs, and a high latency.

Consistency and Completeness: CEP systems focus on temporal completeness and temporal consistency, which can be addressed by making the operator graphs robust to failures [SM11, KMR+13]. Moreover, it can be addressed by ensuring event orderings, i.e., blocking operators until all events of a selection have arrived and revising events [BGAH07, MP13]. In contrast, MRQs provide spatially complete and consistent results, i.e., by re-evaluating the range query with every clock-tick or using triggers like safe zones [TS04, CBL+11]. However, when coupling these techniques, neither spatial consistency, nor temporal completeness is ensured.

Note that operators are stateful [AE04], i.e., they keep a partial result of a correlation step [PSB04, ABB+03] (*processing state*) or events from their incoming streams (*mutable state*). Consider now an operator detecting a critical number of vehicles that reduced their speed within the last 10 minutes in the spatial region that matches the spatial interest of a consumer. This probably indicates an upcoming traffic jam. Unfortunately, the operator is oblivious of changes in the spatial interest, say from a spatial interest R_i to a spatial interest R_{i+1}, while the MRQ covering the spatial interest is informed about such changes. As a result, the operator keeps both the state based on sensor data with respect to R_i and with respect to R_{i+1}. This may lead to a spatially inconsistent situational information, e.g., when the operator reports two vehicles, say vh_A and vh_B, slow down, although vehicle vh_A's location is only comprised in R_i and vehicle vh_B's location is only comprised in R_{i+1}. Moreover, consider that the focal object is, from time t_2 on, interested in events based on sensor data in R_{i+1}. Since the MRQ that covers the spatial interest selects live sensor

[3] The traffic rates were obtained from a recent survey on the German traffic density [Ver12]. We assumed an average speed of 25 m/s and that for 40% of all road segments 10,000 vehicles are passing per day and a large number of 10% of the traffic participants were interested in traffic events.



(a) Two consumer deploy individual operator graphs (b) Two consumers receive events from one merged operator graph

Figure 1.2: Since Consumer 1 and Consumer 2 deploy the same operator graph to detect traffic (see Figure 1.2), CEP mechanisms can automatically merge the deployed operator graphs.

data—sensor data that was captured after t_2—the operator graph will not consider the speed of a vehicle, say vh_C, from a time $t_1 < t_2$. As a result, it may not detect if vehicle vh_C slowed down within the last 10 min. Hence, the operator graph reports spatially incomplete situational information.

Computational Costs: Event streams of operators can be reused in CEP systems to avoid redundant computations when detecting events of interest for various consumers by merging common sub-graphs of deployed operator graphs [CDTW00, HRK+09]. This can lead to a highly degraded quality of situational information in MSA scenarios. For example, since both Consumer 1 and Consumer 2 in the example depicted in Figure 1.2a are both interested in the detection upcoming traffic jams, one merged operator graph can be deployed. To avoid inconsistent results this operator graph processes events with respect to either the spatial interest of Consumer 1 or the spatial interest of Consumer 2 as depicted in Figure 1.2b. However, due to the dynamics of focal objects, both spatial interests only overlap which results in false positives and false negatives. For instance, events are delivered to Consumer 1 that depend on sensor data from vehicle vh_B although vehicle vh_B's location is not comprised in the spatial interest of Consumer 1.

End-to-end Latency: Improvements in communication costs and end-to-end latencies are achieved in CEP systems by placing operators on computing resources close to sources and destinations of event streams connected to the operator. Placement algorithms [LLS08, HLSD11, CSM11, CM13] automate the task of placing operators. These algorithms find and implement an assignment of operators to computing resources which optimizes a specific objective function, e.g., a minimal overall bandwidth-delay product over all streams in the operator graph. However, most of the placement algorithms are tailored for scenarios with infrequent changes, i.e., where sensors and consumers are not mobile.

Therefore, placement algorithms are either neglecting the impact of migration costs on the placement strategy or an amortization of these costs. 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, e.g., 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 requires to first stop 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.

1.1.4 CEP in the Large

The goal of the CEPiL research project was to enhance CEP systems by providing methods for highly scalable, reliable, and secure event detection in dynamic and mobile environments, e.g., in order to support applications like MSA applications. The international project CEPiL has been funded by the research program "Internationale Spitzenforschung II" of the *Baden-Württemberg Stiftung gGmbH* and involved researchers from the University of Stuttgart (Universität Stuttgart) and the Georgia Institute of Technology (Georgia Tech).

This thesis has been partially carried out within the *CEP in the Large (CEPiL)* research project. The particular focus of this work was on achieving the project's scalability goals as well as to seamlessly integrate mobile devices into CEP systems. However, before substantiating the concrete goals of the thesis in Section 1.2, the contributions made to achieve supplemental scalability, security, and reliability goals are briefly summarized in the remainder of this section.

Scalability. Many distributed systems are highly heterogeneous in their capabilities; in particular, when they involve mobile devices, cloud, and fog resources. Placing operators on these resources is subject to many constraints, e.g., latency constraints. CEPiL provides placement algorithms for operators in a heterogeneous, heavy-constrained environment [SKR11].

Reliablity. Due to hardware or software failures (e.g., induced by power blackouts or bugs) operators might crash in distributed CEP systems. Even if the operator eventually recovers, this might lead to event loss since other operators or sources still detect and stream events. Recovered operators might also detect false-positive events, if the detection relies on the internal state of the operator that was kept in main memory and thus is lost during the crash. CEPiL provides mechanisms to ensure the reliable execution of operators while ensuring a low recovery latency and a low streaming overhead [VKR11, KMR+13].

Security. Coupling several domains into a large-scale distributed system induces the threat of information abuse. In particular, event sources have no control over the usage of their events in traditional CEP systems. A major contribution of CEPiL is a mechanism that allows users to specify security rules that restrict the usage of events and enforce them throughout the whole operator graph [SKRR13].

1.2 Research Focus and Goals

This thesis focuses on providing concepts and algorithms that support the efficient and scalable execution of MSA applications with a MCEP system. To this end, the following four goals, which are crucial to meet the requirements outlined in Section 1.1.2, need to be addressed:

Expressive Query Model and Scalable Architecture for Mobile CEP

The first major goal is to provide an expressive MCEP query to consumers, that allows them to specify their dynamic, spatio-temporal interest in situational information with respect to a focal object. This requires clear spatio-temporal delivery semantics that clarify which events are of interest to a consumer in terms of historical and live events. In particular, it requires concepts to formally define the quality, completeness, consistentcy, and delivery order of situational information. The MCEP query needs to deliver situational information and access sensor data at low latency while incurring low network utilization in large-scale MSA scenarios. This requires concepts for a highly scalable, distributed system architecture that can include fog and cloud resources.

Efficient Processing of Mobile CEP Queries

The second major goal is to provide methods to efficiently process MCEP queries while ensuring the consumer-defined delivery semantics. Since it is inefficient and not scalable to predeploy large numbers of fixed operator graphs, this requires concepts to deploy individual operator graphs with individual range queries. To avoid inconsistent operator states, concepts are needed for an efficient coordinated reconfiguration of operator graphs and range queries depending on the focal objects location. This requires to decide at which time and for which location to perform the reconfiguration in order to preserve a desired quality. Moreover, individual operator graphs require concepts to efficiently detect historical situational information for completeness.

Resource Conserving Placement and Migration of Mobile CEP Queries

The third major goal is to efficiently deploy MCEP queries within a large-scale network of computing nodes. This requires not only to find a placement for the operator graphs but also to efficiently adapt their placement through migrations depending on the access points of mobile sensors and consumers. From a consumers point of view this means to

continuously adapt the deployment to one that minimizes the end-to-end latency from the sources to the consumer of an operator graph in order to deliver timely situational information. The provider of a MCEP middleware, is highly interested in using adaptation mechanisms that minimize the network-utilization, to increase the scalability and reduce the costs of maintaining the infrasturcutre (e.g., monetary costs when leasing resources from cloud or fog providers).

Avoidance of Redundant Mobile CEP Query Processing

The fourth major goal is to achieve scalability, in particular to support a large number of consumers with timely situational information. Processing each MCEP query isolated from other queries imposes a significant resource requirement on the system. Therefore, a primary concern for the development of MCEP systems is to increase the resource-efficiency when supporting a large number of consumers. This requires a method that allows to find and avoid redundancies when processing individual operator graphs for each MCEP query.

1.3 Contributions and Structure

1.3.1 Contributions

This thesis combines and extends the findings and contributions of the work presented in [KORR12], [OKRR13], [HLR+13b], [OKR+14a], [OKR+14b], and [OMK14], towards a large-scale *Mobile CEP system*. Note that the previously mentioned work is published in cooperation, e.g., with Kirak Hong [Hon14] or David Lillethun [Lil15]. Therefore, the author of this thesis contributed only a percentage of the individual contributions. In addition, the contributions presented in this thesis profit from the inside gained from large-scale video streaming applications [HOR14] and the programmability of on-demand resources at the edge of the network [HLR+13a]—the author of this thesis claims 20% of each of these contributions.

In particular, the contributions of this thesis are:

1. A novel query model for situational information that depends on an ever-changing set of sources that were recently close to a focal object. This includes a general, expressive operator model based on window semantics. Part of this contribution is a mobility-driven reconfiguration algorithm that uses the dependency of windows over several levels in the operator graph to ensure temporal completeness. To ensure spatial consistency and event orderings, we utilize the concept of inducing markers into event streams to isolate the processing of events for each individual processing interest. This contribution is primarily based on work that has previously been published in [KORR12]; the author of this thesis contributed about 45% of the paper's scientific content. It is supported by work published in [OKR+14a] and [HLR+13a].

2. Methods to minimize the bandwidth and computational requirements for processing MCEP queries. These methods utilize the spatio-temporal overlap between event streams for consecutive locations of the focal object to avoid unnecessary processing and streaming of events. Moreover, we propose methods to gracefully degrade the quality and reduce the completeness of situational information in order to reduce the overhead of reconfigurations. This contribution is primarily based on work that has previously been published in [OKR+14a]; the author of this thesis contributed about 70% of the paper's scientific content. It is supported by work that has been published in [KORR12].

3. Algorithms for a low-latency delivery of historical atomic situational information. These algorithms predict future locations of a focal object and process situational information with respect to the corresponding spatial interest before the focal object arrives at that location. Furthermore, these algorithms utilize parallel resources to (i) further decrease the latency by pipelining the processing for predicted future locations in several discrete time-steps and (ii) increase the quality of the produced result by opportunistically processing several the results for several future locations. This contribution is primarily based on work that has previously been published in [HLR+13b]; the author of this thesis contributed about 45% of the paper's scientific content. It is supported by work that has been published in [OKR+14a].

4. Methods that support the resource-efficient placement and migration in MCEP systems. These methods exploit application knowledge of the MCEP system and predicted mobility patterns to plan the migration ahead of time. This plan allows the MCEP system to amortize the migration costs by selecting migration targets that ensure a low expected network utilization for a sufficiently long time. Moreover, these methods serialize the operator for the migration—which allows the system to migrate parts of the operator a priori—in a way where unnecessary events are not migrated and bandwidth is reduced. This contribution is primarily based on work that has previously been published in [OKRR13]; the author of this thesis contributed about 70% of the paper's scientific content. It is supported by work that has been published in [OKR+14a] and [OMK14]; the author contributed about 90% of the paper's content.

5. System mechanisms for fine-grained reuse of computations and streams between operators to support scalable processing of consumer queries. These mechanisms exploit two inherent characteristics of MSA applications. The first characteristic is that many consumers have overlapping spatio-temporal interests because consumers are typically interested in the same or similar situational information of their surrounding areas. Another important characteristic is that most MSA application do not require to process a perfectly accurate set of atomic events to generate meaningful situational information. By reusing approximate processing results based on slightly mismatching input streams, our system dramatically increases system scalability in terms of consumer queries and input streams. This contribution

is based on work that has previously been published in [OKR+14b]; the author of this thesis contributed about 70% of the paper's content.

1.3.2 Structure

The rest of this thesis is structured as follows: Chapter 2 details the basic MCEP query model and the system architecture for the MCEP system. Chapter 3 discusses methods to process individual MCEP queries and optimizations to save bandwidth and computational costs as well as to ensure a low latency delivery of historical situational information. Chapter 4 details our mechanisms for the latency-efficient adaptation of the deployment of MCEP queries with a low network-utilization. Chapter 5 discusses methods for the scalable and resource-efficient processing of multiple queries. We conclude in Chapter 6 with a short summary and outlook.

2 System Architecture

This section details the MCEP system architecture. In Section 2.1, the system model is discussed, including a formal definition of events, the operator graph, and the distributed infrastructure model. In Section 2.2, the basic mobile CEP Query (MCEP Query) is specified as well as the relevant mobility-aware delivery semantics. Section 2.3 details the software components of our distributed MCEP middleware and their interactions.

2.1 System Model

2.1.1 Event And Operator Graph Model

Event Model. Events describe in a computer-processable way state changes in sensor measurements or systems, such as location changes in a stream of GPS locations or newly detected accidents. Formally, an event is a tuple of attribute-value pairs, i.e., an event e is of the form $e = \{(\alpha_1, \phi_1), \ldots, (\alpha_{\max^\alpha}, \phi_{\max^\alpha})\}$. *Types* of events $\mathbb{E}_\mathbb{T}$ allow for coarse-grained filtering of events,[1] by specifying a tuple of attributes, e.g., events that carry a location attribute and timestamp are all of the same type "location event". An abbreviated formalism for events of the same type $type \in \mathbb{E}_\mathbb{T}$ is therefore $e = \{type, \phi_1, \ldots, \phi_{\max^\alpha}\}$. In order to be precise, this thesis refers to events that are directly emitted from sensors or other data sources as *atomic events*. Events that are detected by a CEP system and delivered to a consumer are denoted *situational information*.

Each event e has an attached *timestamp* ($t(e)$), determining when it occurred in order to reason about temporal correlations. Moreover, each event has a *location* ($l(e)$), determining where it occurred, which allows us to perform spatial selections (e.g., the location identifies atomic events or situational information that occurred close to a focal object). To capture their inherent uncertainty, e.g., introduced by GPS sensors, timestamps and locations are often described by temporal intervals or spatial distributions in the literature [All83, SGM09, LWG+09, MBESG10]. For brevity, but without restricting the generality of the approaches presented in this thesis, timestamps and locations of atomic events are represented by discrete points in time and space.

Operator Graph Model. The detection of situational information is modeled by a directed, acyclic operator graph, $G = (\Omega \cup D \cup U, L)$. Ω denotes the set of operators, D the set of sources, U the set of consumers, and $L \subseteq (\Omega \cup D \times \Omega \cup U)$ the event streams. Each stream

[1] Note that the fine-grained filtering of publish/subscribe [TKK+11] is out of the scope of this thesis.

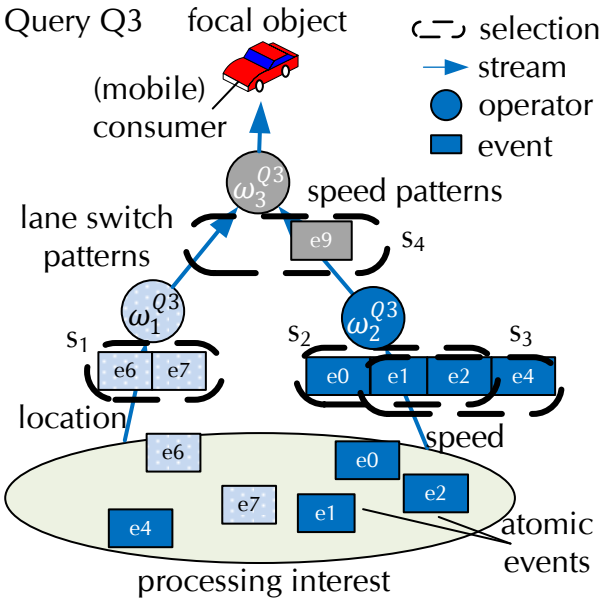


Figure 2.1: Basic CEP Operator Graph of a sample query $Q3$, which processes events from sources in a processing interest.

$(src, sink) \in L$ is composed of events of the same type and is produced at $src \in \Omega \cup D$ using arbitrary restrictions on the attribute values of its events. In Figure 2.1, three operators ω_1, ω_2, and ω_3 are connected by streams to an operator graph. The operator graph in the example receives *atomic event streams* from all sensors in a specific spatial area, denoted *processing interest*, as input, which are step-wise processed by operators, before situational information is delivered to consumers.

The execution of each operator is characterized by performing a sequence of correlation steps. In each correlation step, the operator takes as input a *selection* s of events from its incoming event streams and applies its operator specific correlation function f_ω—which implements the operator logic—to produce a set of outgoing events. The selection is updated for each correlation step by means of a *selection policy*, resulting in a deterministic sequence of selections $S(E)$ for a distinct sequence of incoming events E. For example, ω_2 in Figure 2.1 has specified a selection policy in form of a sliding window to select three subsequent speed events $\{e_0, e_1, e_2\}$ on its incoming stream. In the next correlation step the operator ω_2 will apply its correlation function to the updated selection $\{e_1, e_2, e_4\}$. Formally, let $\mathbb{S}$ be the universe of all selections and $\mathbb{E}$ be the universe of all events, then f_ω defines a mapping from $\mathbb{S}_I \subseteq \mathbb{S}$, which is the power-set of incoming events ($\mathbb{S}_I \subseteq 2^{\mathbb{E}}$), to the power-set of outgoing events ($2^{\mathbb{E}}$); this means $f_\omega : \mathbb{S}_I \mapsto 2^{\mathbb{E}}$, e.g., $f_\omega(\{e_0, e_1, e_2\}) = \{e_9\}$. Hence, each correlation step produces zero, one, or more events as output. Events are evicted from the incoming streams after each correlation step by means of a *consumption policy*. For example, a sliding window specifies that events, like e_0, with smaller timestamps than the timestamps of events comprised in subsequent selections, like $\{e_1, e_2, e_4\}$, can be evicted. Observe, although the model of executing

selections performs subsequent correlation steps, the processing of a selection can happen asynchronously, i.e., the correlation function can start analyzing the events even if not all events comprised in a selection were streamed to the operator.

Any selection s of an operator $\omega \in \Omega$ *depends on* the processing of events in the subgraph $G(\omega)$ which is induced by operators and sources that can reach ω. For example s_4 in Figure 2.1 depends on e_0, e_1, and e_2 since they cause the detection of e_9. Formally we define depends on relations $e \sqsubseteq_s e'$, $e \sqsubseteq_s s$ and $s' \sqsubseteq_s s$ for any ordered pair (e, s), (e, e') and (s', s) of events $e, e' \in \mathbb{E}$ and selections $s, s' \in \mathbb{S}$:

$$s' \sqsubseteq_s s : (s' = s) \vee ((f_\omega(s') \cap s) \neq \emptyset) \vee (\exists s'' : ((f_\omega(s') \cap s'') \neq \emptyset) \wedge s'' \sqsubseteq_s s) \quad (2.1)$$

$$e \sqsubseteq_s s : (\exists s' : e \in s' \wedge s' \sqsubseteq_s s) \quad (2.2)$$

$$e \sqsubseteq_s e' : (\exists s : e' \in f_\omega(s) \wedge e \sqsubseteq_s s) \quad (2.3)$$

In the example $e_0 \sqsubseteq_s s_4$, $e_1 \sqsubseteq_s s_4$, $e_2 \sqsubseteq_s s_4$, and $e_9 \sqsubseteq_s s_4$. We can extend the definition for a sequence E, such that $E \sqsubseteq_s s$ iff $\forall e \in E : e \sqsubseteq_s s$. In particular, we denote the set of atomic events on which s depends as $A(s)$.

Timestamps and Locations. A complex event can depend on a number of events in a selection with different timestamps. Common ways to perform the timestamping is to assign the maximum or minimum timestamp of the events inside the operator's current selection of events, or to adopt an interval-based timestamp [SGM09, AC05]. Throughout the thesis, a timestamping scheme is used that assigns the maximum timestamp of the selection to each complex event. In general, timestamping can be considered a parameter, which can be specified by the domain expert. The results of this thesis can easily be adapted given that timestamps are assigned deterministically and in monotonously increasing order. Using the same reasoning, the location of a complex event is the bounding box of the locations of all events in a selection. In order to define a total order over all events, sequence numbers and unique ids of sensors or operators are assigned and can act as tie-breakers iff $t(e_j) = t(e_{j'})$.

2.1.2 Distributed Infrastructure Model

Operator graphs are deployed on a federation of max^b distributed computing resources, denoted as brokers, $\{b_1 \dots b_{max^b}\}$. Brokers support reliable communication and have sufficient capabilities to execute operators and buffer historical events. The expected network latency $d(b_i, b_j,)$ between any pair of broker b_i, b_j is well known, e.g., by means of Vivaldi coordinates [DCKM04].

The algorithms presented in this thesis are not tailored towards a specific topology of brokers, however, we often refer to the following infrastructure: Similar to typical mobile infrastructures, e.g., GSM networks or location services, brokers are organized in a spatially-partitioned hierarchy [LR01b, dMLR07], as depicted in Figure 2.2. This hierarchy implies that the communication delay to mobile consumers increases from the edge to the

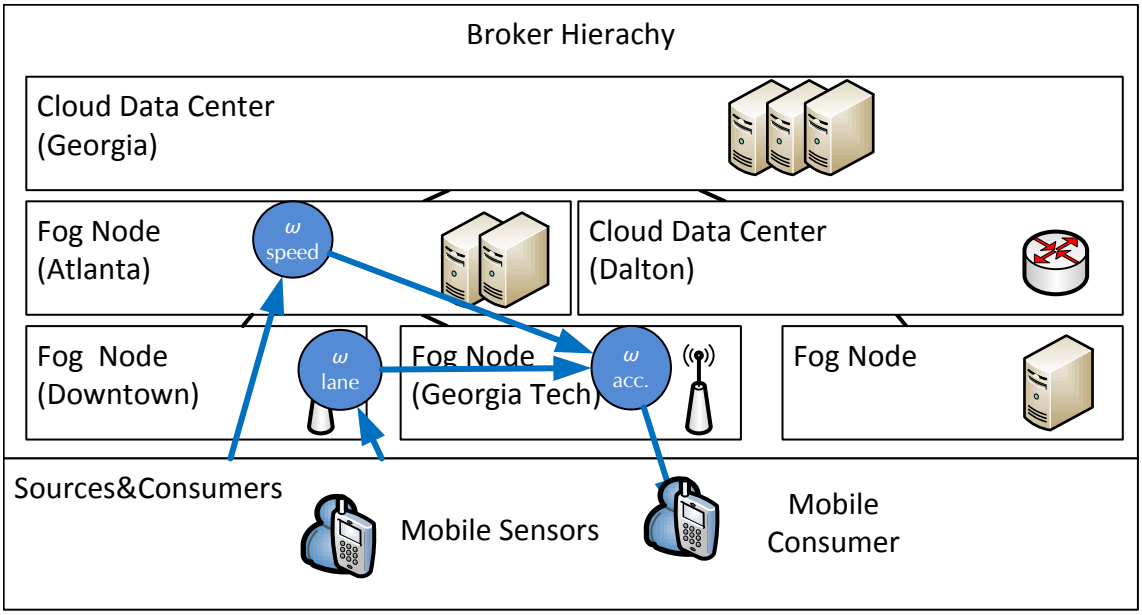


Figure 2.2: Example of a broker hierarchy. Operators are deployed on fog nodes at the edge of the network. Those fog nodes promise low latencies to a consumer who is currently accessing the system from the Georgia Tech campus.

core of the network [HLR+13a]. The spatial partitioning of such a hierarchy allows us to efficiently match atomic events against a processing interest [TDSC07, LDR08].

We further assume that consumers and sources may determine their current time and location, e.g., through a GPS sensor. In order to improve their expected link quality, they greedily connect to the topologically closest broker denoted as *leaf broker*. Mobile consumers and sensors share event streams over a wireless interface with the broker infrastructure. Since processing tasks are deemed to consume a lot of energy,[2] those mobile devices are only thin clients, leaving all the processing to the infrastructure.

2.2 Mobile CEP Query Model

As a next step, the changes in the basic query model of CEP for dealing with dynamic spatial interests and its effects on the delivery semantics are discussed. Note, in a traditional setting, a change in the operator graph is in most cases performed whenever a new query is added or removed. However, in a MCEP system, the interest in the events provided by the operator graph change with the movement of a consumer. Therefore, the sources and the corresponding atomic event streams require constant updates.

[2]Results of a bachelor thesis [Vet12] conducted in conjunction with the work of this thesis showed a high energy consumption for a large spectrum of operators.

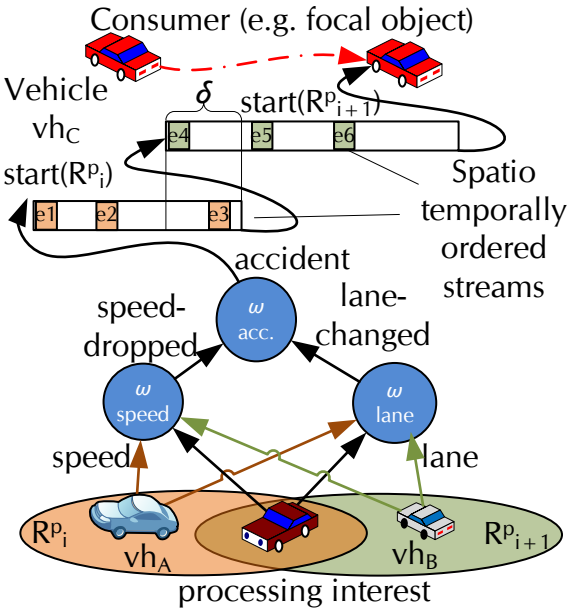


Figure 2.3: MCEP Query for accidents: First, situational information with respect to R_i^p is delivered to a consumer, then with respect to R_{i+1}^p.

2.2.1 Mobile CEP Query

To receive situational information relative to her current location, a consumer registers a *MCEP query*, $Q = \{G, fo, R, \delta, Pol\}$ with the MCEP system. G denotes the application-specific operator graph,[3] fo the focal object of a consumer, R a function to determine the spatial interest which is spanned by fo,[4] δ a lifetime parameter—a temporal interest—that controls the amount of delivered historical situational information, and Pol the delivery semantics. When referring to a query we also refer to its imposed operator graph, whose root is directly linked to the consumer.

In the context of the introduced accident example, a vehicle's head-unit is the focal object and registers a MCEP query with the CEP system for all accidents within a range of 500 m around its current location (see Figure 2.3). To continuously receive situational information while a consumer is moving, the vehicle provides discrete location updates $(l_1, l_2, \ldots)$,[5] resulting in a corresponding sequence of discrete spatial areas $(R_1, R_2, \ldots)$ that represent the spatial interests. Note, when referring to the spatial interest, we implicitly refer to the corresponding spatial area $R_i = R(l_i)$. With a location update l_{i+1}, the operator graph stops processing for the current spatial interest R_i and start processing for the updated spatial interest R_{i+1}, which we further denote as *interest switch*. This way,

[3] Note, this operator graph does not necessarily need to be specified by the consumer. While giving high flexibility to the consumer in specifying individual event streams, an administrator can also predefine operator graphs for valid event streams.

[4] We treat the function to determine the spatial interest as black box which returns an arbitrary polygon or circle.

[5] As surveyed in [LR01a] there already exist numerous location update strategies.

the operator graph is obliged to process *live events* which are currently of interest to a consumer—events that occurred after the spatial interest switched to R_i at time $start(R_i)$. As a result, the consumer is informed about *live situational information*.

Definition 1 (Live event)**.** We say any event e (including atomic events and situational information) occurred live with respect to a spatial interest R_i iff it carries a timestamp $t(e) \geq start(R_i)$.

The temporal interest δ allows consumers to specify situational information that occurred δ time-units before $start(R_i)$. Consider an accident that occurred 30 min before the spatial interest includes the location at which the accident happened. This accident is of interest to the consumer, as it can still block a lane. To this end, consumers receive not only live situational information but also *historical situational information*.

Definition 2 (Historical event)**.** We say any event e (including atomic events and situational information) is historical with respect to a spatial interest R_i iff it has a timestamp $t(e) < start(R_i)$.

In summary, by issuing a MCEP query, the consumer is delivered a spatio-temporal stream that comprises location-dependent historical and live situational information.

Definition 3 (Spatio-temporal event stream)**.** A spatio-temporal event stream comprises for each spatial area $R_i \in (R_1, R_2, ...)$ a sequence of events $E(R_i) = (e_{(1,R_i)}, e_{(2,R_i)}, \ldots)$ where each event of $E(R_i)$ depends on historical or live atomic events with a location in R_i. Moreover, with respect to the time interval $[start(R_i), start(R_{i+1})]$ during which R_i represents a consumer's spatial interest, the sequence $E(R_i)$ comprises events with timestamps from the interval $[start(R_i) - \delta, start(R_{i+1})]$

In the example of Figure 2.3, the spatio-temporal stream comprises historical situational information (e_1,e_2, and e_4) and live situational information (e_3,e_5, and e_6) from R_i and R_{i+1}. In order to distinguish situational information for changing spatial interests, they are stamped with the corresponding area R_i from which they are detected.

2.2.2 Mobile CEP Query Semantics

The policy *Pol* of a MCEP query defines the delivery semantics for the spatio-temporal stream of situational information. The policy specifies a quality—the accuracy with which processing interests match spatial interests, the delivery order, the consistency, as well as the completeness semantics. Since temporal consistency guarantees and event orders are well known, see e.g., [BGAH07], we formally define the relevant mobility semantics in the presence of interest switches, a *spatio-temporal* event order, a *spatial consistency*, and *spatio-temporal completeness* which can be selected by a consumer in the policy *Pol*.

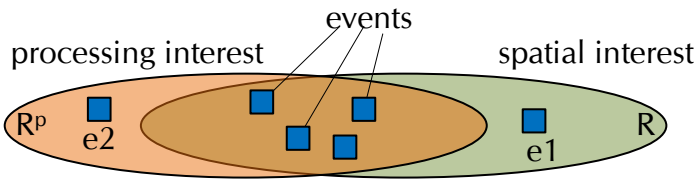


Figure 2.4: Quality of Results

Quality of Results.

Recall that not all MSA applications require to exactly match the spatial interest with the actual input to the operator graph. This observation allows us to decouple the actual spatial interest R_i from the so-called processing interest R_j^p that selects the atomic events as input to G to detect situational information with timestamps in $[start(R_i^p) - \delta, start(R_{i+1}^p)]$. The consumer can specify with the *Quality of Results (QoR)* parameter a relaxation on the overlap of R_j^p and R_i.

Metric. We break the QoR down to two, well-known metrics from the information retrieval domain: *precision* and *recall*. The precision describes how noisy the set of selected atomic events is. Recall indicates how relevant the set of selected atomic events is.

Precision: When processing atomic events from a processing interest R_j^p, which overlaps in large parts with the spatial interest R_i, not all events that lie in R_i are included in the detection of the resulting situational information. For example, event e_1 in Figure 2.4 is not included in the area of the processing interest. 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. Formally, if $A(R_i)$ is a sequence of atomic events selected by R_i and $A(R_j^p)$ is a sequence of atomic events selected by R_j^p, then:

$$precision(R_i, R_j^p) = \frac{|A(R_i) \cap A(R_j^p)|}{|A(R_j^p)|} \tag{2.4}$$

Recall: The recall considers that events can be included in the detection of resulting situational information that are not relevant to the consumer, e.g., event e_2 in Figure 2.4. The recall 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.

$$recall(R_i, R_j^p) = \frac{|A(R_i) \cap A(R^p)|}{|A(R_i)|} \tag{2.5}$$

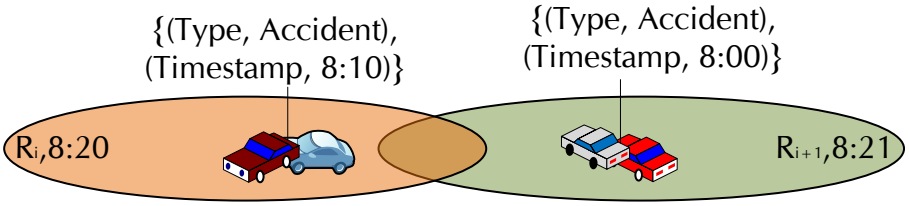


Figure 2.5: The focal object's imposed spatial interest changes form R_1 to R_{i+1}. Due to the order of the processing interests the event with $t(e_2) = 8{:}00$ is delivered after the event with $t(e_1) = 8{:}10$.

The involved actual spatial interest of the consumer and the processing interest can have a low overlap, but the resulting situations may still be meaningful to the consumer when precision and recall are close to 1.

Quality of Results (QoR) is formally an ordered pair $q = (precision, recall)$. For any pair q_1, q_2 we define $q_1 < q_2$ iff precision(q_1) < precision(q_2) and recall(q_1) < recall(q_2). Specifying a quality threshold q_δ in the policy Pol, allows consumers and operators to specify their desired QoR, which our system tries to fulfill at a best effort for each spatial interest.

Spatio-temporal Event Order

In a distributed system, relevant events can arrive at operators or consumers in an arbitrary order, e.g., due to different network latencies to different sources of events [PSB04]. To this end, MCEP allows consumers and operators to choose between different event orders. Note, these definitions apply to arbitrary event streams, they are not limited to streams of situational information.

The temporal order of events has been established to be an important property of event streams [GHAB07, MP13], e.g., to avoid misdetections at operators. Take for example an operator that detects a linear drop in atomic speed events over the last 30 s. If delivered in the right temporal order, e.g., $\{speed, t_1, l_1, 30\,\mathrm{km/h}\}$, $\{speed, t_2, l_2, 20\,\mathrm{km/h}\}$, $\{speed, t_3, l_3, 10\,\mathrm{km/h}\}$, it is a sequential process to just compare the last two events with every new event arrival. Delivered in the wrong temporal order $\{speed, t_3, l_3, 10\,\mathrm{km/h}\}$, $\{speed, t_2, l_2, 20\,\mathrm{km/h}\}$, $\{speed, t_1, l_1, 30\,\mathrm{km/h}\}$, 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. In the worst case, a wrong temporal order can lead to a situation where those events are not comprised in the same selection. However, other classes of operators are resilient to event orders. For example, the result of an operator that filters events with speeds lower than 50 km/h is not altered if events arrive out-of-order.

In a MCEP system, the order of events should also respect the spatial order of updates of its processing interests. For example, to avoid effects like flickering, a navigation

system should be able to display all traffic events that occurred with respect to R_i^p before displaying traffic events that occurred with respect to R_{i+1}^p. Note, such a spatial order does not necessarily result in a temporal order imposed by the timestamps of a stream's events due to the temporal interest in historical events. Consider the example depicted in Figure 2.5. Two accidents occurred at time 8:00 and 8:10. However, due to the order of the processing interests the event with $t(e_2) = 8{:}00$ needs to be delivered after the event with $t(e_1) = 8{:}10$.

Therefore, this thesis proposes two new orders in context of CEP which are formally defined in the remainder: a *spatial order* and a *spatio-temporal order*. For a processing interest R_i^p, let $[start(R_i^p), start(R_{i+1}^p)]$ denote the time interval between the location update that leads to R_i^p and the subsequent one R_{i+1}^p. Furthermore, let $E(R_i^p) := (e_{(1,R_i^p)}, \ldots, e_{(max^e,R_i^p)})$ denote a possible sequence of temporally ordered situational information for which G processes events selected from R_i^p.

Definition 4 (Spatial Order)**.** An event stream is *spatially ordered* iff all events with respect to R_i^p are delivered before events with respect to R_{i+1}^p.

To ensure a spatial order in the example of Figure 2.3 all events of $E(R_i^p) = \{e_1, e_2, e_3\}$ must be delivered before events of $E(R_{i+1}^p) = \{e_4, e_5, e_6\}$. Yet, the temporal order of $E(R_i^p)$ need not be ensured and e_3, e_2, e_1 can be delivered in an arbitrary order.

Definition 5 (Spatio-temporal Order)**.** An event stream is *spatio-temporally ordered* iff it accounts for the sequence of processing interest changes and the temporal ordering within processing interests. The system ensures that all events with respect to R_i^p are delivered before events with respect to R_{i+1}^p. Moreover, all events for R_i^p are delivered in a temporal order, i.e., all events $e_{(j,R_i^p)}$ with $j < j'$ are delivered before $e_{(j',R_i^p)}$.

In Figure 2.3, events are delivered in a spatio-temporal order $e_1, e_2, e_3, e_4, e_5, e_6$.

Spatial Consistency

Spatial consistency ensures to an operator or consumer $u \in \Omega \cup U$ that any operator $\omega \in G(u)$ only processes selections s that depend on atomic events from the same processing interest R_i^p. This means that s comprises incoming events that are either atomic events stemming from the same processing interest R_i^p or are produced as a result of processing such atomic events by their preceding operators.

Definition 6 (Spatial Consistency)**.** A selection s is spatially consistent to R_i^p iff $\nexists a : l(a) \notin R_i^p \wedge a \sqsubseteq_s s$.

In Figure 2.3 it means that no operator processes a selection of incoming events that comprises events stemming from vehicle vh_A and vehicle vh_B.

Temporal completeness.

If all situational information is delivered and streamed in a spatio-temporal ordering with respect to a processing interest and a temporal interest, the result is said to be *temporally complete*. Intuitively, ensuring temporal completeness results in an increased latency if the temporal interest δ is large, since a potentially huge number of events has to be processed. We therefore discuss a strict completeness guarantee and a relaxation, which we call best effort completeness.

In order to be able to formally define temporal completeness, we first introduce the notion of a *covering sequence*. Recall, for each processing interest R_i^p, a consumer is interested in a sequence of situational information with timestamps from the interval $[start(R_i^p) - \delta, start(R_{i+1}^p)]$.

Definition 7 (Covering Sequence)**.** Given a query $Q = \{G, fo, R, \delta, Pol\}$ and a sequence of processing interests $(R_1^p, R_2^p, \ldots)$, $E(R_i^p)$ is a *covering sequence* of situational information for R_i^p iff for all $e \in E(R_i^p)$ the temporal restriction $t(e) \in [start(R_i^p) - \delta, start(R_{i+1}^p)]$ holds.

Note, this definition can not only be applied to a sequence of situational information, but any event sequence $E(R_i^p)$ that is produced by an operator ω with timestamps from an arbitrary temporal interval $[start(E(R_i^p)), end(E(R_i^p))]$.

Observe, a covering sequence of an operator ω's outgoing event stream can be the empty set, e.g., if ω cannot detect any events covering R_i^p. Moreover, it is not sufficient to define temporal completeness (informally) as "a covering sequence comprises all complex events that can be detected in the time interval $[start(E(R_i^p)), end(E(R_i^p))]$". The reason for this is that a covering sequence highly depends on the sequence of selections $S(E^I)$ of ω's incoming events E^I as we detail in the following example.

Consider an operator graph that only comprises a sum operator $\omega_{(+,10)}$. The operator selects individual events that carry an integer value from two input streams I_A and I_B; as a result $\omega_{(+,10)}$ produces output events comprising the sum of the integer values, iff the sum is greater than 10. Processed events are evicted from I_A and I_B. Moreover, consider that a location update that leads to R_i^p occurs at time $t = 5$ and a subsequent location update that leads to R_{i+1}^p occurs at $t = 9$. If the temporal interest is set to $\delta = 2$, a covering sequence of R_i^p produced by $\omega_{(+,10)}$ only contains situational information with timestamps in the temporal interval $[3, 9]$. Further consider that a MCEP system can stream the following input event sequence comprising events of type A and B after the location update of R_i^p to R_{i+1}^p. Note, each event carries a timestamp besides the integer value, e.g., $e_1 := \{(timestamp, 1), (integer\ value, 5\})$:

$$\{e_1 := \{A, 1, 5\}, e_2 := \{A, 2, 8\}, e_3 := \{B, 4, 3\}, e_4 := \{A, 8, 5\}, e_5 := \{B, 9, 7\}\}$$

Observe now that a MCEP system can consider to initialize the event sequence at e_1 or stream less events by initializing it at e_2; the above informal definition of temporal completeness would apply in both cases. When using a selection policy that selects events

as input iff they are at most 4 time units apart the following sequence S_1 of selections is generated starting from e_1: $(s_1 := \{e_1, e_3\}, s_3 := \{e_4, e_5\})$. This leads to the outgoing event sequence $E_1(R_i^p)$ as described by Equation 2.6. Starting from e_2 a second sequence of selections S_2 is generated: $(s_2 := \{e_2, e_3\}, s_3 := \{e_4, e_5\})$, leading to an outgoing event sequence $E_2(R_i^p)$ as described by Equation 2.7.

$$E_1(R_i^p) = (o_2 := \{(Sum, 12), (timestamp, 9)\}) \quad (2.6)$$

$$E_2(R_i^p) = (o_1 := \{(Sum, 11), (timestamp, 4)\}, o_2 := \{(Sum, 12), (timestamp, 9)\}) \quad (2.7)$$

For both sequences, no other event with a timestamp in the temporal interval $[3, 9]$ could have been detected. Recall, detected complex events are stamped using the maximum timestamp of events comprised in a selection. Since e_3 is the first incoming event matching the temporal interval $[3, 9]$, processing either selection $\{e_1, e_3\}$ or $\{e_2, e_3\}$ would each result in the first possible detection of an outgoing event comprised in the interval $[3, 9]$. Following the same argumentation, o_2 is the last possible detection of an outgoing event comprised in the interval $[3, 9]$. Moreover, in our example is $E_1(R_i^p) \subset E_2(R_i^p)$ while $S_1 \not\subseteq S_2$ and $S_1 \not\supseteq S_2$. To be able to deal with these ambiguities, we introduce the notion of a maximum covering sequence:

Definition 8 (Maximum Covering Sequence)**.** Let S be a sequence of selections that leads to the covering sequence $E(R_i^p)$ and S' be the sequence of selections that leads to $E'(R_i^p)$. $E(R_i^p)$ is called a *maximum covering sequence* of R_i^p iff there exists no further covering sequence $E'(R_i^p) \neq E(R_i^p)$ such that $E(R_i^p) \subset E'(R_i^p)$ and $S \subset S'$.

Note, a maximum covering sequence is still not necessarily uniquely defined, i.e., both $E_1(R_i^p)$ and $E_2(R_i^p)$ are maximum covering sequences. However, for each distinct sequence of selections exactly one maximum covering sequence is defined.

Temporal completeness. Due to the previous observation about the ambiguity of maximum covering sequences we adopt a weak and strong formulation for the strict temporal completeness:

Definition 9 (Weak temporal completeness)**.** A MCEP system ensuring a *weak temporal completeness* delivers for every processing interest $R_i^p \in (R_1^p, R_2^p, \ldots)$ all events of *a* maximum covering sequence $E(R_i^p)$.

Definition 10 (Strong temporal completeness)**.** Let $E_1(R_i^p)$ and $E_2(R_j^p)$ be two maximum covering sequences for G w.r.t. two distinct focal objects fo_1 and fo_2. A MCEP system ensuring a *strong temporal completeness* ensures for every pair of processing interests (R_i^p, R_j^p), which refer to the same spatial region, that $E_1(R_i^p)$ and $E_2(R_j^p)$ will comprise exactly the same events within their temporal overlap $[\max\{(start(R_i^p), start(R_j^p)\}, \min\{end(R_i^p), end(R_j^p)\}]$.

The strong temporal completeness assures to consumers that maximum covering sequences of a specific processing interest R_i^p are consistent to other maximum covering sequences of that processing interest referring to different focal objects. In the above example the strong semantics assures that if the location update of a focal object fo_1 for R_i^p occurred at $t = 5$ all events e of a maximum covering sequence with $t(e) \in [3, 9]$ are generated. In the presence of a location update of a focal object fo_2 for R_j^p at $t = 6$ which is referring to the same range as R_i^p, receiving all events e with $t(e) \in [4, 10]$, consumers will receive identical covering event sequences within the temporal overlap $[5, 9]$.

Recall, a potentially large number of events has to be processed to ensure the strong temporal completeness if δ is large, which can increase the latency. To this end, we define a *best effort completeness*.

Definition 11 (Best effort completeness)**.** The system delivers all situational information $E(R_i^p)$ with respect to the current spatial interest R_i^p that can be produced until an interest switch occurs.

Best effort completeness can result in temporally incomplete data when not enough resources are available to process all (historical) atomic events, e.g., e_3 in Figure 2.3 could be missing. Yet, consumers are typically only interested in situational information of their current spatial interest.

2.3 System Components

We now discuss the logical structure of the MCEP event processing architecture as shown in Figure 2.6. Upon registering a MCEP query Q for a mobile consumer, Q is directed to the logically centralized MCEP controller component to bootstrap the deployment of the query. The controller is then in charge of finding an initial placement for the operators of the operator graph G on the set of brokers [Riz13]. As a result of registering a query, the consumer is continuously updated with situational information.

MCEP's main component is the *MCEP server* which runs on each broker. Operators of the operator graph G and a range query for a consumer's spatial interest R are executed by the MCEP servers' *execution environments* (see Section 3.2.3). Note, if the parameters R, fo, and δ of Q are not provided, our system expects and executes a typical CEP operator graph with respect to a fixed set of sources.

The system includes a spatio-temporal *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. All events in the spatio-temporal event storage are indexed by location, time, and type attributes. To be able to tune the index for different MSA scenarios, we do not dictate an implementation but refer to the large body of related work [TDSC07, LDR08, AN08, LRM12]. Events are evicted from the event storage when no operator graph is expected to request them, which

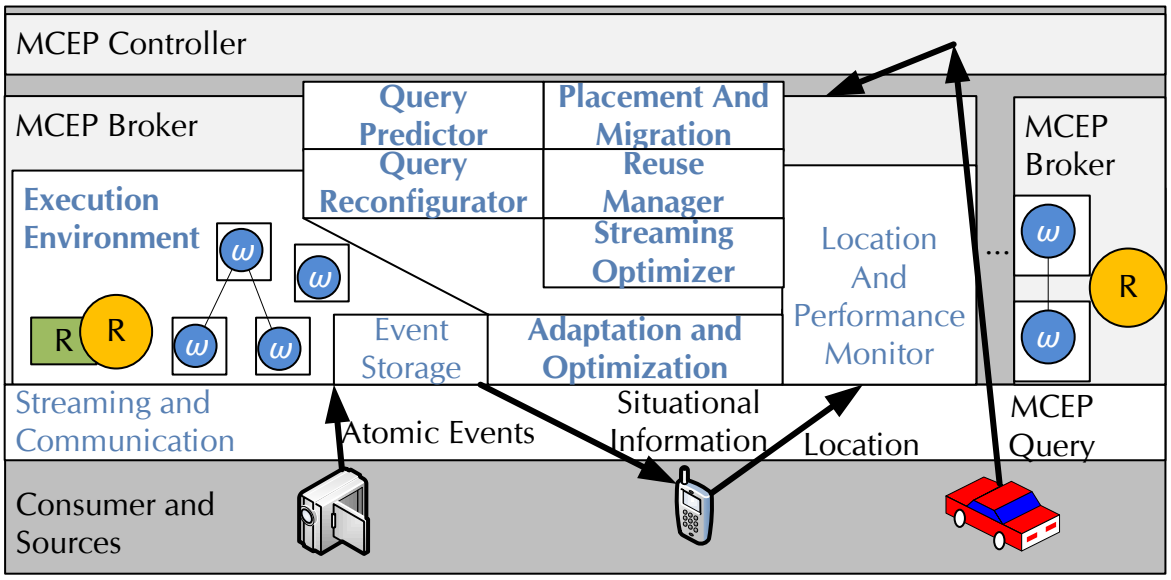


Figure 2.6: MCEP software architecture overview: Consumers register queries with a MCEP broker and receive a stream of situational information in return. A set of MCEP broker perform the query execution and several run-time optimizations for operator graphs. The logically centralized MCEP controller is responsible for maintainence task, e.g., admission control.

is determined based on an dynamic time-to-live, specific for an event type.[6] Moreover, events can be persisted in an external spatio-temporal data-base 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.

MCEP's *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*:

- Location updates trigger the *query reconfigurator* which initiates the reconfiguration of an operator graph. The implemented adaptation algorithms ensure temporally complete and spatially consistent results and trade-off QoR to communication and computation costs (see Section 3.2.4 and Section 3.3.2).
- To reduce the communication and computation overhead during an interest switch, the *streaming optimizer* prevents the streaming of events that have already been sent for a previous spatial interest (see Section 3.3.3).

[6] This crucial parameter can influence the temporal completeness, since relevant atomic events might not be persisted. We discuss, however, a bound for the parameter based on our execution model.

- The *query predictor* is triggered with each location update to support the query reconfigurator with preprocessed historical situational information and this way reduce the latency during interest switches (see Section 3.4).
- 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 preserve a low latency migration by opportunistically starting the migration before the operator needs to be active on a new MCEP server (see Section 4.3).
- Location updates and incoming events trigger the *reuse manager*, which allows our system to scalably process many similar queries. The reuse manager reduces the system's communication and computational load by preventing multiple operators that process events with respect to similar spatio-temporal interests from detecting and streaming redundant outgoing events (see Section 5.3).

3 Mobility-Aware Processing of Individual CEP Queries

This chapter provides MCEP's execution model for individual MCEP queries and its support for latency-efficient and communication-efficient detections of situational information. Recall that we opt for the deployment and execution of individual operator graphs for each individual MCEP query to avoid the inefficiencies of predeploying large numbers of fixed operator graphs. The individual operator graph is continuously reconfigured by adapting its processing interest and operators to the successive spatial interests with respect to the focal object's location. For each distinct processing interest, MCEP processes a (potentially empty) sequence of historical events and a sequence of live events, this way delivering a spatio-temporal stream of situational information that satisfies the spatio-temporal interest of a consumer.

Supporting communication-efficient reconfigurations and low latency detections of historical situational information, however, comprises several challenges. With each update of a query's spatial interest, operator graphs that perform spatial interest specific detections of situational information need to be linked to new atomic event streams with respect to the updated processing interest. The operators of an operator graph are stateful and require with every change in the processing interest also a reconfiguration of the operators' states in order to maintain spatial consistency. For instance, a reconfiguration needs to account for new and obsolete events that result from a changed sequence of atomic events. Furthermore, it is often unclear which historical atomic events may contribute to the detection of a new historical situational information. To this end, the sequence of atomic events that is the input to the operator graph for the updated spatial interest during the reconfiguration has to be anticipated. This way, all situational information of interest to the consumer can be detected, i.e., completeness is achieved, with low overhead. Moreover, in order to provide timely historical situational information with respect to a processing interest, this situational information must already be detected when switching to the processing interest. This requires to start processing early for an uncertain future location of the focal object.

In this chapter we are going to discuss several approaches to perform a reconfiguration efficiently. Broadly spoken, those approaches can be divided into a *reactive* approach and a *proactive* approach:

- The reactive approach initiates a reconfiguration after a changed location of a focal object updates the processing interest of an operator graph. The system subsequently stops streaming atomic events for the previous processing interest, starts streaming

relevant historical atomic events with respect to the updated processing interest from a buffer, and thereafter starts streaming live events. Consistent and spatio-temporally ordered results are provided by injecting so called *markers* into the streams which indicate that no further situational information will be detected with respect to a distinct processing interest. Complete results are achieved by utilizing a novel window-based processing model that allows MCEP to estimate the required sequence of historical events to satisfy the consumers spatio-temporal interest. Moreover, our system can reduce streaming and processing costs by degrading the quality of situational information, i.e., by reducing the accuracy with which the processing interest matches the spatial interest, and reuse state that depends on atomic events from the spatio-temporal overlap of subsequent processing interests.

- The proactive approach extends the reactive mechanisms in order to provide timely historical situational information. In particular, historical situational information is already detected before the focal object's changed location initiates a reconfiguration. Historical situational information can then be streamed from a buffer to the consumer, hiding processing delays for the detection. To this end, MCEP predicts future locations of a focal object and starts to process historical atomic events with respect to corresponding future processing interest. MCEP pipelines the processing with respect to predicted locations at several future times. This avoids bottlenecks that build up back-pressure. Such bottlenecks occur when the temporal distance between subsequent location updates is longer than the time it takes to detect all historical situational information. In addition, our system also deploys operator graphs at other nearby locations. This way, we increase the chance that at least one operator graph provides high quality situational information although the predicted and real locations of a focal object might deviate.

In this context we provide several contributions: (i) a mobility-aware operator execution model, (ii) methods to reduce streaming and processing costs (bandwidth usage and number of correlation steps) when establishing the previously established delivery semantics, and (iii) methods to reduce the perceived latency by a consumer when establishing this delivery semantics. Major parts of this chapter base on the work that has been published in [KORR12], [HLR+13b], and [OKR+14a].

The remainder of this chapter is structured as follows. We will first discuss changes to the system model (Section 3.1). We then discuss our mobility-ware execution model using the reactive approach (Section 3.2), including the operator programming model, before detailing corresponding optimizations (Section 3.3) and extensions to support the proactive approach (Section 3.4). Our evaluation results of both, the reactive and proactive approach, are presented (Section 3.5) before discussing the related work (Section 3.6) and concluding the chapter with a short summary (Section 3.7).

3.1 System Model

We rely on a similar system model as in Chapter 2. Operator graphs run on distributed computing resources—the brokers. This allows us to serve a large number of operator graphs in parallel while each operator graph potentially involves processing a large amount of event streams. Each broker can deploy operator graphs on topologically close computing resources that can be accessed with neglectable small latencies. A dynamic number $\rho_{\max}$ of unused resources (CPU, memory, ...) on nearby brokers is well-known. These resources can be pre-allocated to ensure that the operator graphs' expected workloads will not overload the brokers.

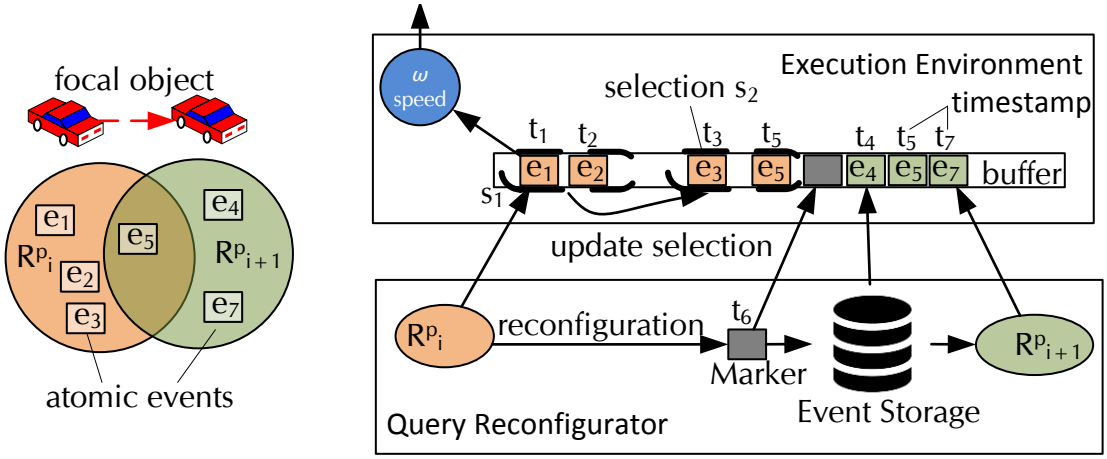
3.2 Mobility-aware Operator Execution

This section covers the concepts of MCEP's *operator programming model*, *execution environment*, and the *query reconfigurator* [KORR12, OKR+14a]. The problem addressed in the next sections is to establish an execution environment for a MCEP system that yields the introduced delivery semantics by providing guarantees for temporal completeness, spatial consistency, and a spatio-temporal event delivery order iff no quality is to be degraded ($q_\delta = (1, 1)$). In Section 3.2.1 we give a brief overview over the concepts for the mobility-aware operator execution. In Section 3.2.2, we introduce the basic concepts of MCEP's operator programming model and its API. In particular, we focus on MCEP's expressive window-based selection concept. In Section 3.2.3, we give a reference algorithm to clarify the operator's execution by determining and processing a sequences of selections. In Section 3.2.4 we then detail the basic mobility-driven reconfiguration algorithm that uses the dependency of windows over several levels in the operator graph to ensure temporal completeness.

3.2.1 Overview

Recall, the execution of an operator $\omega \in \Omega$ comprises a sequence of correlation steps S. For each correlation step, a selection $s \in S$ of events needs to be determined, for which the correlation function f_ω can be applied to produce an outgoing event sequence $f_\omega(s)$. To determine the sequence of event selections in subsequent correlation steps, programmers need to provide *selection* and *consumption* policies as part of the operator description. The selection policy specifies which events are comprised in a correlation step, the consumption policy decides which events can be evicted from the incoming stream after a correlation step.

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 enforce the delivery semantics with respect to s and $f_\omega(s)$. To this end, the MCEP system keeps a buffer for each incoming event stream; for each processing interest events from preceding operators are appended in a temporal order over all incoming streams to this buffer. In



(a) A focal object updates its location and this way selects other atomic events of interest as input to an operator graph. The selected events are processed by an execution environment (see Figure 3.1b).

(b) An operator determines consecutive selections on its incoming stream and applies its correlation function on the selections. The query reconfigurator initiates the operator's reconfiguration after an interest switch from R_i^p to R_{i+1}^p: Markers indicate that events from the next processing interest will be streamed to the operator; first historical events like e_4, then live events like e_6.

Figure 3.1: MCEP Execution Environment

the example depicted in Figure 3.1, e_1 is appended before e_2. This way, MCEP can enforce a deterministic order over all selections and detected complex events with respect to these selections, i.e., $s_1 < s_2$, as follows: A new correlation step is initiated at the end of the previous correlation step by determining the next selection in the buffers matching a *selection policy*. 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. To ensure a temporal consistent processing, the execution environment receives as a result a signal when the operator finished processing the events of a selection. Finally, an optional eviction of events from the incoming buffers is triggered according to the *consumption policy*. The MCEP system delivers and stamps the outgoing events produced by an operator.

In addition, the execution environment is informed about an interest switch from processing interest R_i^p to processing interest R_{i+1}^p through so-called *markers* in incoming event streams. Markers ensure to the execution environment that no further event will be streamed with respect to R_i^p. Therefore, when selections are identified which comprise markers, i.e., after s_2 has been processed, the reconfiguration of the operator is triggered. Events depending on obsolete historical events, e.g., events that depend on atomic events with locations outside of the current processing interest like e_1 through e_3 in the example,

are purged from the buffers. This way, markers ensure spatial consistency, i.e., markers isolate the processing of individual temporally ordered sequences of events for each processing interest. Markers are initially injected by the *query reconfigurator* with each interest switch and propagated from the leaves to the root of the operator graph. To this end, each execution environment injects a marker in its outgoing stream after the reconfiguration of the operator's state. Moreover, the query reconfigurator determines historical atomic events to ensure temporal completeness for the subsequent processing interest from the event storage (i.e., e_4) and streams those historical events before streaming live events (i.e., e_6).

3.2.2 Operator Programming Model

In order to focus on domain specific solutions, operator programmers delegate the stream management to the MCEP system by means of selection and consumption policies. Moreover, we show in Section 3.2.3 that exposing knowledge on the selection and consumption of events performed by the operator bears a huge potential to reduce the overhead of processing historical atomic events. At the downside, the programmer of a MCEP operator now needs to implement the operator against a specific programming model that exposes knowledge on the current state of an operator's selection as detailed in the following section. In the remainder we detail the concepts behind the selection and consumption policies, the API an operator has to implement, and the resulting expressiveness.

Selection and Consumption Policies

In the literature, stream processing and CEP systems offer a variety of ways to model selections and consumptions. Typical stream processing systems determine a selection by means of a window that specifies a sequence of events on an incoming stream [ABW06, KNV03, HFAE03]. A downside of these systems is that pattern detection, e.g., the detection of a specific sequence of events, is out of their scope. CEP systems [CM10, PSB04] often build on automata and are therefore suited for pattern detection. With each event streamed to the operator a new automaton is created or a state transition is triggered in existing automata until an accepting state is reached. As a result, automaton only represent the processing state of an operator and additional steps need to be taken to expose the operator's selection and consumption.

To get the best of both worlds, we propose to use a partitioned window model for the selection of events of incoming streams. Each incoming stream is associated with its own set of *selection windows SW* implementing a *selection policy*. However, the sequence of events on which the selection can be performed is restricted by five other mechanisms in order to increase the expressiveness of a window-based model. First, a *restriction window rw* which assures that only temporally relevant events can be selected, e.g., acceleration events that date back one day do not indicate a recent acceleration pattern. Second, dependencies *dep* between individual windows assure temporal orders between events of different incoming streams, e.g., for the sequence $\omega_{\rightarrow}$ on two streams E_A and E_B, no

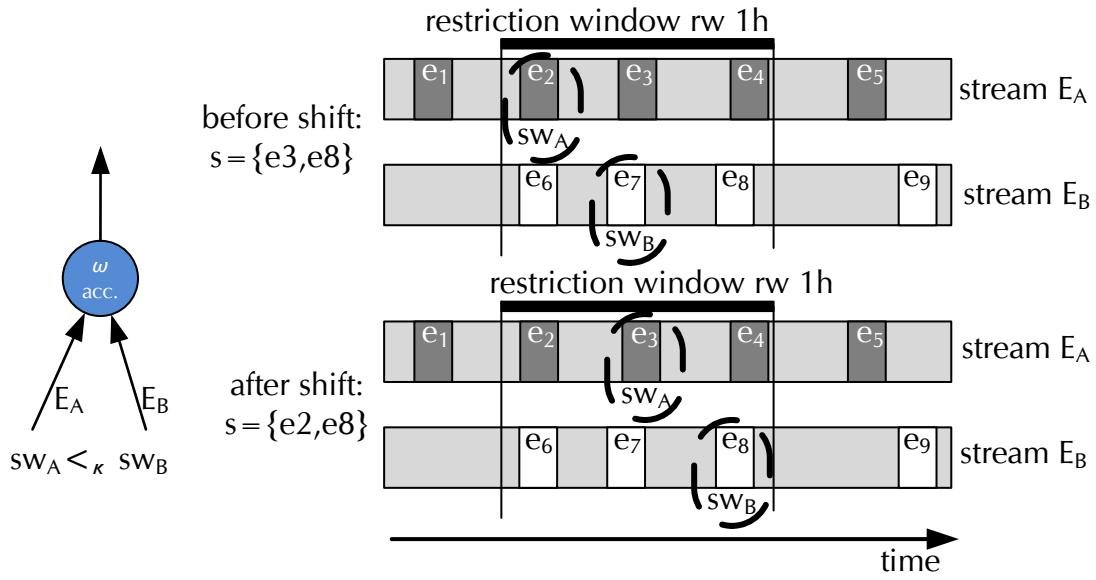


Figure 3.2: Example of a "happens before" operator which selects events from two streams E_A and E_B using two selection windows that each select at most one event. After a correlation step finished, the window sw_B is shifted to the *future* for exactly one event. Window sw_A is allowed to shift for two events to the *future* but is prevented from doing so by the windows' $<_\kappa$ dependency.

selected event of stream E_A should have a greater timestamp than a selected event from E_B. Third, *consumed* events should not be selected. Fourth, *processing predicates* which ensure that only relevant correlation steps are executed, e.g., the processing of $\omega_{\rightarrow}$ should only be invoked if both streams E_A and E_B comprise events. Last, for higher expressiveness, MCEP offers to partition incoming streams and group events according to a *partitioning policy*. The relevant concepts are detailed in the remainder of this section. Note, such concepts can easily be integrated with most existing CEP correlation engines, e.g., by means of a wrapper [SKPR10].

Selection. To determine a selection in the MCEP system, the programmer is obliged to specify at least one *selection window* for each incoming stream of the operator, which determines zero or more events that are included in the selection. Consider Figure 3.2, which depicts a simple "happens before" operator that selects an event from stream E_A before an event from stream E_B. Note that a happens before operator can implement ω_{acc} of the accident example depicted in Figure 1.1. It detects those vehicles that were involved in an accident and therefore drastically reduced their speed before several other vehicles switched a lane to avoid the site of the accident.

Each selection window $sw = (type, length, shift\_policy)$ determines *start* and *end* points in streams, where events in-between are selected for the correlation. In particular, the *type* specifies if a window selects a number of subsequent events of the stream bounded

by a maximal count (*counting_window*) or a sequence of all events of the stream where the difference in the timestamps between the first and last event included in the window is smaller than a fixed time span (*temporal_window*), e.g., sw_A and sw_B for the streams E_A and E_B of ω_{acc} are counting-windows. The *length* determines the respective length of the time span or the count on the number of events, e.g., 1 for sw_A. MCEP allows programmers to define a window with an infinite length $length = \infty$, which selects all events after a distinct *start* point. The *end* point of such windows is then dynamically set at run-time, e.g., by the operator's logic.

The *shift_policy* determines how the start and end points change after each correlation step. With the start of a new correlation step, the selection windows are *shifted*, according to the *shift_policy*—new start and end points are calculated. The window, with respect to either the *start* or *end* point, shifts to the *history* or *future* of the stream for a fixed amount of non-consumed events, a specific time with respect to the time-stamps, the most recently added event, the oldest non-consumed event in the stream, or *not* at all. The first window sw_A in the example can shift one event to the *future* selecting the next event e_3 (see the second step in Figure 3.2). Note that the selection s of an operator comprises the set of all events determined by selection windows SW.

Dependencies. It is often important to detect causal dependencies, e.g., to detect if an accident was responsible for a traffic jam. A typical operator from the CEP domain that detects such dependencies is the sequence operator "happens before" (see Figure 3.2). To this end, MCEP allows programmers to define optional *temporal dependencies* with the selection policy, which can indicate causal dependencies. Dependencies between *start* or *end* points of a pair of windows sw, sw' are defined in form of the binary ordering relation $<_\kappa, >_\kappa, \geq_\kappa$ and $\leq_\kappa$, denoted κ-relations. For example, the relation $\text{end}(sw) <_\kappa start(sw')$ indicates that all events selected by sw have to comprise smaller timestamps than the events selected by sw'. In our example Figure 3.2, no event of stream E_A should have a timestamp greater than a selected event of stream B. Hence, $end(sw_A) <_\kappa start(sw_B)$ is defined by the programmer and the end of window sw_A should not shift further than the beginning of window sw_B—the point indicated by the black arrow.

Temporal dependencies imply that other selection windows can limit the shift of a selection window on an event stream. In the example, sw_A's *shift_policy* indicates that it should shift two events to the *future* while sw_B's *shift_policy* indicates that it should shift by one event to the *future*. If doing so, sw_A would select e_4 with a greater timestamp than e_8 selected by sw_B. To ensure $\text{end}(sw_A) <_\kappa \text{start}(sw_B)$, either sw_B's *start* has to be shifted further to the end of sw_A and select e_9 or sw_A has to end before sw_B by selecting e_3. We resolve this ambiguity through another binary dependency relation which is implicitly defined by the programmer when specifying a κ-relation for an ordered pair (sw, sw'): the first window sw depends on the second window sw'. Both windows are allowed to apply their individual shift policies, however, if after the shift a κ-relation is violated, then artificial window bounds are forced to the depending window. In the previous example, if $\text{end}(sw_A) <_\kappa \text{start}(sw_B)$, then $\text{end}(sw_A) = \text{start}(sw_B)$ and e_3 is selected by sw_A.

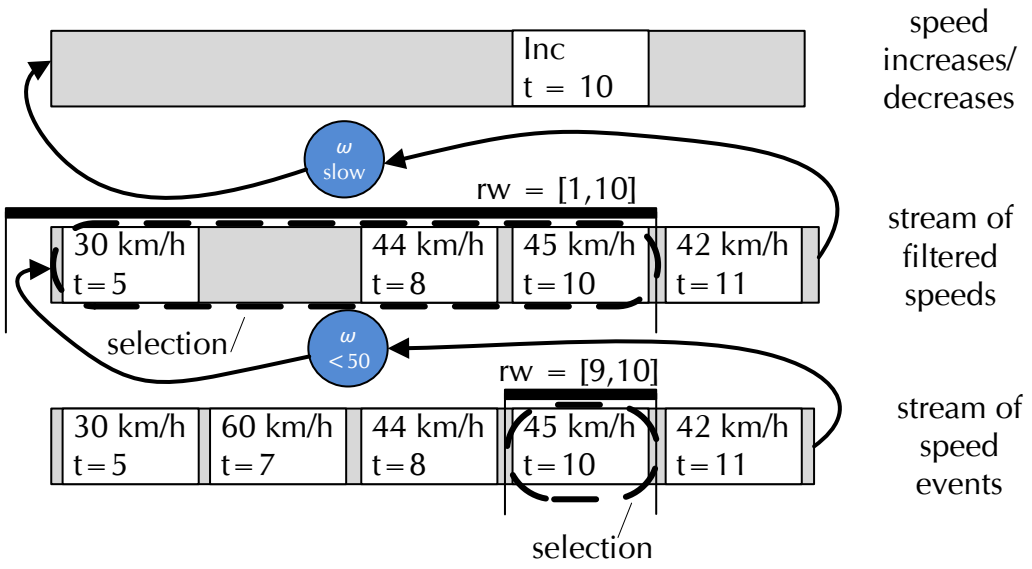


Figure 3.3: Example: Events depend on historical atomic events, Inc at t=10 depends on 30 km/h at t=5.

To avoid further ambiguities, any window is only allowed to depend on one other window. The exception are dynamic selection windows with $length = \infty$, which can depend for each, their *start* or *end* point, on one non-dynamic window, this way determining these points. Cyclic dependencies, e.g., $end(sw) <_\kappa end(sw') <_\kappa \text{start}(sw) \leq_\kappa end(sw)$, over one or more *start* and *end* points are prohibited.

Restrictions. Events are typically only relevant for a short time. For instance, an accident will typically not block a lane for two days straight, hence all relevant events that indicate an accident must occur within a shorter time window. Moreover, we argue that it is important to be able to limit the temporal relevance of an event to ensure temporal completeness. Consider an operator graph detecting if the traffic slows down or speeds up in an area. A simplified version of such an operator graph is depicted in Figure 3.3. The operator graph receives as input speed events from vehicles in the area, detects with a filter operator ($\omega_{<50}$) outliers (i.e., events with speeds of above 50 km/h), and finally detects if the average speed over the last three speed events dropped or increased with a second operator ω_{inc}. The latter operator selects the last three events happening within a dedicated time-span, e.g., the last ten minutes, to ensure the recency of situational information. However, without the notion of a relevance time for events, MCEP cannot ensure temporal completeness. Intuitively, after an interest switch, MCEP cannot easily decide if there have ever been three historical events detected by $\omega_{<50}$, without checking a potentially large amount of historical speed events against $\omega_{<50}$. In theory an infinite number of speed events have to be checked against the operator, which is not practical.

To this end, we oblige that programmers define a relevance time span in form of a *restriction window rw* associated to a selection, a distinct temporal range $[\text{start}(rw), \text{end}(rw)]$ on the incoming streams which limits the shift of the selection windows. For example, a selection window of operator ω_{inc} in Figure 3.3 is not allowed to select any event that has a larger timestamp than $t = 10$ and a smaller timestamp than $t = 1$. The restriction window is updated whenever (i) no selection window will select new events within the restriction windows bounds or (ii) the operator indicates with a flag that no event can be detected with respect to the current restriction window. In the example the bound for ω_{inc}'s restriction window is updated after the outgoing event with $t = 10$ is detected. In this case no shift of the selection window to the future is possible. The programmer can specify how the restriction window is shifted when it requires an update. In particular, the restriction window can shift by one event, a count of events with respect to a specific buffer, or it can shift by a fixed amount of time. In the context of the example of Figure 3.3, the next restriction window ends either at $t = 11$ or at a fixed time after $t = 10$. Selection windows are always initialized after a shift of the restriction window by shifting to the most recent event or the most historical event on their respective streams—other rules are possible but not in the scope of this thesis. In the example, ω_{inc} the selection windows end is set to the restriction windows end ($t = 11$), the beginning of the selection window is thus the event with $t = 8$. Note that the restriction window cannot depend on any other window, since it can lead to ambiguities.

Processing predicates. The programmer of an operator can specify a set of optional processing predicates which must be fulfilled in order to apply the operator's correlation function to a selection. This way the programmer can control if a potentially computational expensive function needs to be applied to a selection. A processing predicate can be defined for each selection window $sw \in SW$ as a mapping $PC_{sw} : \mathbb{N} \mapsto \{0, 1\}$ that takes the number of currently selected events by the selection window as input and returns 1 if the correlation function f_ω should be invoked or 0 otherwise. By default, events are processed by the correlation function when the number of events in sw is greater than or equal to zero (i.e., $|sw| \geq 0$). The keywords *all* and *none* specify that either all events that will eventually be comprised in the selection need to be selected or none at all. All processing predicates of the selection s are combined in the function $P(s)$ using the logical operators $\wedge$ and $\vee$. For example, processing predicatees for the selection windows in Figure 3.2 might be combined to Equation 3.1 to indicate that if at least one event in each window is present the operator may receive the events.

$$PC(s) = \begin{cases} 1 & \text{if } (sw_A \geq 1) \wedge (sw_B \geq 1) \\ 0 & \text{else} \end{cases} \tag{3.1}$$

Consumption. A *consumption policy* allows programmers to explicitly evict events from incoming buffers, i.e., it allows the programmer to specify to evict a fixed number beginning with the first (or last) of the selected events, all events before a time given in a

signal from the operator, a set of events specified in the signal from the operator, or all selected events.

Note that our system can implicitly evict events from a buffer iff the events' timestamps are smaller than the start point of the restriction window. Moreover, our system can also evict events based on the smallest start point of corresponding selection windows, iff selection windows only shift to the future and depend on selection windows that only shift to the future.

Stream Partitioning. Operators often need to detect events on partitioned incoming streams. Consider an operator detecting average speeds: the operator might want to detect the average with respect to speed events from different street segments, with respect to each individual vehicle within the spatial interest, or with respect to types of vehicles. MCEP supports three optional ways to unburden the programmers of operators from partitioning the streams themselves. In particular, MCEP allows to partition all incoming streams by source, by region, or by a common attribute.

When partitioning a stream by source (i.e., using a unique source *id*), each partition is supported with an individual set of buffers and windows, for each partition an individual sequence of selections is determined, and f_ω is applied for each sequence individually. Note that to be able to partition according to a source *id*, all predecessors of the operator have to be sources (selected by processing interests), operators that process individual events, or operators that are partitioned in the same manner.

When partitioning by spatial region, e.g., partitioning the spatial interest into fixed spatial grid cells or according to a street map, each region is supported with an individual set of windows, incoming buffers, and initialization points. Note that to be able to assign events to (spatial) regions, a distinct location stamp is required, which is only ensured if all predecessors of the operator are either sources (e.g., selected by processing interests), or operators that process individual events (i.e., select one event from one incoming stream), or operators that are partitioned in the same way by (spatial) region.

When partitioning by attribute, events are grouped by a common attribute, where each group has its own set of buffers, windows and initialization points.

Expressiveness.

The MCEP system relies on a variety of windows with different consumption and selection policies on each incoming stream for a high expressiveness. In contrast to many state-of-the art systems, the MCEP system is expressive enough to support a hybrid system that comprises typical stream processing [AcT08] and typical CEP operators [PSB04, CM94] due to the partitioned window model. Yet, we constrain the expressiveness with a restriction window. We will show, however, in context of the location-aware reconfigurations, how temporal completeness is ensured through restriction windows.

In the remainder, we will show how the typical operators Atoms, Negation, Concatenation, Sequence, Iteration, Alternation, Timing, and Parallelization as discussed in [PSB04] can be realized using the MCEP system. Note, however, that the actual detection is done

by the operator and not by the MCEP system, which only supports the operator with the selection.

Atoms. The detection of (atomic) events, this is, filtering streams for specific types of events, is a basic operation. In MCEP this is realized by partitioning the streams according to the corresponding event type and selecting events using a counting window of length 1 that always shifts one event to the future and a consumption policy that removes the contents of the selection after each correlation step. Moreover, the restriction window always shifts one event to the future.

Negation. A negation operator detects that an event did not occur, e.g., during a time interval. To this end, it can be supported by implementing a temporal selection window sw, a restriction window of the same length, and a processing policy $|sw| = 0$.

Concatenation and Sequence. Concatenation and sequence are binary operators (i.e., they have two incoming streams E_A and E_B) and are closely related, both detect if an event $e_a \in E_A$ happens before an event $e_b \in E_B$. They differ in systems where time intervals are assigned as timestamps. A concatenation is the weaker version of a sequence: the first operator allows timestamps of e_a and e_b to overlap, where the second requires non-overlapping timestamps. Interval-based timestamps in MCEP means for counting windows of length 1 to have *start* and *end* points that relate to the beginning and end of the interval. Therefore, the first, weak version is implemented using a pair of counting windows (sw_A,sw_B) of length 1 on both incoming streams and defining a dependency $\text{start}(sw_A) <_\kappa \text{start}(sw_B)$. A sequence simply has a different dependency: $\text{end}(sw_A) <_\kappa \text{start}(sw_B)$.

Iteration. An iteration detects the repeated occurrence of a pattern. For the selection, this means that all events from a specific stream need to be analyzed by the operator in order to find patterns, which is best implemented by having a selection window of size ∞. The consumption is then managed by the operator, e.g., by consuming all events before the potential start of a sequence. When shifting the restriction window, it shifts to the oldest event in the stream.

Alternation and Paralellization. Alternation and Paralellization are binary operators, with two incoming streams E_A and E_B, that detect if either $e_a \in E_A$ or $e_b \in E_B$ are detected (alternation) or if $e_a \in E_A$ and $e_b \in E_B$ are detected (parallelization). Both operators are implemented using counting windows of size > 0. The processing predicate can be set to $sw_A \geq 0$ and $sw_B \geq 0$ for the alternation and to $sw_A \geq 1$ and $sw_B \geq 1$ for the parallelization.

Timing. The timing operator detects the occurence of events within a time span. Since MCEP supports temporal windows, this operator is straight forwardly implemented.

Operator API

In order to interact with the execution environment, e.g., to be informed about the correlation steps, each operator has to implement the API defined in Table 3.1 in addition

Function	Description
open()	Operator is informed about a new selection
$f_\omega(Set\_of\_Events)$	Operator is informed about new events
$close() \rightarrow Set\_of\_Events$	Operator is informed about end of selection and returns events that can be consumed

Table 3.1: API each operator needs to implement

to exposing the previously defined selection and consumption policies. The operator can react to the start of a correlation step by implementing *open()*, e.g., to prepare relevant data structures like instantiating an automaton. The correlation function $f_\omega()$ allows operators to be informed about changed selections. It is called asynchronously whenever a new event (or set of events) is added to the selection, e.g., values of received events can already be aggregated. With *close()* the operator can react to the end of a correlation step, e.g., it can clean up stale processing state. Note that MCEP assumes that the operator is stateless in between correlation steps. Moreover, *close()* is a blocking API call, which allows the operator to block the callee until all processing is finished and returns references to events that need to be consumed.

3.2.3 Operator Execution Environment

We now discuss the operator's execution environment by means of a reference algorithm. This algorithm focuses on the correlation steps in between interest switches. At first, to understand the basic principles, we detail the algorithmic steps of the execution environment to process correlation steps sequentially (see Algorithm 3.1). A particular focus is on the mechanisms to adapt selection and restriction windows (see Algorithm 3.2). However, we also discuss in this section how MCEP can determine and process correlation steps in parallel. We conclude by discussing properties of the algorithm.

Initialization. The setup of an operator ω's execution environment is performed with the deployment of an MCEP query $Q = \{G, fo, R, \delta, Pol\}$ (see Algorithm 3.1). The execution environment ensures that references to the events for a selection s will be present locally in a buffer B such that the correlation function f_ω can be applied.

To achieve temporal completeness with respect to the initial spatio-temporal interest, the execution environment initially determines for each of its incoming streams $I = (in, \omega) \in G$ an initialization point, a point in time $\Delta_i(R_1^p, I)$ from which the event stream needs to be processed in order to generate a maximum covering sequence of the initial processing interest R_1^p. Recall that the timestamp of a complex event $o \in f_\omega(s)$ comprises the timestamps of multiple incoming events of the selection s. Therefore, this point in time needs to be individually determined for each input stream as well as for every operator in G. To guarantee a maximum covering sequence for a spatio-temporal interest,

Algorithm 3.1 Basic Execution Environment

```
1:  Requires: Operator ω
2:  Defines: B[(in, ω)] ← ∅ //Event buffer for individual incoming streams
3:           s ← ∅ //Currently opened selection
4:  upon initialization execution environment()
5:   while execution environment is active do //Loop over correlation steps
6:      determine_selection(s, B, selection_policy(ω)) //Open selection and inform ω
7:      call open() of ω
8:      s ← find_matching_events(s, B, selection_policy(ω))
9:      if PC(s) : call f_ω(s) of ω // (Incremental) changes streamed to ω
10:     while ¬ closeable(s, B, selection_policy(ω)) do
11:        e, I ← wait_for_next_event(B) //Wait until event e is added to B[I]
12:        if match(e, I, s, selection_policy(ω)) then //Restrict selection using policy
13:           s ← s ∪ {(e, I)} //Add event to current selection
14:           if PC(s) : call f_ω(s) of ω // (Incremental) changes streamed to ω
15:        end if
16:     end while
17:     deletes ← call close() of ω //Inform ω about closed selection s
18:     comsume_events(s, B, consumption_policy(ω), , deletes)
19:  end while
20: end
21: upon receive event(Event e, Incoming stream I)
22:  B[I] ← B[I] ⋃ {e} //Concurrently add events to local buffer
23: end
```

this requires a coordinated decision of all operators in G, which will be introduced in Section 3.2.4.

Correlation Step. A correlation step is characterized by three phases: *opening*, *processing*, and *closing* a selection (i.e., one iteration of the loop in Line 4-Line 20). When opening the selection, the system determines the constraints that need to be adhered to by the events comprised in the open selection according to the operator specific selection policy, e.g., for selection and restriction windows new start and end points are calculated. An operator is then explicitly informed by the MCEP system about the opened selection (Line 6).

In order to process a selection, events in the local buffers are matched against the windows of the opened selection (Line 8) and streamed to the corresponding operator if they are comprised in the selection (Line 9). However, at the time when a selection is opened, not all events that are eventually comprised in the selection might be available in the local buffers, e.g., a relevant atomic event might not be detected yet by a sensor. Events that arrive from preceding operators or sources are therefore asynchronously added to the local buffers (Line 11), matched against the windows of the open selection (Line 12),

Algorithm 3.2 Management of Selection and Restriction Windows

```
1: function closeable(Selection s, Buffer B, Policy sp = (SW, rw, dep))
2:   return ∀sw ∈ SW : (is_full(s, sw, dep) ∨ ∃e ∈ B : t(e) > end(sw)) ∨ finished(ω)
3: end
4: function determine_selection(Selection s, Buffer B, Policy sp = (SW, rw, dep))
5:   if ((∄sw ∈ SW : can_shift(sw, rw, B)) ∧ (∃e ∈ B : t(e) > end(rw)))
       ∨ finished(rw) then
6:     update_restriction(s, sp, shift_policy(rw) //Initialization of Restriction
7:   else
8:     update_selection(s, sp, shift_policy(SW)) //Change Selection
9:   end if
10: end
```

and streamed to the corresponding operator if comprised in the selection (Line 14). This way, the correlation function can already process partial selections asynchronously while waiting for relevant events.

When all events of a selection are streamed to the operator (Line 10), the MCEP system closes the selection and explicitly informs the operator about its end. The execution environment blocks until the operator finished processing the events of a selection (Line 17), which then triggers an optional eviction of events from the incoming buffers with respect to the operator specific consumption policy.

Updating Selection and Restriction Windows. The operations determine_selection() and closeable() in Algorithm 3.1 can be implemented using MCEP's selection and restriction windows. A selection is closeable when (i) all windows are filled (i.e., counting windows have selected the maximum count of events), or (ii) arriving events won't fit the temporal restrictions of the window (i.e., counting windows have selected less than the maximum count of events but cannot expand due to dependencies), or (iii) the operator indicates with a flag that all relevant evens were detected on a partial selection (i.e., a specific pattern cannot be detected by the operator's logic) (Line 2, Algorithm 3.2).

The restriction window's update of its start and end point is either triggered when no selection window can shift within then restriction window's bound and at least one event in the incoming buffers has a larger timestamp than end(rw) (Line 6). The first condition ensures that all possible correlation steps within the restriction window's bounds are carried out. The second condition ensures that no selection within the restriction window's bound is missed when not all events covered by the restriction window have arrived yet over an incoming stream. Note that the update of a restriction window is a blocking operation since not all events might be present in the local buffers that are covered by the restriction window, e.g., the concrete bounds for a restriction window that shifts for a certain count of events can only be calculated after all necessary events arrive. Otherwise

new start and end points within the restriction's bound are calculated for all selection windows (Line 8).

Event Stamping. An operator asynchronously informs the execution environment about detected complex events. For each successor of ω, i.e., $out \in G$, the execution environment provides an individual outgoing sequence that matches the specified restrictions on the attribute-value pairs of $f_\omega(s)$. The MCEP system delivers and stamps the outgoing events produced by an operator with the largest timestamp comprised in the selection and, to avoid ambiguities, with a sequence number. To ensure a strictly monotonically increasing temporal event order of streams, when selection windows are shifted to the history, we stamp events in those cases with the maximum timestamp of the restriction window. Note that other timestamp mechanisms are applicable, but for the sake of clarity we rely on this mechanism for the rest of the thesis.

Buffer Management. To ensure a temporal ordering of events in incoming streams, the system relies on FIFO channels between operators. Moreover, to ensure the temporal ordering over all incoming streams, the system queues incoming events before they are actually inserted in the local buffers. The system only inserts queued events in the corresponding incoming buffers if over all FIFO channels events with fresher timestamps arrived. The system uses heart beat messages that are injected into the stream to ensure progress. For other implementations we refer to the literature [MP13].

Events that are buffered in the execution environment are only evicted after a correlation step finished and the selection is closed, in order to allow operators to process selections asynchronously without interruption. For large selections, this may accumulate to a vast number of events and may strain the available memory. Since memory is not in the focus of this theses, we only outline a solution for this problem. In particular, each operator has to acknowledge when an event e is processed for a selection s. Event e can then be evicted when all selections that comprise e triggered a corresponding acknowledgment.

Parallelism. Algorithm 3.1 processes each correlation step sequentially, in order to prevent disambiguities for dependent correlation steps. Consider, as an example, that an event e is comprised in two overlapping selections s, s'. If an operator signals that after processing s the event e needs to be consumed, it wouldn't be comprised in s'. However, if processing for s' is started in parallel to s, it would comprise e. To this end, we restrain our system to only process selections in parallel iff no consumption policy is defined.

The major differences for processing several correlation steps in parallel can be summarized as follows. First, the system has to check with every event, if a selection needs to be opened, instead of checking it after processing for a previous correlation step has finished. For example, for a selection window with no dependencies, which shifts by one event, each new event that arrives over the corresponding event stream opens a new selection. Second, when processing all open selections s in parallel, Lines 8-12 of Algorithm 3.2 are executed in parallel for each s. Moreover, to avoid inconsistencies, the programmer of an

Algorithm 3.3 Basic Reconfiguration of Query

1: *Requires:* Dynamic Interest Query Q
2: **upon** update_operator_graph(`Location` l, `Time` $t = start(R^p_{i+1})$)
3: stop_query_after($R^p_i(Q)$,t) //Stop atomic events and situational information
4: $R^p_{i+1}(Q) \leftarrow$ interest(Q,l) //Determine processing interest
5: $\forall src \in D(Q)$: **send** Marker($t, R^p_{i+1}(Q)$) to successor(src)//Inject marker in streams
6: **trigger** find_initialization_points(Q,$R^p(Q)$,t)
7: **end**

8: **upon** initialization_point_found(`Interest` R^p_{i+1}, `Init. Point` Δ_i) **for** `Source` src
9: start_streaming(R^p_{i+1},Δ_i,src,Q) //First stream historical, then live events
10: **end**

operator must keep separate processing states for each selection, while MCEP informs the operator about changed selections. Third, events can only be evicted from incoming buffers if all selections that comprise them are closed.

3.2.4 Location-aware Adaptation of Operators

In addition to selecting and disseminating events—in order to account for the mobility-aware delivery semantics—the execution environment also has to ensure a reconfiguration of the operator's state if updates of spatial interests for a MCEP query occur. To this end, we first describe how the MCEP system reacts to location updates and triggers the adaptation of the complete operator graph, before we describe how the execution environment performs location-aware adaptations.

Initialization of Operator Graph Adaptation

Recall, that a consumer's location update triggers the interest switch, which is performed by the query reconfigurator. To clarify the basic reconfiguration steps, we now discuss Algorithm 3.3 which performs the interest switch 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 to ceases the streaming of a range query that is deployed on behalf of the current processing interest R^p_i (Line 3). The query reconfigurator blocks until all events with smaller timestamps than $start(R^p_{i+1})$, 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 processing interest R^p_i is stopped by injecting a marker in each atomic event stream (Line 5), which allows the system to isolate the streams for each individual processing range and clearly identify the end of a complete stream for R^p_i. The query reconfigurator then updates the range query, which is similarly to [XECA07] 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^p_{i+1} by streaming historical atomic events

to the operators from the event storage with respect to R^p_{i+1}. Each historical atomic event stream AE starts at a specific time denoted as *initialization point* $\Delta_i(R^p_i, AE)$ (Line 9). 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^p_{i+1}) and the time the system switches to the live stream are buffered to avoid event losses.

Adaptation of the Execution Environment

The *marker messages* inform the execution environments of the operators of G about the timestamps start(R^p_{i+1}) of the interest switch and a new processing interest R^p_{i+1}. Each marker $\mathbb{M}$ implies that from now on the streams contain only events that are selected and processed relative to R^p_{i+1}. Note, that at this point in time the operator may still need to produce events with respect to the old processing interest to ensure temporal completeness, e.g., when the operator awaits events with a smaller timestamp than start(R^p_{i+1}) on a second incoming stream. The execution environment therefore prevents selection windows from shifting if they would select a marker $\mathbb{M}$ in the next correlation step or events after $\mathbb{M}$ in the stream. Furthermore, if all incoming streams comprise a marker $\mathbb{M}$, 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^p_i are consumed, the operator is informed about R^p_{i+1}, and processing on the events of R^p_{i+1} is started by initializing the restriction window at an *operator specific initialization point* Δ_i. Note that best effort completeness allows our system to immediately cease processing for a processing interest R^p_i when at least one stream comprises a marker for R^p_{i+1}.

Finding Initialization Points

Consumers define with the MCEP query their interest in historical situational information starting with timestamps greater than $start(R_{i+1}) - \delta$. The *initialization points* of operators and sources, however, do 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 3.3 where $start(R^p_{i+1}) - \delta$ is $t = 10$. For temporally complete situational information about smoother traffic, all atomic speed events with timestamps $t \geq 5$ are required in order to detect an increase in speed events with values $< 50\,\mathrm{km/h}$.

Therefore, to find initialization points, our system has to consider dependencies on historical (atomic) events over several levels in the operator graph hierarchy. Note, due to the dependencies of outgoing events on historical events in selections, the timestamps 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 distances are smaller than the relevance time span. This observation allows us to formalize Lemma 1.

Lemma 1. Given a $Q = \{G, fo, R, \delta, Pol\}$, a restriction window rw for each operator $\omega \in G$, and an interest switch from R^p_i to R^p_{i+1}. Let $\mathfrak{p}_{src}$ be a path in G from an operator or

source in the processing interest R^p_{i+1} to the consumer of G. If each operator or source ($src \in \omega \cup D$) produces an outgoing stream O for its successor in $\mathfrak{p}_{src}$ with an individual initialization point $\Delta_i(src, O)$ which yields

$$\Delta_i(src, O) \leq \text{start}(R^p_i) - \delta - \left(\sum_{\omega \in \mathfrak{p}_{src} \setminus src} \text{length}(rw(\omega)) \right)$$

then G will produce a maximum covering sequence of R^p_i.

Proof. Recall that each operator assigns to each outgoing event o the maximum timestamp of its selection s of incoming events. If the root operator ω_{root} produces an outgoing event with timestamp $\text{start}(R^p_i) - \delta$, it requires at most the incoming events within the restriction window, i.e., incoming events with greater timestamp than $\text{start}(R^p_i) - \delta - \text{length}(rw(\omega_{root}))$. All incoming events with smaller timestamps are, by definition, no longer relevant for the correlation. Thus, any operator ω_1 on the level below the root has to produce outgoing events, starting from $\text{start}(R^p_i) - \delta - \text{length}(rw(\omega_{root}))$. These operators additionally require at most the incoming events within the time span of the restriction window, hence, $\text{start}(R^p_i) - \delta - \text{length}(rw(\omega_{root})) - \text{length}(rw(\omega_1))$. Thus, on the path $p_{src} = (\omega_{root}, \omega_1, .., \omega_{max^s}, src)$ from ω_{root} to src it sums up to $\text{start}(R^p_i) - \delta - \text{length}(rw(\omega_{root})) - \ldots - \text{length}(rw(\omega_{max^s}))$. □

Our basic approach to find initialization points is therefore to sum up the restriction window lengths of the operators as a lower limit for the initialization points. In Section 3.3.4 we will discuss several approaches to determine an infimum on this lower limit. To determine the initialization point for an outgoing stream O at run time, our system maintains for each operator and processing interest the sum of lengths of the restriction window for each possible path $\mathfrak{p}$ through the operator graph to the consumer as additional time $\Delta_a(\mathfrak{p})$. The initialization point for O produced by $src \in \omega \cup D$ with a longest path $\mathfrak{p}_O$, from src to the consumer, is then simply calculated as $\Delta_i(R^p_{i+1}, O) = start(R^p_{i+1}) - \delta - \Delta_a(\mathfrak{p}_O)$. In the example of Figure 3.3, $\Delta_a(\omega_{<50}, \omega_{inc}) = 9 + 1$ and thus $\Delta_i(R^p_{i+1}, speed) = 10 - 10 = 0$.

Realizing the Strong Delivery Semantics

Note that we always use an individual initialization point for the event streams for each individual operator graph. Realizing the strong delivery semantics, however, would need to determine an exact point in time from which any operator graph would consistently initialize its event streams. It is difficult to find such points in time for an individual operator graph, and sometimes even not possible. The current reconfiguration technique needs to be extended, such that at all levels an initialization point is found where the consecutive correlations do not depend on previous correlation steps. For example, we have to make sure that none of the consecutive input events would have been consumed in a previous correlation step. Such an initialization point may not exist, e.g., when events in the overlap of two selection windows are consumed.

Therefore, to realize a strong delivery semantics, MCEP queries need to be coordinated. This can be achieved by (i) sharing the operator graph for two distinct queries with respect to the same processing interest R^p (see Chapter 5), (ii) fixing the additional time Δ_a and the temporal interest δ to a predefined value Δ_{fix} for all queries, and (iii) continuing to process the shared operator graph for R^p for a time $T_{fix} > \Delta_{fix}$ after an interest switch from R^p to another processing interest R^p_i occurred for any query. The time T_{fix} is large enough to ensure that two maximum covering sequences of two distinct queries do not overlap, otherwise consumers use the same operator graph that produces identical covering sequences.

3.2.5 Properties

We now discuss several properties of the Execution Environment and the Query Reconfigurator and the corresponding algorithms discussed in Section 3.2.3 and Section 3.2.4; especially temporal completeness, spatial consistency, and the subsequent implications on performance metrics.

Temporal Completeness and Spatial Consistency

The reconfiguration, performed after receiving the marker message, will produce a maximum covering sequence of R^p_{i+1} and ensures spatial consistency. The performed reconfiguration ensures that only atomic events from R^p_{i+1} are processed, because all events of previous ranges are consumed. Recall, that the timestamp mechanism always selects the greatest timestamp of the selection. Therefore, shifting the end of *rw* to the initialization point Δ_i of the succeeding operator makes sure that the first produced event e is the first possible event with $t(e) > \Delta_i$. Further recall, that the reconfiguration is delayed until the marker is selected by *rw*. Therefore, it is assured that all events e of the outgoing sequence with $t(e) \leq \text{end}(R^p_i)$ are produced, otherwise events with greater timestamps would be produced.

Comparison of Predeployment vs. MCEP

We analytically compare a predeployment strategy to a mobility driven reconfiguration of operators (for practical evaluation results see also Section 3.5). Let *np* be the number of predeployed operator graphs and let *nq* be the number of individually processed dynamic interest queries per unit-time. Further let the average mobility-rate, i.e., the rate at which the processing interests of an individual operator graph change, be denoted by ν_m. The average number of received events at each level is $\nu_e(l)$ and the average number of correlations at each level *lv* of the operator graph is $\nu_c(lv)$. The average overhead to update the operator graph is the sum of C_{ev}, for processing historical event stream s, and C_{up}, for additional update messages. The operator graph has an average-path length of ν_o. The cost for correlating with an operator graph is therefore $\sum_{lv=0}^{\nu_o-1} \nu_e(lv)\ \nu_c(lv)$. In

particular, the following inequation must hold for our approach to be better in terms of bandwidth and processing costs:

$$np \sum_{lv=0}^{\nu_o-1} \nu_e(lv)\ \nu_c(lv) > nq\ (\sum_{lv=0}^{\nu_o-1} \nu_e(lv)\ \nu_c(lv) + \nu_m\ (C_{ev} + C_{up})) \tag{3.2}$$

$$\Leftrightarrow (np - nq)\ (\sum_{lv=0}^{\nu_o-1} \nu_e(lv)\ \nu_c(lv)) > nq\ \nu_m\ (C_{ev} + C_{up}) \tag{3.3}$$

Obviously, if the update costs are low and the number of MCEP queries is low compared to the number of predeployed operator graphs we can save bandwidth and processing costs. Note, that if ν_δ is the average length of all temporal interests of the deployed queries and ν_Δ the average additional time, then $C_{ev} = (\nu_\delta + \nu_\Delta) \sum_{lv=0}^{\nu_o-1} \nu_e(lv)\ \nu_c(lv)$ for the not optimized reconfiguration. This clearly indicates that our approach benefits from small life-time parameters.

3.3 Reconfiguration Algorithms for Minimal Streaming and Processing Costs

In the previous section we discussed the reconfiguration of the operator graph with every location update of a focal object by streaming historical and live events to ensure temporal completeness. This approach, however, can result in a bottleneck, since a huge number of historical atomic events potentially need to be streamed to and processed by an operator graph G. For example, to detect traffic situations within the recent 30 min, several thousand location events of vehicles on the same road may need to be streamed and processed with each location update of the focal object [Ver12].

We propose to address this problem by exploiting characteristics of sources and atomic event streams that cause the detection of situational information in the current spatio-temporal interest. The first characteristic is the spatial distribution of atomic events, e.g., when a focal object moves several meters, the spatio-temporal interest will typically still capture an accident that happened within a perimeter of 1 km and depends on atomic events covered by that interest. The second characteristic is the heterogenity in the frequency of streamed atomic events for different event types. In the example of Figure 1.1, speed and location events are both needed to detect accidents. Hence, no processing has to be performed by an operator graph if only one of the two event types occurred in a spatial interest. Since vehicles rarely change their speed when driving on a highway, but constantly change their location, the frequency of speed events is lower than the occurrence frequency of location events and it is possible to only perform the accident detection when speed events occurred.

The contribution addressed in this section is a set of methods for optimizing the basic reconfiguration algorithm based on the spatio-temporal overlap of interests that we integrated in the proposed MCEP system. These methods save resources by reducing the number of streamed events between operators and the number of processed events

by operators. This section bases on previous work which is published in [KORR12] and [OKR+14a].

The rest of this section is structured as follows: Section 3.3.1 details the problem, Section 3.3.2 discusses conditions to dynamically reconfigure operator graphs, Section 3.3.3 discusses methods to reduce the number of duplicates in the system, and Section 3.3.4 discusses methods to approximate initialization points.

3.3.1 Problem Description

Our overall goal in the next sections is to establish methods to perform an optimized reconfiguration of the operator graph that reduces the overall streaming and processing costs. The supplemental goals in this context are (i) to preserve the introduced delivery semantics at least with a best effort and (ii) to preserve a high QoR—above the user defined threshold q_δ. To achieve these goals, we address the following three individual problems: *the optimal reconfiguration frequency problem*, *the redundancy-free streaming problem*, and *the optimal initialization point problem*.

To reduce the resource requirements of the system in streaming and processing historical events, a MCEP system can reduce the frequency of interest switches by ignoring location updates. These savings, however, come at the cost of a reduced accuracy with which a processing interest matches the actual spatial interest. By regulating how often a location monitor informs the query reconfigurator about location updates of a consumer, the MCEP system can decide to trade-off system resources against QoR for each individual operator graph. The *optimal reconfiguration frequency problem* is therefore to find a minimal interest switch frequency while preserving a high QoR.

Moreover, note that an interest switch between two subsequent processing interests R_i^p and R_{i+1}^p implies overlapping maximum covering sequences. In particular, a spatial overlap $R_i^p \cap R_{i+1}^p$ as well as a temporal overlap for (at least) events produced with respect to the temporal interest in the interval $[start(R_{i+1}^p) - \delta, start(R_{i+1}^p)]$. This can lead to *duplicates*, this means, the same event is streamed between operators multiple times, at least once with respect to R_i^p and once with respect to R_{i+1}^p. By omitting duplicates from streams between operators, which are hosted on different brokers, bandwidth can be saved. Moreover, by omitting correlation steps for an operator that lead to duplicates, computing resources can be saved. The *redundancy-free streaming problem* is therefore to efficiently identify duplicates in order to to minimize the number of duplicates in streams with a low overhead.

Furthermore, an interest switch from R_i^p to R_{i+1}^p requires to determine for each $\omega \in G$ the initialization points Δ_i of all incoming event streams $(in, \omega) \in G$ in order to produce a maximum covering sequence for each operator. However, the basic approach uses the restriction window to determine initialization points, which makes it possible that historical events can be processed and streamed unnecessarily. In Figure 3.3 the restriction window of ω_{inc} can include many events with timestamps smaller than $t = 5$, which can lead to an over-provisioning of historical situational information, i.e., situational

Algorithm 3.4 Timed and Spatial Update Condition

```
1: //Timed Reconfiguration Condition
2: Requires: Operator graph G, temporal threshold t_δ
3: upon receive Loction(Location l_u, Time t_u)
4:  if last_update + t_δ < t_u then
5:      reconfigure_operator_graph(l_u, t_u)
6:      last_update ← t_u
7:  end if
8: end
```

```
1: //Spatial Reconfiguration Condition
2: Requires: Operator graph G, spatial threshold l_Δ
3: upon receive Loction(Location l_u, Time t_u)
4:  if |last_update − l_u| > l_δ then
5:      reconfigure_operator_graph(l_u, t_u)
6:      last_update ← l_u
7:  end if
8: end
```

information with smaller timestamps than $t = 10$. By carefully adjusting the initialization points $\Delta_i(R_i^p)$ to a time $\Delta_i'(R_i^p)$ with $\text{start}(R_i^p) \geq \Delta_i'(R_i^p) > \Delta_i(R_i^p)$ completeness can be traded-off for less streaming and processing costs. The *optimal initialization point problem* is therefore to minimize the number of streamed events while ensuring completeness.

3.3.2 Operator Graph Reconfiguration Conditions

We approach the reconfiguration frequency problem by employing additional reconfiguration 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 individual operator graph.

1. The *basic reconfiguration condition* performs the operator graph reconfiguration as soon as the focal object 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 reconfiguration reconfiguration* initiates the operator graph reconfiguration after some predefined time t_δ has passed or when the consumer has moved a predefined distance l_δ.

3. A *QoR-aware reconfiguration* initiates the operator graph reconfiguration if the QoR calculated on the most recent atomic events in the spatial and processing interest is below the quality threshold q_δ given by the MCEP query.

Algorithms for Reconfiguration Conditions

We now focus on the algorithm for the QoR reconfiguration condition (see Algorithm 3.5), since the algorithms for the timed and spatial reconfiguration (see Algorithm 3.4) are straight forward implementations. Each of these algorithms is called when a location update l_u occurs at a time t_u and receives as an additional parameter a temporal, location,

Algorithm 3.5 Event-aware QoR reconfiguration condition

1: *Requires: Operator graph G*, quality threshold q_δ, historical time h_δ
2: *Defines:* $R^p \leftarrow R(G)$ //processing interest, initialized with spatial interest R_1
3: **upon** receive Loction(`Location` l_u, `Time` t_u)
4: $c_n \leftarrow |\{e|(e \in A(R(l))) \wedge (t_u - h_\delta \leq t(e) \leq t_u)\}|$ //Count recent atomic events
5: $c_p \leftarrow |\{e|(e \in A(R^p)) \wedge (t_u - h_\delta \leq t(e) \leq t_u)\}|$ //Recent processing interest
6: $c_o \leftarrow |\{e|(e \in A(R^p \cap R(l_u))) \wedge (t_u - h_\delta \leq t(e) \leq t_u)\}|$ //Atomic events in overlap
7: **if** $(\frac{c_o}{c_n} \leq \text{recall}(q_\delta)) \vee (\frac{c_o}{c_p} \leq \text{precision}(q_\delta))$ **then**
8: reconfigure_operator_graph(l_u,t_u)
9: $R^p \leftarrow R(l_u)$ //Buffer spatial interest as current processing interest
10: **end if**
11: **end**

or quality threshold that can be set by the system, e.g., according to the defined quality threshold.

The algorithm for the QoR reconfiguration condition initializes the reconfiguration of the operator graph based on an anticipated future QoR. The QoR is anticipated using recent historical atomic events, which benefits from the facts that (a) the most recent atomic event patterns often resemble future atomic event patterns (e.g., vehicles will drive along roads and provide sensed data with location stamps from those roads) and (b) those historical atomic events directly affect the QoR, since they need to be streamed in order to ensure completeness. To this end, our algorithm relies on a system parameter, a time span, h_δ which allows us to control the number of historical events selected for the anticipation of a future QoR. In particular, the algorithm counts at the time t_u of a location update (Lines 4-6) the number of recent historical atomic events in the time interval $[t_u - h_\delta, t_u]$, for the current processing interest (R^p), the most recent spatial interest ($R(l_u)$), and their overlap ($R^p \cap R(l_u)$). Based on the counts, precision and recall can be estimated and compared against corresponding thresholds (q_δ, Line 7) in order to determine if the operator graph needs to be adapted. In addition to monitoring the QoR with each location update, MCEP can optionally monitor the QoR on live data and depending on the monitored QoR initiate the reconfiguration.

Properties. Overall savings in streaming and processing costs are achieved if the cumulative savings for omitted operator graph reconfigurations outweigh the cumulative overhead for evaluating the update conditions. The processing overheads to evaluate the timed or spatial reconfiguration conditions for each location update are neglectable ($\mathcal{O}(1)$). Algorithm 3.5 (QoR reconfiguration condition) opts for a simple count on recent atomic events to ensure low streaming and processing costs. Counting recent events produces three results c_o, c_p, c_n and can be performed by the event storage without streaming any historical atomic events. Depending on the implementation of the event storage (e.g., [AN08]), the processing costs may differ, however, let A_c be the set of counted atomic events and c_f the amortized cost for fetching one event, it can be bound by $\mathcal{O}(|A_c| * c_f)$.

For all update conditions, the savings in processing and streaming costs depend on the set A_t of historical events that are not streamed after an interest switch is omitted. Streaming costs (in Landau notation) of $\omega(|A_t|)$ are saved, since not only the atomic events, but also events that depend on them are not streamed. Moreover, at least processing costs for finding selections in those atomic event streams are saved, which implies savings of $\omega(|S(A_t)|)$. The savings for all update conditions grow linear with the number of omitted interest switches.

Parametrization of Spatial Update Condition. We now analyze the relationship between the expected QoR and the distance between the spatial interest R and the actual processing interest R^p. This analysis considers circular spatial interests, an even distribution of events in all regions, and perfect location fixes from the GPS.

Let r_c be the radius of a circular interest and $\mathtt{d}_c$ be the distance between the centers of the circles of the spatial interest R and the processing interest R^p. We can determine the area $|R_o|$ of the lens that is formed by the overlap of both circles:

$$|R_o| = 2r_c^2 cos^{-1}(\frac{\mathtt{d}_c}{2r_c}) - \frac{\mathtt{d}_c}{2}\sqrt{4r_c^2 - \mathtt{d}_c^2}) \tag{3.4}$$

In this case precision and recall can easily be approximated as follows:

$$precision(R, R^p) = recall(R, R^p) = \frac{|R_o|}{|R^p|} = \frac{2r_c^2 cos^{-1}(\frac{\mathtt{d}_c}{2r_c}) - \frac{\mathtt{d}_c}{2}\sqrt{4r_c^2 - \mathtt{d}_c^2})}{\pi r_c^2} \tag{3.5}$$

Given the quality threshold and radius by the consumer, solving Equation 3.5 with respect to $\mathtt{d}_c$ would allow our system to parameterize l_δ of the spatial update condition (see Section 3.3.2). However, there is no closed form solution for Equation 3.5. Therefore, we approximate the lens formed by the circles (R^p, R) to a circle (R'_o) of the same area $|R_o|$. The circle R'_o's radius r'_c can then be determined using the quality threshold $q_\delta \in (0, 1]$:

$$r'_c = \sqrt{q_\delta * r_c^2} \tag{3.6}$$

Since the lens-shaped overlap is a subset of the area of the spatial interest we can deduct $|R^p| = |R| > |R_o|$ and thus the following inequation holds: $r_c > r'_c$. Note, also $2r'_c > |\mathtt{P}_S\mathtt{P}'_S|$ holds, since the diameter of a circle R'_o is larger than the segment $[\mathtt{P}_S\mathtt{P}'_S]$ of the line between the centers of the circles of R^p and R that is bounded by the pair of intersection points $\mathtt{P}_S, \mathtt{P}'_S$ with the lens. The distance $\mathtt{d}'_c$ between the centers of R and R^p is then approximated by:

$$\mathtt{d}_c > \mathtt{d}'_c = 2 * (r_c - r'_c) \tag{3.7}$$

The error $\mathtt{e}_c$ of this approximation is determined as follows using the area of R'_o ($|R'_o| = \pi r'^2_c$), Equation 3.5, and $\mathtt{d}_c = \mathtt{d}'_c\mathtt{e}_c$:

$$|R'_o| = \pi r'^2_c = 2r_c^2 cos^{-1}(\frac{\mathtt{e}_c\mathtt{d}'_c}{2r_c}) - \frac{\mathtt{e}_c\mathtt{d}'_c}{2}\sqrt{4r_c^2 - \mathtt{e}_c\mathtt{d}'^2_c} = |R_o| \tag{3.8}$$

Class	Example Operators	Condition
Non-locality- preserving	Aggregation (Sum, Count), Binary Operators (Concatenation or Alternation with consumption), ...	*Independence Condition* := $\forall E, E' : (s = s') \wedge s \in S(E) \wedge s' \in S(E') \Rightarrow f_\omega(s) = f_\omega(s')$
Locality-preserving	Filter (Values, Atoms), Projection, Vision (Face Detection, Face Recognition), ...	*Independence Condition* $\wedge$ $\forall E, E' : s \cap (E \cap E') \neq \emptyset \Rightarrow (s \in S(E) \wedge s \in S(E'))$

Table 3.2: Operator Classification for Duplicated Event Detection

In Equation 3.8 we can substitute r'_c by $r_c - \frac{\mathtt{d}'_c}{2}$ using Equation 3.7. Moreover, we can use the approximation $2r_c^2 cos^{-1}(\frac{\mathtt{e}_c\mathtt{d}'_c}{2r_c}) - \frac{\mathtt{e}_c\mathtt{d}'_c}{2}\sqrt{4r_c^2 - \mathtt{e}_c\mathtt{d}'^2_c}) < 2r_c^2 cos^{-1}(\frac{\mathtt{e}_c\mathtt{d}'_c}{2r_c})$ which utilizes that $2r_c > \mathtt{d}_c > 0$. Also using the approximation $(r_c^2 - 2r_c\frac{\mathtt{d}'_c}{2}) < (r_c^2 - 2r_c\frac{\mathtt{d}'_c}{2} + \frac{\mathtt{d}'^2_c}{4})$ leads to the following inequation:

$$\pi(r_c^2 - 2r_c\frac{\mathtt{d}'_c}{2}) < 2r_c^2 cos^{-1}(\frac{e\mathtt{d}'_c}{2r_c}) \tag{3.9}$$

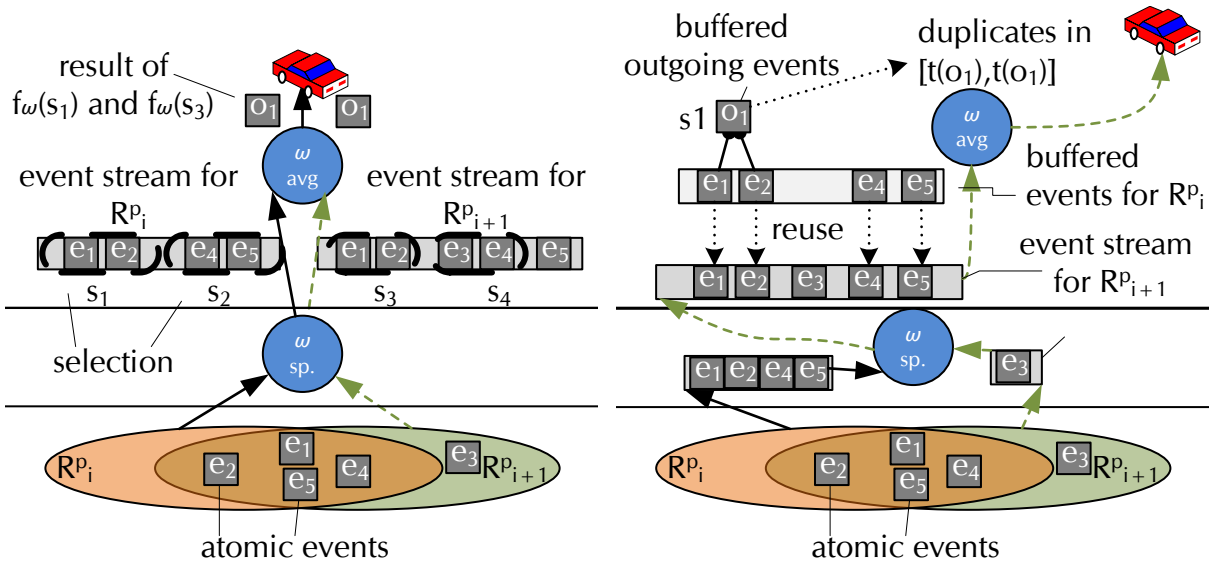
The error bounds for $\mathtt{e}_c$ are then determined by solving Equation 3.9 to:

$$-\frac{2r_c}{\mathtt{d}'_c} \leq \mathtt{e}_c \leq \frac{2r_c}{\mathtt{d}'_c} \text{ for } \mathtt{d}'_c \geq r_c > 0 \tag{3.10}$$

$$-\frac{2r_c}{\mathtt{d}'_c} \leq \mathtt{e}_c < \frac{2r_c \cos\frac{\pi r_c - \pi\mathtt{d}_c}{2r_c}}{\mathtt{d}'_c} \text{ for } r_c > \mathtt{d}'_c > 0 \tag{3.11}$$

3.3.3 Duplicated Event Detection

MCEP performs a duplicate detection on behalf of any operator ω of an operator graph G to deal with the problem of overlapping event sequences. There are two directions to approach the duplicate detection: (i) by finding duplicates of outgoing events in a buffered outgoing stream or (ii) by finding duplicates of selections that cause duplicated outgoing events in buffered incoming streams. The first approach, in comparison to an approach where no duplicates are detected, trades-off memory for buffering events against bandwidth for sending duplicated events. Observe, however, that this approach can only be applied after a duplicated correlation step has been performed; this wastes computing resources. In our example depicted in Figure 3.4a, a correlation function f_ω is applied to a selection s_1 with respect to R_i^p and a selection s_3 with respect to R_{i+1}^p. However, a pair-wise comparison with buffered events reveals that o'_1, which is detected by processing s_3, is a duplicate of o_1, which is detected by processing s_1. The second approach can avoid such unnecessary correlation steps and is therefore the chosen direction of MCEP. By



(a) Since operator ω_{avg} identifies duplicated selections s_1 and s_3, one with respect to R_i^p the other with respect to R_{i+1}^p, o_1 is detected for both processing interests.

(b) Atomic events are streamed only once to ω_{sp} and ω_{avg}, i.e., either with respect to R_i^p or R_{i+1}^p. By buffering incoming events for R_i^p, ω_{avg} can reuse them for R_{i+1}^p. By buffering the outgoing event o_1, ω_{avg} can indicate duplicates of o_1 to the consumer.

Figure 3.4: Example for duplicates when detecting events in two subsequent processing interests.

buffering selections like s_1, in addition to the outgoing events like o_1, our system can identify that a succeeding selection like s_3 is a duplicate of the buffered selection. The deterministic result of the buffered selection is then identified as duplicate as well. To buffer information about the selections detected in the event streams of R_i^p more memory is required than for the first approach.

In the remainder of this section we discuss operator classes which are distinguished by MCEP. Those classes allow our system to determine which operators ω of an operator graph G need to buffer event streams and selections. Moreover, we present class dependent approaches to interact with the buffers and thereby minimize the bandwidth that is required to stream historical events and reduce the number of processed selections to preserve computational resources.

Operator classes. MCEP distinguishes between *non-locality preserving* and *locality preserving* operators, which differ in the way their outgoing event streams are affected by selections comprising duplicates from incoming event streams. For operators of the first class only some selections comprising duplicated ingoing events will produce duplicated outgoing events. Operators of the second class ensure that all selections comprising

duplicates will produce a sequence of duplicated outgoing events. Table 3.2 gives for each of these classes example operators and formal conditions that are fulfilled by its members which are detailed in the following paragraphs.

For any pair of incoming event sequences E and E', processing equal selections ($s \in S(E) = s' \in S(E')$) at non-locality preserving operators results in detecting the same sequence of outgoing events. As an example, consider the operator ω_{avg} of Figure 3.4a, which is used to detect the average speed of speeding vehicles by processing speed events reporting such vehicles. Recall, equal selections ($s_1 = \{e_1, e_2\}, s_3 = \{e_1, e_2\}$) are detected with respect to events in the incoming event sequence E for a processing interest R_i^p and E' for R_{i+1}^p which each result in the same outgoing event o_1. Yet, not all events detected with respect to $E \cap E'$ are necessarily also detected with respect to E'. One historical event e_3 is integrated during the interest switch and due to the spatio-temporal event order the selections determined for the new incoming event sequence change. Although E' comprises all events of $s_2 = \{e_4, e_5\} \in S(E)$ a different selection $s_4 = \{e_3, e_4\} \in S(E')$ is processed instead.

For any pair of incoming event sequences E and E' of locality preserving operators, all selections s comprising events from the overlap of E and E' will be determined for both event sequences. An example is operator ω_s, depicted in Figure 3.4a, which detects if speed events reported by vehicles in a processing interest indicate speeding, e.g., ω_s is a filter who detects events carrying speeds above 50 km/h. Since filter operators process an event sequence event by event at most one event is comprised in any selection. This leads to identical selections comprising speed events with locations in $R_i^p \cap R_{i+1}^p$ for ω_s's incoming event sequence with respect to R_i^p and R_{i+1}^p. Moreover, it also leads to duplicated outgoing events, e.g., e_1, e_2, e_4, e_5 are detected for both processing interests in the example.

Other operator classes, e.g., *context-sensitive operators*, do not only rely on the selection to produce events, but also on additional context, which means results cannot be reused without extensive knowledge about the operator's semantics. Since we focus on a general purpose MCEP middleware, we omit the latter class from any further discussions. Remark, however, by not applying one of the following approaches MCEP allows programmers to include operator specific solutions like [XJ07] or [XECA07].

Optimizations using the Spatial Overlap: The duplicate detection approach for locality-preserving operators ω_l makes use of our previous observation that processing events with locations in $R_o^p \subseteq R_i^p \cap R_{i+1}^p$ multiple times always leads to the same outgoing events. In fact, this means locality-preserving operators can avoid duplicates by not processing events from R_o^p, e.g., in Figure 3.4b e_1, e_2, e_4, e_5 do not need to be processed by ω_s with respect to R_{i+1}^p. In particular, preceding operators filter out all historical events with locations in R_o^p before streaming them to a locality-preserving operator. In our example e_1, e_2, e_3, e_4 are filtered out by the range query for R_{i+1}^p. This allows locality-preserving operators to neither keep incoming nor outgoing buffers. To access duplicates if needed, succeeding non-locality preserving operators like ω_{avg}, buffer the incoming stream from

Algorithm 3.6 Optimizations using temporal reuse

Requires: `Operator` ω
  R_i^p, R_{i+1}^p //previous and current processing interest
Defines: $B_L, B_I, B_O \leftarrow \emptyset$ //Buffers for logs, incoming, and outgoing streams
  $log \leftarrow \emptyset$ //current log sent to successors of ω in the operator graph
upon receive event(*Event* e)
 if e is log **then**
  $B_L(R_{i+1}^p) \leftarrow B_L(R_{i+1}^p) \cup \{e\}$ //add log to corresponding buffer
  for all $e' \in B_I(R_i^p) : t(e') \in [t_{\min^l}(e), t_{\max^l}(e)]$ **do**
   $B_I(R_{i+1}^p) \leftarrow B_I(R_{i+1}^p) \cup \{e'\}$ //fetch events indicated by log from buffer
  end for
 else
  $B_I(R_{i+1}^p) \leftarrow B_I(R_{i+1}^p) \cup \{e\}$ //add new events to current buffer
 end if
end
upon open selection(*Selection* s)
 if $\exists [t_b, t_f] \in B_L(R_{i+1}^p) : t_s(s) \in [t_b, t_f] \wedge t_e(s) \in [t_b, t_f]$
  $\wedge \exists bu \in B_O(R_i^p) : t_s(bu) \in [t_b, t_f] \wedge t_e(bu) \in [t_b, t_f]$ **then**
  update log with $\{o_1, \ldots\}$ from bu //reuse output events
  $B_O(R_{i+1}^p) \leftarrow bu$ //ensure reuse of results for R_{i+1}^p
 end if
end
upon send outgoing event(*Set_of_Event* E_O, *Selection* s)
 send log to all successors of ω
 $log \leftarrow \emptyset$ //clear log
 $B_O(R_{i+1}^p) \leftarrow B_O(R_{i+1}^p) \cup \{E_O, \text{first}(s), \text{last}(s)\}$ //buffer outgoing tuple
end

ω_{sp} with respect to R_i^p. After an interest switch to R_{i+1}^p operator ω_{avg} can automatically retrieve omitted events between subsequent events e, e' received from ω_{sp} by looking up buffered events with locations in R_o^p and a timestamps in the interval $(t(e), t(e'))$. For example, in Figure 3.4b e_1 and e_2 can be fetched from a buffer when e_3 arrives at ω_{avg}.

Optimizations using the Temporal Overlap: The duplicate detection approach for non-locality preserving operators $\omega \in G$ makes use of the temporal overlap of its produced event sequence E' for R_i^p and the event sequence E'' produced for R_{i+1}^p. Instead of streaming all events, ω injects *logs* which inform a succeeding operator $\omega' \in G$ about omitted events. In particular, logs are represented by time spans. For example, operator ω_{avg} of our example indicates with a log that all events between the timestamps $t(o_1)$ and $t(o_1)$ are duplicates of the event sequence with respect to R_i^p, i.e., o_1 can be reused by the consumer. Alternatively, MCEP can also support negative logs, which indicate the

eviction of events from buffered streams E' for R_i^p. It is highly depending on the scenario how many logs or negative logs are streamed from ω to ω', which is why we allow the programmer to choose between the two methods. Since both approaches are very similar we focus on the log approach in the remainder. Moreover, for the sake of simplicity, we refrain from any discussion about the impact of consumption policies and asynchronous processing; the algorithm's extensions to support consumption policies and asynchronous processing are rather straight forward.

To utilize and build up logs (*log*) our system keeps three additional buffers for each processing interest R_i^p (see Algorithm 3.6). First, $B_I(R_i^p)$ for all incoming events from streams with respect to R_i^p, which can be used to reuse these events with respect to successive processing interests R_{i+1}^p. Second, $B_L(R_i^p)$ for temporally ordered logs from preceding operators in the operator graph, which allows us to verify that an event has been reused from a buffer. Third, $B_O(R_i^p)$ for temporally ordered outgoing tuples to identify duplicated outgoing events for a distinct selection s. Outgoing tuples are of the form $\{\{o_1, \ldots\}, t_s(s), t_e(s)\}$, where $\{o_1, \ldots\}$ is a set of outgoing events produced with respect to the same selection s, $t_s(s)$ is the first timestamp of events comprised in selection s, and $t_s(s)$ is the last timestamp of events comprised in selection s (see Line 2). Figure 3.4b depicts this buffer for s_1 as $\{\{o_1\}, t_s(e_1), t_e(e_2)\}$.

When a log is read from the incoming stream it is immediately added to $B_L(R_{i+1}^p)$ (see Line 7). Furthermore, events indicated by this log (e.g., o_1) are buffered in the buffer for incoming events $B_I(R_{i+1}^p)$ alongside actually streamed events for R_{i+1}^p, which enables the operator to process them (see Line 9/Line 12). Note, if the preceding operator has been locality-preserving, the log is created on the fly when filling the gaps between subsequently received events. For example, by fetching e_1 and e_2 from the buffer a corresponding log $[t(e_1), t(e_2)]$ can be buffered in B_L. After opening a selection, say s, and adding all events, e.g., e_1 and e_2 to s_3 in the example, our algorithm checks if s is a duplicate (see Line 16). For example, since $t(e_1) \in [t(e_1), t(e_2)]$ and $t(e_2) \in [t(e_1), t(e_2)]$ are captured by a time interval of a log, say $[t_s, t_f] \in B_L(R_{i+1}^p)$, we ensure that all events of the selection in our example have been reused; in particular, no further event e' has been streamed with a timestamp $t(e_1) < t(e') < t(e_2)$. Moreover, a duplicated outgoing tuple is found if the first and last timestamp of the selection match the timestamps of the tuple. A log is sent to all successors when a new event is detected (see Line 22) or, to keep low latencies, after a predefined time has passed since the first element has been added to the log.

3.3.4 Approximation of Initialization Points

In this section we show how to (i) efficiently determine initialization points of an event stream even if operators are hosted at different brokers, and (ii) keep the overhead of an operator's reconfiguration low by proposing several optimizations that allow us to minimize the amount of unnecessarily processed and streamed events to reduce the cost of a reconfiguration.

Approach Overview

We approach the initialization point determination problem by (i) also utilizing the selection window definitions beyond the restriction window to determine the initialization point of the atomic event stream *ae*, (ii) only reconfiguring operators if their incoming streams changed, and (iii) further relaxing temporal completeness—situational information at the beginning of the stream is only delivered with best effort.

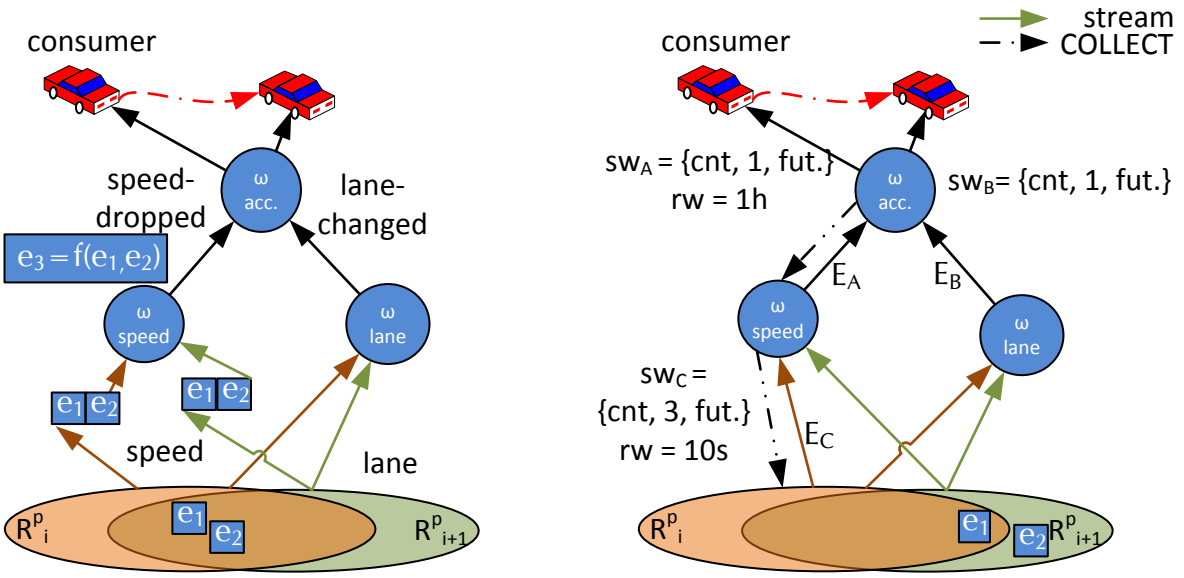
The core idea is to keep an initialization point for each individual incoming stream of an operator based on its selection window *sw*. The MCEP system can then determine the initialization points in the event stream I similar to the basic approach (see Section 3.2.4) by determining an *additional time* $\Delta_a(\mathfrak{p}_I)$ for each path $\mathfrak{p}_I$ from the event stream to the consumer. We therefore have to first determine a selection window-dependent time $\Delta_a(sw)$, with $\Delta_a(sw) \leq (end(rw_\omega) - start(rw_\omega))$, for each individual operator ω on the path. The additional time is then determined as $\Delta(\mathfrak{p}_I) = \sum_{\omega \in p_I} \Delta_a(sw)$.

In addition, observe in the accident detection example depicted Figure 3.5a that after the interest switch operator ω_{speed}'s produced sequence of events with respect to the corresponding temporal overlap does not change; both times it comprises the detected historical event (e_3). This results from the corresponding *dependent atomic event sequence* of speed events, i.e., (e_1, e_2), which does not change since all sources only produced events with locations in the spatial overlap of the subsequent processing interest. This observation leads to Lemma 2, which states that not all operators of G need to be reconfigured when an interest switch occurs. In the example, we can decrease the overall reconfiguration costs for ω_{acc} and the *induced sub-graph* $G(\omega_{speed})$, the sub-graph which comprises all operators and sources that can reach ω_{speed} in G, as follows: Operator ω_a does not need to process a marker $\mathbb{M}$ and the events e_1 and e_2; ω_{speed} does not even need to send logs to perform the optimization described in Section 3.3.3.

Lemma 2. Given a $Q = \{G, fo, R, \delta, Pol\}$, an interest switch from R_i^p to R_{i+1}^p, and initialization points Δ_i which are determined using the basic approach. If for an induced sub-graph $G(\omega)$ of G there is no dependent source *src* with a location $l_s \in (R_{i+1}^p \setminus R_i^p) \cup (R_i^p \setminus R_{i+1}^p)$ that produced (historical) events in $[\Delta_i(R_{i+1}^p, src), \text{start}(R_{i+1}^p)]$, then the sub-graph does not require an update of its operator's states to produce a maximum covering sequence of R_{i+1}^p.

Proof. We proof our claim by contradiction. Assume, that the produced sequence $E_\omega = [\Delta_i(R_i^p, \omega), \text{start}(R_{i+1})]$ is not a covering sequence for $E'_\omega = [\Delta_i(R_{i+1}^p, \omega), \text{start}(R_{i+1})]$ and thus no maximum covering sequence is produced by G for R_{i+1}^p. This means that $E'_\omega \not\subseteq E_\omega$. Observe, both event sequences can be produced by streaming all historical atomic events A_{src} for each source $src \in D(G(\omega))$ that occurred in R_{i+1}^p. Since each operator $\omega' \in G(\omega)$ applies its correlation function on the same sequence of selections S, they produce the same sequence of outgoing events $O_{\omega'}$ (see Table 3.2). In particular, ω produces O_ω with $E_\omega \subseteq O_\omega$ and $E'_\omega \subseteq O_\omega$. Moreover, since $\Delta_i(R_i^p, \omega) < \Delta_i(R_{i+1}^p, \omega)$ also $E'_\omega \subseteq E_\omega$. □

Note that the additional times which determine initialization points can depend on the state of operators in G as well as the properties of the incoming event streams. Many



(a) Speed events are selected as input to the operator graph and result in the detection of e_3 for both processing interests R_i^p and R_{i+1}^p.

(b) Speed events are selected as input to the operator graph. Since new atomic events are selected by R_{i+1}^p, the state of operators like ω_{speed} changes.

Figure 3.5: Extended accident scenario:In (a) the sequence of atomic events selected is the same for both, the current and subsequent processing interest. A reconfiguration of the operator's states is not required. In (b) new speed events are selected for R_{i+1}^p but no new lane events. As a result, operators in the sub-graph comprising ω_{speed} and ω_{acc} need an update of their states after an interest switch, but not ω_{lane}. To ensure temporal consistency and completeness after an interest switch, each operator needs to be initialized with historical events that fill up its (counting) selection windows sw_A,sw_B,sw_C. The additional time Δ_a, which covers the timestamps of these events, is collected in a top down direction.

operators, like ω_{acc} of the example depicted in Figure 3.5b, need events from two or more incoming streams to produce an outgoing event: for an accident event, ω_{acc} requires that both a lane change event and a slowdown event are present in its respective input stream. In the example, no steering event is streamed to ω_{lane} with a timestamp in the temporal overlap of the processing interests R_i^p and R_{i+1}^p. Thus, no event about vehicles changing lanes will be streamed from ω_{lane} to ω_{acc}. As a result, our system can save resources by not streaming and processing historical events with timestamps in said temporal overlap in the sub-graph $G(\omega_{acc})$. In particular, neither e_1 is streamed to ω_{speed} by our system, nor e_3 is detected and streamed from ω_{speed} to ω_{acc}.

Algorithm 3.7 Basic Reconfiguration Algorithm

1: *Requires: receiver* //either operator $\omega \in G$ or query reconfigurator of query Q
2: **upon** receive *COLLECT*(`Id` id_R, `AdditionalTime` Δ_a, `Sources` *Src*, `Time` *t*)
3: **if** *receiver* is operator **then**
4: $Pred \leftarrow \{pred | pred \in \text{predecessors}(receiver) \wedge \exists src \in Src : src \leq pred\}$
5: **for** $\forall pred \in Pred$ **do**
6: $\Delta_\omega \leftarrow$ refine_additional_ime(Δ_a,*t*,*pred*)
7: **trigger** *send COLLECT*(id_R, Δ_ω, $\{src | src \in Src \ \wedge \ src \leq pred\}$, *t*) to *pred*
8: **end for**
9: **else**
10: $\forall src \in Src$: **trigger** initializationPointFound($R^p(id_R)$, *t*-Δ_a) **at** Q
11: **end if**
12: **end**

13: **upon** find_initialization_points(`Query` Q, `Interest` $R^p = (R^p_{i+1}, R^p_i)$, *Time t*)
14: $Src \leftarrow \{src | \exists e : (e \text{ produced by } src) \wedge (l(e) \in (R^p_{i+1} \setminus R^p_i) \cup (R^p_i \setminus R^p_{i+1}))$
$\wedge (t(e) \in [start(R^p_{i+1}) - \Delta_{max}(Q), start(R^p_{i+1})])\}$)
15: $\forall r \in \text{roots}(G)$: **trigger** *send COLLECT*(id(R^p_{i+1}), $\delta(Q)$, *Src*, *t*) **to** *r*
16: **end**

Distributed Reconfiguration Algorithms

We now discuss two distributed algorithms, which allow our system to determine the initialization points $\Delta_i(src)$ by adding up additional times on each path $\mathfrak{p}_{src}$ from an atomic event stream to the consumer. We first clarify the basic principles by introducing the pull-based algorithm (see Algorithm 3.7), which re-determines the initialization points with each interest switch, thus reflecting recent changes in the additional times $\Delta_a(p)$ of a path p when they are needed. We then discuss a less restrictive push-based algorithm, which only updates the additional times $\Delta_a(p)$ if, based on the operators states, they are expected to change.

Pull-based Algorithm to Determine Initialization Points. When an interest switch is triggered and the initialization points need to be determined by a query reconfigurator, the system first collects the sources that require reconfiguration by querying the event storage using the spatio-temporal condition of Lemma 2 and the maximal additional time Δ_{max} (see Algorithm 3.7, Line 14); the latter is found using the basic approach to determine initialization points. This information is then forwarded to the root operator of G, which can determine from the collected set of sources the induced sub-graphs of G which require a reconfiguration. For the example depicted in Figure 3.5b a message is sent to ω_{acc} that only speed events (i.e., e_2) are newly selected. The root initiates the initialization point determination for the sub-graphs by sending a *COLLECT* message to all its preceding operators that are member of a induced sub-graph that requires reconfiguration. In the

example, a *COLLECT* message is sent from ω_{acc} to ω_{speed} and comprises information to identify the initialization points of the outgoing event streams $(\omega_{speed}, \omega_{acc})$ with respect to the processing interest R^p_{i+1}. In particular, it comprises information about the set of sources that require a reconfiguration (e.g., sources of speed events), the additional time Δ_a determined by its successor (e.g., $\Delta_a = 1\,\text{h}$), the time $t = start(R^p_{i+1})$, and an identifier for the processing interest id_R. Each operator receiving a *COLLECT* message[1] will refine the additional time and sends *COLLECT* messages to all predecessors that are member of an induced sub-graph that requires reconfiguration (Line 6). Operator ω_{speed} receives the *COLLECT* message, refines the additional time, and sends it to the responsible query reconfigurator. Finally, the responsible query reconfigurators uses the time $start(R^p_{i+1})$ and additional time Δ_a to initialize the relevant atomic event streams of the sources (Line 10) by sending marker messages, atomic event streams, and live event streams (see Section 3.2.4).

Note, this algorithm allows for another optimization, which is omitted for brevity from Algorithm 3.7: sub-graphs do not need to process a marker messages if they are not reconfigured. This optimization forces our system to attatch information about recofnigured sources to the actually disseminated marker messages. This way, operators $\omega \in G$ can be reconfigured when marker messages from all incoming streams that depend on reconfigured sources have arrived.

Additional Time Estimation. Now we detail how the operators perform the estimation of the additional times of incoming streams from additional times of their outgoing streams. To this end, we discuss three strategies to determine and represent the additional time Δ_a.

Time-span strategy: An initial additional time is set to the temporal interest $\Delta_a = \delta$. Consider now a pair of operators ω_1, ω_2 from G where ω_1 needs to send a *COLLECT* message to ω_2. At this time ω_1 has an estimate of the end point $end(rw_{\omega_1}) = t - \Delta_a$ of its restriction window rw that will ensure a maximum covering event sequence of G. From $end(rw_{\omega_1})$, ω_1 can determine for each of its incoming streams the beginning $start(sw)$ of a corresponding selection window sw through an increase of the additional time. For temporal windows, this typically relates to their fixed time span added to Δ_a. For counting windows the system maps the count to an estimated time span. Therefore, the system monitors the time between the inter-arrival times of recent events. To account for the dependencies of sw on several other selection windows sw' through sw^n with $sw <_\kappa sw' \ldots <_\kappa sw^n$ the system increases the additional time Δ_a by $\Delta_a(sw) + \Delta_a(sw') + \ldots + \Delta_a(sw^n)$. In particular, ω_{acc} of the example can add $\Delta_a(sw_A)$ for sw_A, plus the estimated time $\Delta_a(sw_B)$ for the dependent window sw_B to $\Delta_a = \delta$. Note, when selection windows are initialized at the beginning of the restriction window or are shifted to the past, our system anticipates the additional time by monitoring their recent maximum differences between the first event in the selection window and the end of the restriction window.

[1]Note, when more than one successor of $\omega \in G$ exist, ω waits until a *COLLECT* messages arrived from each of them before triggering Algorithm 3.7 with the message that comprises the largest Δ.

Count-based strategy: In some scenarios counting windows are better reflected as a count and not a time span. For example, when selecting exactly 5 speed events with highly varying inter arrival rates from vehicles to determine an average speed. To this end, we allow the programmer of the operator graph to select a second strategy where the additional time is represented as an expression. The expression comprises constant time spans ($T : t_{const}$), variable timestamps ($S : t_{var}$) and numbers ($N : n$)[2] reflecting the different window types supported by the proposed window model. For instance, initially the root of G will receive the expression $(S : \delta)$, representing the temporal interest. Similar to the first approach, each temporal window reflects a time span, yet, each counting window adds a number to the expression. In particular, ω_{acc} can add $(N : 1)$ for sw_A, plus the estimated time span from the depending sw_B $(S : \Delta_a(sw_B))$ and ω_{speed} adds $(N : 3)$, which indicates that the query reconfigurator selects the 4 most recents events before $t - \delta - \Delta_a(sw_B)$.

State-awareness: Our last optimization accounts for situations where operators need events from two or more incoming streams according to the processing predicates, like ω_{acc}, but no events occur in at least one incoming stream like in Figure 3.5b. Let SW be the set of selection windows that can be determined on streams which are not reconfigured, e.g., on stream B of ω_{acc}. This set can be used to adjust Δ_a for sw_A to $\min(\Delta_a, t - start(SW))$. In the example Δ_a can be set to 0 since no event from stream B occurs.

Push-based Algorithm to Determine Initialization Points. Observe that, depending on the scenario, inter-arrival rates may not change often, operators are not selective, partial reconfigurations of sub-graphs are seldom viable, or operators simply use temporal windows, and thus the initialization point Δ_i does not change with every interest switch. In this case, the initialization point is determined once with the pull-based algorithm and buffered afterwards at the query reconfigurator. Operators then monitor key metrics, in particular the inter-arrival rates of events, and push down changes of the initialization point.

Properties. 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 can be missed.

[2]Notice that each operator can refine such a number by an estimate on how many events are required from its predecessors to produce n events.

3.4 Prediction-based Algorithms for Minimal Latency

This section covers the crucial topic of timeliness when detecting situational information. For example, there is no point in notifying a 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. To this end, we tackle the most significant processing bottleneck that occurs when detecting situational information with dynamic interest queries as discussed in Section 3.2: the reactive reconfiguration of operator graphs after a switch to a processing interest.

Recall, with each interest switch processing of historical atomic events is necessary before processing of live events can begin. A potentially large number of historical events must be processed in temporal order to ensure the delivery semantics, so there is a processing delay before live event processing begins. Therefore, if the operator graph is reactively reconfigured for a processing interest when a focal object moves to the corresponding location, the focal object can be in a different location by the time the operator graph has finished processing historical events.This can lead to a situation where the operator graph is constantly trying to "catch up" to the location of the focal object, resulting in processing events in inappropriate locations and never actually getting to process any relevant "live" events.

We propose to address this problem by processing the historical events for a processing interest before the mobile focal object arrives at the corresponding location, so that live event processing begins at the moment the user arrives, if not before. This can be done when two things are available: future location predictions for the mobile focal object and processing time estimations. Several location prediction algorithms exist and profiling techniques can be used to estimate processing time [WSCY99, ZPG+10]. 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, typically the location prediction does not give a single, accurate location. To mitigate the imperfection of location prediction algorithms, we propose taking several predictions for each future time step and opportunistically compute the events for all of those locations. When the actual interest switch is triggered, the prediction among those that is closest to the focal object's actual location will be selected and its situational information returned. Second, if the speed of the focal object 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 a pipelined processing of future locations in several future time steps. To handle the increased event load when processing for many future locations of a focal object, we rely on parallel resources, i.e., a cluster of fog nodes which can be accessed with nearly no network latency.

Our research contributions in this section base on previously published work ([HLR+13b, OKR+14a]) and comprise methods for (a) starting event processing at predicted future locations in advance of a focal object's arrival, (b) pipelining multiple future prediction points to enhance the completion of event processing by the time the focal

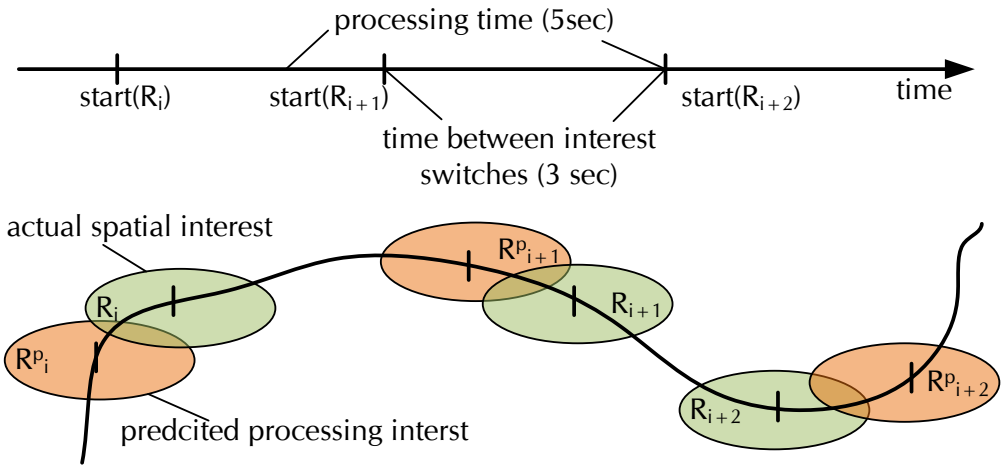


Figure 3.6: Examples for predictive query processing: operator graphs do not match with predicted locations and may require more time to process all required historical events to produce a maximum covering sequence than it takes to switch from one processing interest to the next

object arrives.

The rest of this section is structured as follows: We first detail the problem of processing latency for historical events when reconfiguring an operator graph (Section 3.4.1), then we detail the methods to minimize the latency using predictive query processing (Section 3.4.2).

3.4.1 Problem Description

To achieve minimal latency, an operator graph must have processed all historical events with respect to a processing interest R^p_{i+1} when an interest switch from the previous processing interest R^p_i to R^p_{i+1} occurs. In the ideal case, there is no historical event for the new processing interest and therefore the processing latency is for historical events zero. However, if the processing interest R^p_{i+1} contains historical atomic events matching to the consumer's temporal interest, the processing of events with respect to R^p_{i+1} has to start earlier, before the interest switches, giving enough time to process all historical events.

To start streaming and processing events with respect to R^p_{i+1} earlier, we explore the potential of opportunistic computing based on the focal object's future locations. Since mobile users tend to move according to a pattern, e.g., the focal object can drive along a road, we expect that future locations can be anticipated using well-known location prediction techniques. Using the predicted locations, copies of the operator graph can process events for processing interests with respect to future location before a focal object reaches one of the future locations.

The timeliness and QoR of the resulting situational information detected by the predicted operator graphs highly depends on when and where an operator graph is initialized. Since processing of historical events takes time, it is also possible that the focal object moves faster to future locations than the operator graph requires to process all historic events. Consider Figure 3.6, the temporal axis shows that a focal object updates its spatial interest within 3 s from R_1 to R_2 while the processing takes 5 s. Moreover, since a discrete future location is subject to high location uncertainties, i.e., the prediction is not a perfect oracle, it is highly probable that the spatial interest does not match with the predicted processing interest. Consider in Figure 3.6 the predicted processing interest R_1^p deviates from the actual spatial interest R_1 and the circles that indicate those interests overlap.

To increase the chance of detecting situational information with respect to a predicted processing interest that provides a suitable QoR, multiple operator graphs that simultaneously process atomic events selected by slightly different processing interests must be deployed as early as possible. While continuously processing events with operator graphs for an infinite number of predicted future processing interests would result in processing enough operator graphs to also comprise the one that actually matches the spatial interest, this is not practically feasible. An insurmountable amount of resources would be needed to process all those operator graphs. As part of our solution, we deploy a discrete number of future processing interests around the predicted future location of the focal object at a reasonable time before the focal object arrives at one of the future locations.

The timely delivery problem addressed in this section is to reduce the perceived latency by a consumer when receiving historical situational information with respect to the spatial interest R_i while preserving quality and resource constraints. In particular, the following constraints regarding the consumer's QoR threshold q_δ and a system threshold $\rho_{\max}$ for the overall resource usage ρ are preserved:

1. $\mathrm{QoR}(R_i, R_i^p) \geq q_\delta$
2. $\rho < \rho_{\max}$

3.4.2 Predictive Query Processing

MCEP addresses the timely delivery problem by buffering situational information that stems from opportunistically deployed copies of operator graphs at predicted future locations in the event storage. This way, buffered situational information can be streamed from the event storage to the consumer when she actually switches to the corresponding processing interest without an additional processing delay before "live" situational information is directly streamed. To this end, we now focus on an algorithm for the query predictor and the changes required by the query reconfigurator that supports the processing of operator graphs at predicted locations. To explain the basic principles of the algorithm, we first describe how the system uses a predicted future location of a focal object to configure an operator graph in advance (see Algorithm 3.8). We then address the corresponding changes to the query reconfigurator (see Algorithm 3.9). To avoid

Algorithm 3.8 Basic Query Prediction

1: *Requires:* MCEP Query Q
2: *Defines:* Operator Graph $G_{curr} \leftarrow$ *initial_operator_graph* // Current operator graph
Operator Graph $G_{next} \leftarrow \emptyset$ // Predicted operator graph

3: **upon** receive Location(`Location` l_u, `Time` t_u)
4: **if** update_condition(l_u,t_u) **then**
5: $G_{curr} \leftarrow$ reconfigure_predicted_operator_graph($t_u, l_u, G_{next}, G_{curr}$) //call to query reconfigurator
6: $t_{next} \leftarrow$ next_predicted_update(t_u) //predict next time of reconfiguration
7: $l_{next} \leftarrow$ next_predicted_location(l_u, t_{next}) //predict future location of focal object
8: $G_{next} \leftarrow$ determine_interest(l_{next}) //generate copies of operator graph
9: **trigger** start_streaming($R^p(G_{next})$,$t_{next} - \Delta_a$,$D(Q)$, G_{next}, Q) **at** ($t_{next} - T_C(l_{next})$)
10: **end if**
11: **end**

the problem of late processing in the face of fast movements, we extend that algorithm by pipelining future operator graphs for not only the next predicted location, but also for more subsequent locations (see Algorithm 3.10). We then extend the mechanism to trade-off computing resource usage against QoR. To this end, the system anticipates a minimal set of predicted locations whose processing interests provide a coverage that is expected to provide good QoR and preserve the resource constraints for all possible future locations.

Predicting Operator Graphs

We now detail an algorithm to determine and deploy a future operator graph at a predicted location in order to start processing early. An initial operator graph is deployed when the consumer initiates her query Q with the focal object's initial location (*initial_operator_graph*). Henceforth, Algorithm 3.8 keeps track of the operator graph G_{curr} that currently processes events with respect to the focal object's current processing interest and a future operator graph G_{next} that already processes events for the next predicted future processing interest (Line 2).

With each location update triggered from the location monitor, the update conditions discussed in Section 3.3.2 are checked. If the currently selected reconfiguration condition is triggered, then the switch to a new processing interest is initialized by replacing the current operator graph with the future operator graph (Line 5). For example, the operator graph deployed for the processing interest R_i^p depicted in Figure 3.6 is replaced with the operator graph deployed for R_{i+1}^p. Then the query prediction is initialized: our system predicts the future location of the focal object at the next discrete time when the update condition is expected to evaluate to true (Line 7).[3] In the example, the location which

[3] We assume for the sake of argument a temporal update condition. However, for other update conditions the recent frequency of reconfigurations can be used to determine the time of the next reconfiguration.

Algorithm 3.9 Extended Query Reconfigurator

```
1: Requires: MCEP Query Q
2: function reconfigure_predicted_operator_graph( Time t_u, Location l_u, Operator Graph G_next,
   Operator Graph G_curr)
3:   if G_new = ∅ ∨ complete(G_next, t_u) ∨ (QoR(R(G_new), R(l_u)) ≤ q_δ(Q)) then//ensure quality
4:       G_next ← reconfigure_operator_graph() //call to basic Query Reconfigurator
5:   else
6:       stop_query_after(Q, R(Q),t_u(l_u)) //Stop range query and situational information
7:       ∀src ∈ D(G_curr) : send Marker(t, D(G_curr)) to successor(src)
8:       discard(G_curr) //free resources
9:       R(Q), G(Q) ← R(G_next), G_next
10:  end if
11:  return G_next
12: end

13: upon receive Marker(Time t, Set of Sources D) from G
14:  deliver_historical_information(G, D) //deliver historical situational information
15:  initiate_live_notification(G, D) //process live situational information
16: end
```

spans R^p_{i+2} is predicted when R^p_i switches to R^p_{i+1}. The location monitor thus implements an additional location predictor which returns a future location of the focal object, e.g., by using a linear dead reckoning algorithm [LR01a] to predict a future location. Our system deploys a copy of the operator graph and a range query for each of the future processing interest with respect to the newly predicted location (Line 8). The start of streaming for the future processing interest is scheduled with respect to the initialization point of the future processing interests at some time T_C before the actual interest switch is expected to occur (Line 9). For now, we assume that the system parameter T_C is fixed, however, we will discuss that this parameter can be anticipated by our system based on the expected workload for the future processing interest.

Predictive Query Reconfigurator. Few changes have to be made to the query reconfigurator (in contrast to Section 3.2.4) to deal with future processing interests (see Algorithm 3.9) and to provide a consumer with buffered historical situational information. As a first step after the reconfiguration is triggered, e.g., by Algorithm 3.8, the algorithm checks if completeness requirements are ensured and the future processing interest is expected to maintain a QoR above the consumer defined threshold. In this case, the system switches to this operator graph instead of reconfiguring the current operator graph (Lines 3-10). The resources of the range query and operators for the previous location are then freed after processing and sending the marker message. When all resources are freed the query reconfigurator receives the corresponding marker message and 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

Algorithm 3.10 Pipelined Query Prediction with Look-Head

```
1: Requires: MCEP Query Q
             eagerness //maximum number of future locations to look ahead
2: G_curr ← initial_query // current operator graph
3: B[Step] ← ∅ // Future operator graphs, indexed by future steps
4: current_step ← 0

5: upon update Location(Location l_u, Time t_u)
6:  if update_conditions(t_u,l_u) then
7:      inc(current_step)
8:      G_curr ← reconfigure_predicted_operatorGraph(t_u,l_u,G_curr,B[currentStep])
9:      discard(B,currentStep) //remove step from buffer
10:  end if
11:  l_next, t_next ← l_u, t_u
12:  for step ∈ [current_step, current_step + eagerness] do
13:      t_next ← next_predicted_update(t_next)
14:      l_next ← next_predicted_locations(l_next, t_next)
15:      if not_exists(B[step]) ∨ location_deviates(B[step], l_next) then
16:          discard(B[step]) //release resources (if an operator graph is already deployed)
17:          B[step] ← determine_interest(l_next) //generate copies of operator graph
18:          trigger start_streaming(R^p(B[step],t_next − Δ_a,D(Q), B[step], Q) at (t_next − T_C(l_next))
19:      end if
20:  end for
21: end
```

and afterwards delivers situational information produced by the new operator graph (Lines 13-16).

Pipelining Future Operator Graphs

Recall, when interest switches occur at a high frequency, e.g., since the focal object moves at a high speed, it can happen that not all historical events have been processed by an operator graph for a predicted future processing interest. This happens when, like in Figure 3.6, the time between two subsequent interest switches is shorter than the time it takes to detect all historical situational information in the temporal interest of the latter's predicted processing interest. In such a case it is useful to pipeline more than one operator graph. This means that future operator graphs are not only processing for the next possible processing interest, but also for a sequence of processing interests that are further away. For example, when switching to the processing interest R_i^p, not only the future operator graph for the processing interest R_{i+1}^p is deployed but also the future operator graph for the subsequent processing interest R_{i+2}^p.

Algorithm 3.10 presents the extended algorithm to consider this issue. The basic difference to Algorithm 3.8 is that iteratively more operator graphs are added for several

future locations of the focal object. To this end, an *eagerness* parameter is set by the system administrator that dictates how many predicted locations we look ahead. Moreover, we enumerate the location updates of a focal object that trigger a reconfiguration, denoted as steps, in the order of their occurrence. This allows us to keep track of all operator graphs that are deployed in parallel for several future steps in a buffer B and identify the operator graph which needs to be deployed for the latest location update (*current_step*).

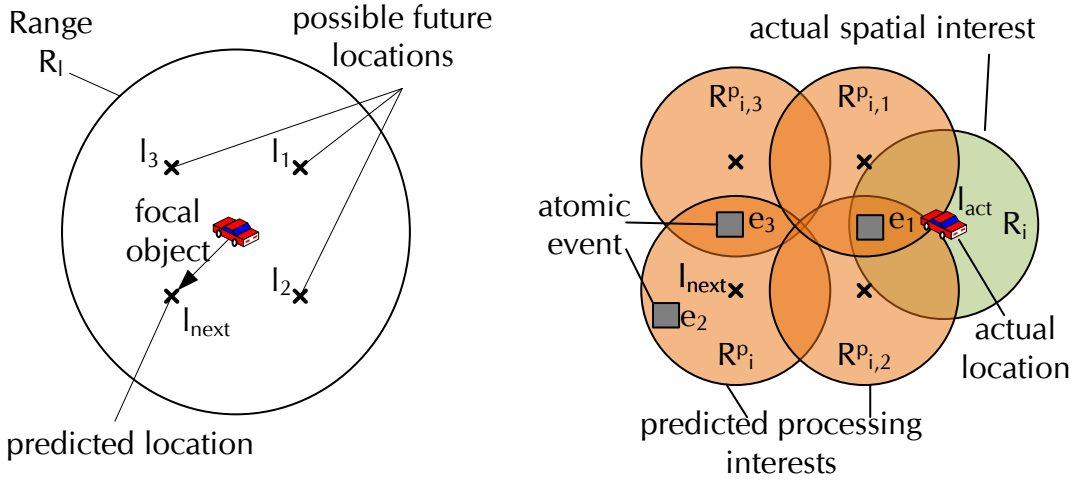
For each future step until the *eagerness* is reached (Lines 12-20), the location predictor is called with the predicted locations of the previous step as input. The intuition is that those locations are highly likely to represent the next actual locations (Line 14). The algorithm checks if an operator graph is already deployed due to a previous location update. If this is not the case, the system selects and initializes a new set of operator graphs at each of the predicted location. Moreover, if an operator graph is already deployed through a previous location update, the system checks if the predicted locations deviate beyond a threshold from the previous predictions and is suspected to detect situational information with a low QoR. In this case the operator graph that already processes for that step will be stopped and their resources will be freed, before a new set of operator graphs is initialized(Lines 15-18).

Properties. Note, that feeding a set of predicted locations to the location predictor increases the number (bounded by $\mathcal{O}(eagerness)$) of predicted locations and uncertainty, resulting in a large number of potential future operator graphs. A large resource consumption for processing a potentially large amount of operator graphs is thus traded off for a timely delivery of high quality situational information and controlled by the *eagerness* parameter.

State Reuse. Although the algorithm creates copies of operator graphs, it is not required to copy the whole state. Immutable state can be shared by the active operators hosted on the same broker. Moreover, situational information of an operator graph can easily be reused by other operator graphs, since it is buffered in the spatio-temporal event storage. This reduces the overall resource needs if multiple consumers deploy identical future operator graphs for the same spatial interest. When initializing operator graphs (Line 8 of Algorithm 3.8) the system has then to check if other operator graphs for the same spatial interest already exist. To this end, operator graphs are also indexed in the spatio-temporal event storage. However, now the system needs to coordinate when to safely release the resources of the operator graphs. Here, we propose a reference-counting mechanism that releases operator graphs that keeps track of the number of operator graphs that match in the processing interest and overlap in the temporal interest.

Processing Interests Determination

In this section we now detail our mechanism to increase the chance of detecting situational information with respect to a future processing interest that provides a good QoR by



(a) A vehicle can reach all possible locations within a range R_l in the time until the next interest switch occurs. A location prediction method predicts that the focal object will be at l_{next} at that time.

(b) A discrete number of processing interests are deployed, at the locations l_{next},l_1,l_2, and l_3 depicted in Figure 3.7(a). This way, our system can provide results with a high QoR from processing interests like $R^p_{i,1}$, when the focal object's actual location l_{act} deviates from the predicted location l_{next}.

Figure 3.7: Example for processing interest determination.

deploying multiple operator graphs that process for slightly different processing interests. For example, a low QoR is avoided in the scenario depicted in Figure 3.7b by deploying several operator graphs for processing interests around R^p_i, i.e., $R^p_{i,2}$ provides a high QoR with respect to the spatial interest R_i.The more future processing interests are deployed, the higher the chances to find a processing interest that highly overlaps with the actual future spatial interest of the focal object. Yet, although the focal object, as depicted in Figure 3.7a, can only reach locations confined in the circular area R_l before a location update triggers the next reconfiguration, it would require an insurmountable amount of resources to deploy an operator graph for each of this infinite number of potential future locations. To restrain the resource usage, a discrete number of n_g future operator graphs has to process events from slightly different processing interests. This number can easily be determined when the resource usage ρ_G for each deployed operator graph G is fixed, then $n_g = \lfloor \frac{\rho_{max}}{\rho_G} \rfloor$. However, the resource usage of an operator graph depends on the dynamic number of events reported in the area covered by the processing interests, e.g., more events will be reported on highly frequented highways than on a country road with light traffic. For instance, in the example depicted in Figure 3.7a, the workload of the operator graph processing for R^p_i is higher than the workload of the operator graph processing for $R^p_{i,3}$, since more events were reported by sources in that region.

Two approach directions can be considered to find a discrete number of operator graphs that ensure a high QoR. The first approach is to deploy operator graphs with respect to equidistant locations that span processing interests with a high spatial overlap, e.g., like in Figure 3.7b can be deployed. The distance between such operator graphs can be derived from our previous observation about correlation of the expected QoR and the distances between spatial interests and processing interests(see Section 3.3.2). Such a deployment will therefore 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, i.e., all deployed operator graphs require in total more resources than $\rho_{\max}$. Moreover, events are typically not uniformly distributed in space, e.g., in a traffic scenario events are only emitted by vehicles driving on roads, which can lead to redundant future processing interests. In the running example, both $R^p_{i,2}$ and $R^p_{i,3}$ can provide a QoR of $(1,1)$ with respect to the focal object's spatial interest. To avoid the aforementioned problems, MCEP opts for a second approach, where processing interests are selected based on the expected event distribution. For example, if our system anticipates that only events in the overlap of $R^p_{i,2}$ and $R^p_{i,3}$ occur, one operator graph for either $R^p_{i,2}$ or $R^p_{i,3}$ suffices. The downside of such an approach is typically higher processing costs to anticipate the event distribution compared to the first approach.

To find a good approximation that selects a set of future processing interests which can provide a high QoR while the resource usage stays below a resource limit ρ_{max}, we will first discretize the problem and then reduce it to a minimum set coverage problem, which is known to be NP-hard.

Discretization. For the discretization we assume that only a discrete number n_p of predicted locations are valid future locations of the focal object, e.g., all locations in Figure 3.7a form the set of predicted future locations `P`. As a result, the focal object's actual future location may deviate from all predicted future locations and even if operator graphs are deployed with respect to all predicted locations, the QoR may fall below the threshold q_δ. However, such a quality degradation becomes less likely the more n_p tends to infinity.

Set Cover Problem. Given the discretization, the problem is to select a minimal set of high quality processing interests (W) which are spanned by the discrete set of predicted locations `P`. In context of the example, we have to select from R^p_i through $R^p_{(i,3)}$ those processing interests that ensure a high QoR with anticipated spatial interests, which are spanned by the locations of `P` due to the discretization. Moreover, the expected computing resource usage must stay below a system defined limit ρ_{max}.[4]

To model our problem as a set coverage problem, we consider the set of atomic events that is covered by all processing interests spanned by all predicted locations `P` to be a set

[4]Note, this linear model is a simplification to explain the basic principles. Deploying a discrete number of operator graphs on a discrete number of distributed parallel resources is a bin packing problem,e.g., solved with a First Fit algorithm [LTC14].

Algorithm 3.11 Greedy Set Cover [CSRL01, Chapter 35.3]

```
1: function set cover(𝕏, F)
2:   W ← ∅
3:   U ← 𝕏
4:   while U ≠ ∅ do
5:       select an S ∈ F that maximizes |S ∩ U|
6:       U ← U − S
7:       W ← W ∪ S
8:   end while
9:   return W
10: end
```

of elements $\mathbb{X}$. In the example, $\mathbb{X} = \{e_1, e_2, e_3\}$. Since atomic events that are processed in the future by an operator graph are not known at the time when the algorithm selects processing interests, $\mathbb{X}$ can be built up by relying on recent atomic historical events. Those historical events give an indication about the spatial distribution of future events, as discussed in previous sections. Each processing interest can be viewed as a subset of $\mathbb{X}$, i.e., in the example the subset for processing interest R_i^p is $\{e_2, e_3\}$. Hence all processing interests represent a family F of subsets of $\mathbb{X}$, i.e., $F = \{\{e_2, e_3\}, \{e_1\}, \{e_1\}, \{e_3\}\}$.

The classical set cover problem is then to select a minimum number of subsets $W \subseteq F$, s.t., all elements of $\mathbb{X}$ are covered.

However, we have to consider two additional conditions:

- All processing interests of not selected subsets $N \subseteq F$ must overlap with areas of selected subsets $W \subseteq F$, s.t. the QoR for all sets in N is expected to be ensured.
- The overall expected resource usage ρ_{tot} over all operator graphs deployed for the processing interests must stay below ρ_{max}.

The first condition strives to ensure the quality, while the second one strives to ensure the resource limits. For example, even if $\{e_1\}$ is selected for R_i^p, $\{e_2, e_3\}$ is selected for $R_{i,1}^p$, and all elements are covered, we have to select $R_{i,3}^p$'s $\{e_1\}$ to ensure a precision of more than 0.5 to the spatial interest which covers the same area as $R_{i,3}^p$. Moreover, if the maximum of available resources ρ_{max} is low, it is possible that only one operator graph can process events, e.g., from the processing interest covering $\{e_2, e_3\}$, while $\mathbb{X}$ is not completely covered.

Greedy Set Cover Heuristic. Our greedy solution of the set cover problem for processing interests is inspired by the well-established heuristic solution of [Joh73] and [CSRL01, Chapter 35.3], which is still considered relevant today since being a polynomial-time $(ln|X| + 1)$-approximation. Algorithm 3.11 depicts this solution. The algorithm receives the set $\mathbb{X}$ and the family of subsets F as input. The algorithm iterates over the sets comprised in F until all elements of $\mathbb{X}$ are covered; i.e., a set U which comprises uncovered

Algorithm 3.12 Future Processing Interest Determination

```
1: Query Q //dynamic interest query
2: function determine_interests(Locations P) //P ⇒ set of processing interests F
3:   W ← ∅ //set of future processing interests
4:   Ψ ← calc_initial_costs(P, Q) //set of resource requirements per location
5:   ρ_tot ← 0 //expected resource requirements
6:   while P ≠ ∅ do
7:     l ← select_and_remove_next(l, Ψ, P, Q)
8:     if ∄w ∈ W : (precision(R(l), w), recall(R(l), w)) > q_δ(Q) then
9:       if ρ_tot + Ψ[l] ≤ ρ_max then
10:        W ← W ∪ {R(l)}
11:        ρ_tot ← ρ_tot + Ψ[l]
12:      end if
13:    end if
14:  end while
15:  return W
16: end
```

elements of $\mathbb{X}$ is empty. The algorithm chooses each round a set from F that contains the largest number of uncovered elements and adds it to the result W. After each round all elements in the selected subset are marked as covered by taking them from U.

Future Processing Interest Determination Heuristic. Our algorithm (Algorithm 3.12) is based on the heuristic greedy set cover solution and receives a set of predicted future locations P as input to determine a minimum set of processing interests W. In addition to the basic solution, however, our solution considers QoR and resource usage of deployed operator graphs. To this end, our algorithm estimates as a first step the resource usage for each processing interest spanned by a location in P 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. For example, a set Ψ can convey for our example depicted in Figure 3.7b that if R_i^p is selected sufficient resources are needed to process 2 events and sufficient resources are needed to process 1 event for $R_{i,3}^p$. In a second step, 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 [Joh73]. With this policy the algorithm always selects the next location that spans a processing interest covering the largest uncovered region of recent atomic events with respect to already selected processing interests in W. In the example, the location that spans R_i^p would be selected before the location that spans $R_{i,2}^p$ would be selected. Another policy is one that favors processing interests that is the most likely future locations of focal objects, e.g., based on given probabilities [HM12a] or because they are close to an actually predicted discrete location (*location policy*). A last policy (*resource policy*) aims to increase the number of deployed

operator graphs by favoring those with the lowest expected resource consumption. In the example, the location that spans $R^p_{i,2}$ would be selected before the location that spans R^p_i would be selected. A processing interest that is spanned by the selected location is then inserted to the result set W if it satisfies the following two conditions, which ensure the additional conditions to the classical set cover problem: (i) a processing interest is not suspected to be redundant, i.e., another overlapping processing interest $w \in W$ is already selected that satisfies the consumer's requirement 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 ρ_{max} (Line 8).

Selecting the operator graph. When the actual interest switch occurs, the query reconfigurator selects one operator graph from the set of future operator graphs that are deployed for the processing interests W. Policies to select the next operator graph G_{curr}—specified by the domain expert that designed the operator graph—are to switch to the operator graph with a processing interest that has the highest overlap with the current spatial interest of the query, the highest precision, or the highest recall.

Properties. The presented algorithm targets the basic query prediction algorithm (Algorithm 3.8). An extension for the pipeline approach is, however, straight forward: Ψ has to be maintained and checked for several future steps in Line 9 of Algorithm 3.12. Since Algorithm 3.10 prefers more current steps, fairness between operator graphs of different steps is not ensured. However, this can be achieved by executing the rounds of the loop in Algorithm 3.12 in a round robin fashion for each future step.

We provide a general framework to process MSA applications, however, the location prediction is highly dependent on the scenario,e.g., if a focal object drives along a well-known route or not [LNR02]. To this end, the query predictor is treated as a black box. Independent of the actual location prediction method, the number of predicted loactions, their density, and the density of locations of events, influences the QoR and resource consumption. Consider the set of historical atomic events ($A_h = \{a | a \in A(R_1) \vee ... \vee a \in A(R_{n_p})\}$) selected for n_p future processing interests, streaming costs for the whole operator graph are at least in $\omega(|A_h|)$. The probability p_{q_δ} for a QoR above the consumer's threshold q_δ, however, also increases with n_p. Consider a bounding area R_g for the predicted future locations, a circular spatial interest, and evenly distributed events. In this case, all predicted locations within a circular area R_q around the actual future location l_u span a processing interest with a QoR above the threshold; thus $p_{q_\delta} \geq \frac{n_p}{R_g}$.

3.4.3 Metric Estimation

All approaches to ensure timeliness rely on a QoR 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. How this metric can be anticipated has already been discussed in Section 3.3.2.

We now discuss how the processing time for historical events by an operator graph can be anticipated, in order to automatically adjust the T_c (see Algorithm 3.8).

Timeliness Estimation 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_e be the average processing time per event, R^p be a processing interest, and $ev(R^p)$ be an average value of events per second in the spatial range determined from the recent historical atomic events already available in the area covered by R^p, then the processing time computes T_C to:

$$T_C(R^p) = C_i(ev(R^p)T_e) \tag{3.12}$$

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

3.5 Evaluation

We evaluated the MCEP system using the Omnet++ network simulator [VH08]. To model realistic traffic patterns of vehicles on an OpenStreetMap graph [HW08] we used the traffic simulation package SUMO [BBEK11].

3.5.1 Common Setup

Our simulations cover three MSA applications. First, an *accident detection* application similar to the one detailed in Section 1.1.3, where the accident is detected when a sudden speed drop in an area is followed by a high number of lane-changes. Second, a *video friend finder* application that streams videos of friends that were recently close to a focal object, 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 the one depicted in Figure 3.3, where the average speed was calculated over the most recent speed events that were above a predefined minimal speed.

Over the course of 500 to 1,000 simulated seconds, upto 5,000 vehicles drove on a street map of a German town near Stuttgart, with corresponding speed limits, of the size 7.7 km $\times$ 3.5 km. The vehicles emitted events when accelerating or decelerating beyond a threshold or changing lanes. Each vehicle published, on average, one event

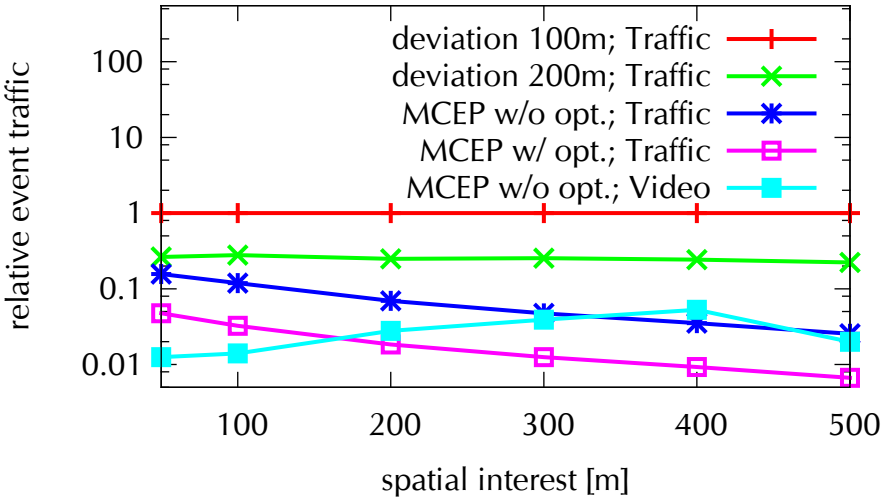


Figure 3.8: Predeployment vs. MCEPs

per second in this scenario. The spatial interest of consumers was described in all scenarios by rectangular shapes with varying edge lengths. We repeated the experiments approximately 10 times with different sets of trajectories for the vehicles. All vehicles were connected to the same broker.

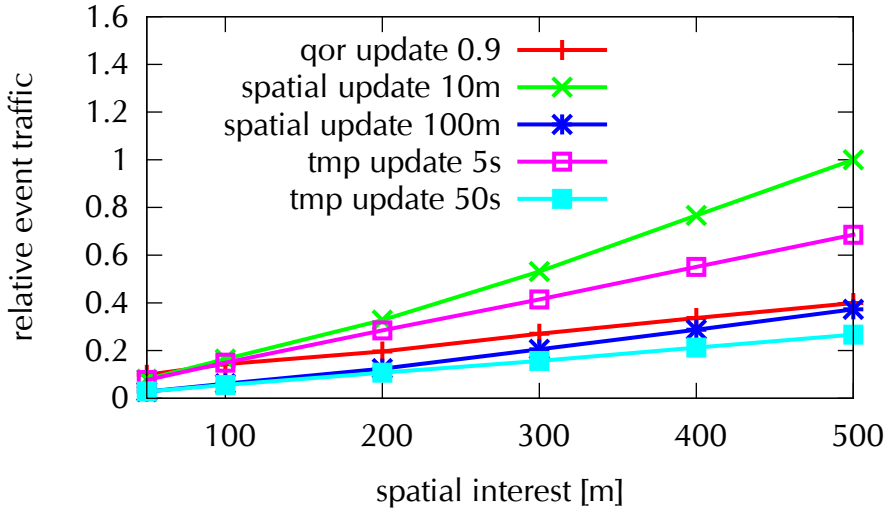
3.5.2 Evaluation of Query Reconfiguration

In this section we present and discuss our evaluation of the basic MCEP adaptations and corresponding optimizations to save resources using a single operator graph.

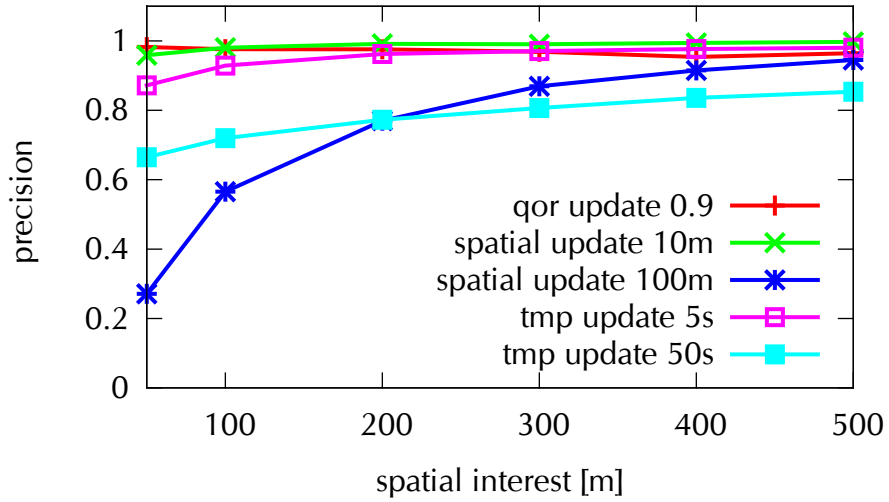
MCEP vs. Predeployment.

We evaluated the benefits of using the basic MCEP approach (*MCEP without optimizations*) and the streaming optimizer (*MCEP with optimizations*) compared to an approach with a fixed amount of statically deployed overlapping processing interests (*pred.*) in terms of resource usage. 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 3.8 depicts the number of streamed events (*event traffic*) relative to a fixed set of operator graphs with a deviation of 100 m for different sizes of the spatial interest, varying from 50 m to 500 m. It shows that our approach of dynamically adapted processing interests, with and without the streaming optimizations of Section 3.3.3, reduces the number of streamed events for a single query Q in both scenarios. The savings of our approach are achieved since only events of a sub-part of the geographic space, selected



(a) Update Strategies - Events.



(b) Update Strategies - QoR.

Figure 3.9: Evaluation of Update Strategies

by the spatial interest, have to be processed. Deploying more *diq*s linearly increases the results.

Impact of Reconfiguration Conditions.

We also evaluated the benefits of the reconfiguration conditions presented in Section 3.3.2 in terms of the event traffic and their effect on the QoR for the accident example. In particular, we performed interest switches when a new reported location of a consumer deviates a fixed distance from the previous location, when more than a predefined time

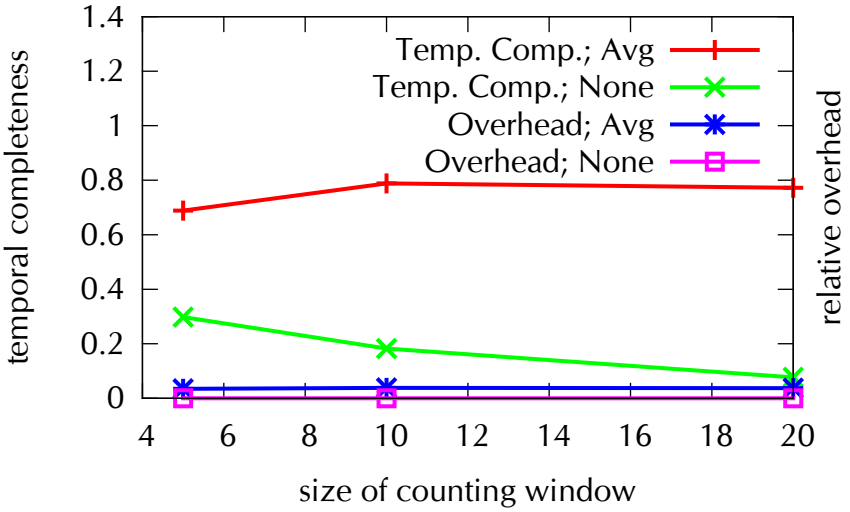


Figure 3.10: Temporal completeness

passed since the last interest switch, or when omitting the next interest switch is expected to drop the QoR, represented by the precision, below 0.9.

The results depicted in Figure 3.9a show the relative event traffic with the maximal number of streamed events as baseline over several spatial interests, while Figure 3.9b 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.

Impact of Initialization Points.

We further analyzed the impact of several approaches to select 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 restriction window of 25 s. In particular, we tested the basic approach, the window approach that uses an average over the last inter arrival times of events (*Avg*), and a naive approach that did not stream any historical events (*None*).

Figure 3.10 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 correlation 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

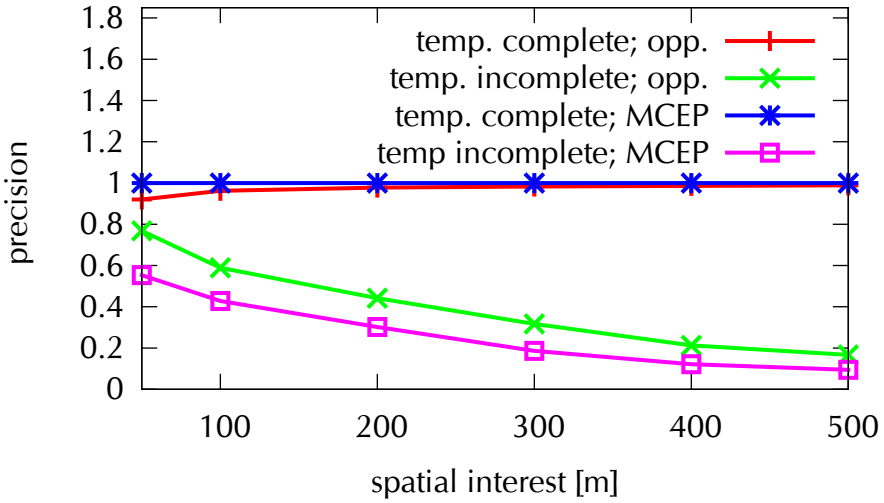


Figure 3.11: Precision vs. spatial overlap

completeness of around 80%. Not streaming historical atomic events at all leads to a temporal completeness of 20%.

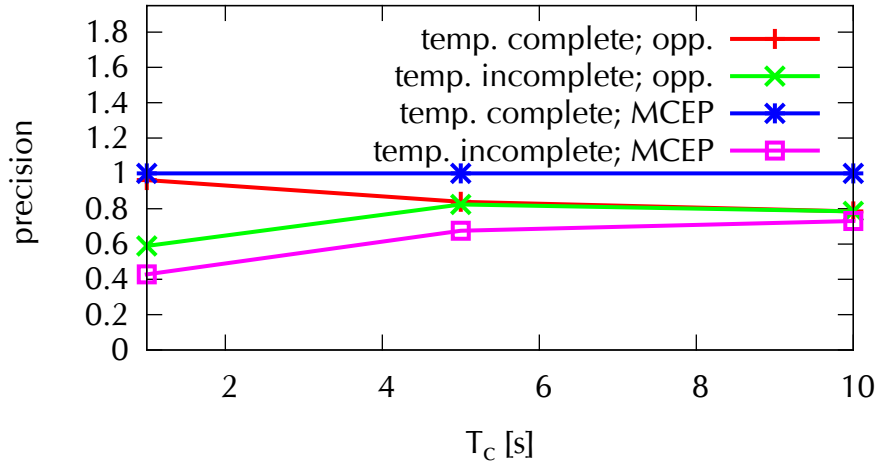
3.5.3 Evaluation of Query Prediction

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 time T_c 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, i.e., 5 ms 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 graph s (*opp.*). On the other hand, we deployed operator graph s 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*).

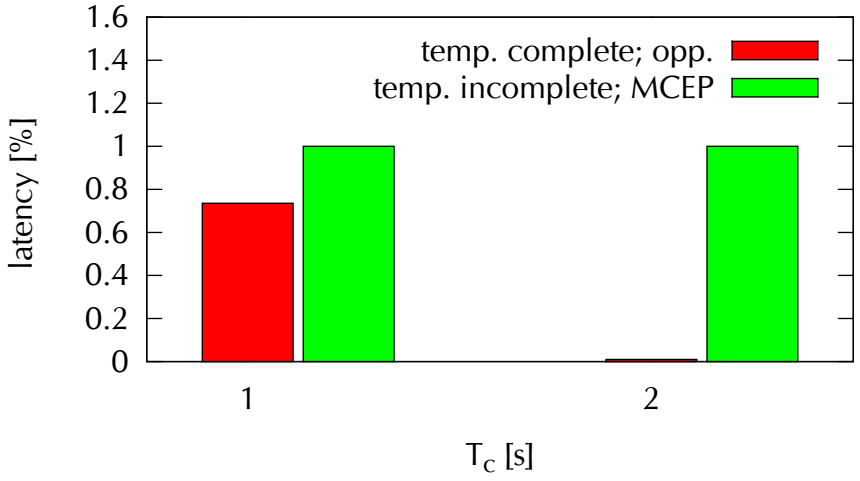
Impact of size of spatial interest.

The results in Figure 3.11 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 1 s 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 interest switch is

performed and does not stop until all relevant events are processed, at the cost of late results (i.e., up to 10 s). 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 correlation steps can be finished until the next operator graph switch is performed.



(a) Precision vs. eagerness



(b) Relative impact on latecy

Figure 3.12: Impact of T_c

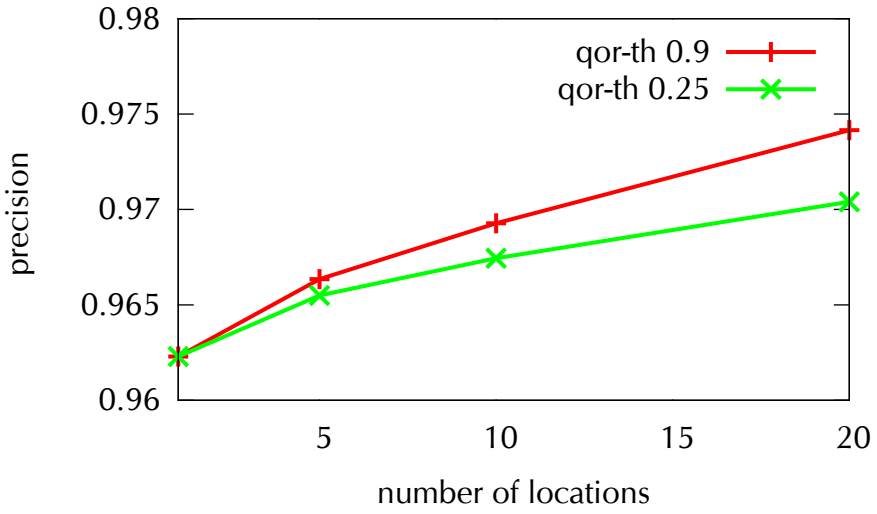


Figure 3.13: Overprovisioning of operator graphs

Impact of T_c.

The results in Figure 3.12a show the effects of varying the time T_c on the precision and the temporal completeness. With a higher T_c, 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 T_c, since the location predictor has to provide consumer locations that are further in the future.

Moreover, the results in Figure 3.12b show how the time T_c (x-axis) affects the the latency (y-axis) after an interest switch until the first situational information with respect to the new processing interest can be detected. The results are relative with respect to the average latency incurred by MCEP's reactive approach, i.e., when operator graph's are deployed on demand. In particular, the results show that with the selection of a sufficient time T_c, like two seconds for this scenario, we achieve an interest switch with near zero latency.

Impact of Overprovisioning.

For the results in Figure 3.13, we increased the number of initially selected locations and applied the location selection algorithm (Algorithm 3.12) with the location policy. For these experiments we kept the eagerness at one second and the spatial interest at 500 m. It shows that using more predicted locations increases the precision since the consumer can choose between more deployed operator graph s 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.

3.5.4 Discussion

The mechanisms implemented in the MCEP system prove to save bandwidth, computational resources and deliver historical events at a low latency while ensuring completeness and a high QoR. Our system's mechanism to adapt the processing of an operator graph depending on a focal object's location allowed us to reduce bandwidth and computing resources tremendously since the number of streamed events could be reduced by over 99%. Reducing the frequency of operator graph adaptations with a QoR-aware reconfiguration allowed us to further reduce the number of streamed events by a factor of 2 while keeping the QoR above 0.9. Best effort completeness based on estimated initialization points allowed us to reduce the overhead for streaming historical events by up to 98% while ensuring a completeness of 80%. Our prediction-based algorithms allowed our system to keep a high QoR tending to 1 while avoiding latencies of up to 10 seconds. Tuning the eagerness parameter for the concrete scenario allowed us to achieve a QoR of up to 0.8 without delivering late events. Over-provisioning operator graphs further allowed us to increase the QoR.

3.6 Related Work

Processing Models. Many traditional CEP approaches [UMJ+14, PSB04, GD94, KKR10, CM94, WDR06, Luc01] address how to efficiently detect patterns on event streams. However, the operators and sources are statically defined before the query is deployed and hence cannot react on location updates. Many different methods for the detection of events have been proposed, such as Petri Nets [GD94], finite state automata [PSB04], detection trees [CM94], or disjoint normal forms [KKR10]. The introduced partitioned window model allows in contrast to previous work to efficiently determine the temporal and spatial overlap in two subsequent selections and hence especially helps to significantly reduce the reconfiguration overhead of mobility-aware CEP. While many different window semantics were presented for stream-based systems, e.g., [ABW06] or [PS11] this work can be considered the first work to realize typical CEP operations in a partitioned window model.

Operator Graph Adaptations. For spontaneous as well as continuous range and moving range queries, a vast amount of approaches has been proposed [PJT00, CBL+10, FCR07, TDSC07, GL06, DHL09, BKR11]. The main objective of these approaches is the selection of primary events, e.g., sensor streams, relative to a consumer-specified region in an efficient manner. In contrast, for complex event processing, complex events are the result of multiple dependent operations on primary as well as complex event streams. To account for a meaningful event delivery semantics, i.e., to account for completeness and order of produced event streams, a coordinated reconfiguration of all operators is required.

Some systems like Place* [XECA07] build moving range queries over specific aggregation functions. The approach supports the distributed processing of a single aggregation

operator. However, the distributed processing of multiple dependent operators is not addressed by existing work. This way, it becomes possible to apply many optimizations used for distributed CEP systems [PLS+06, RDR10, SKR11, VKR11] also in the context of distributed range queries.

Lately, distributed event based dissemination systems [EFGK03, TKK+10, HEG12] have proved the benefits for using parametrization of queries to support mobility-aware applications. In [JJE10], a parametrizable publish/subscribe system is presented, which enables location based queries. Yet this work focuses on the reconfiguration of stateless operators like a filter. In order to support stateful operations, our work has shown that it is necessary to determine appropriate initialization points for the chain of dependent operators in order to provide guarantees for the delivery of events.

The importance of spatial correlations of atomic events has been recognized [ZTH08]. Systems like [HV03] detect events as soon as the location of a focal object changes, yet do not provide the vast set of optimizations presented in this chapter. Eventlets [AFFB12] are execution environments with a managed life-cycle. The eventlet middleware allows MCEP systems to instantiate a processing interest specific execution environment when a new location event occurs, this way ensuring spatial consistency. However, it needs to be extended by our marker messages and initialization points to detect when the execution environment can safely be discarded and what historical events need to be processed. Moreover, eventlets are designed for isolated execution environments, which requires additional coordination overhead between pairs of eventlets that are processing for subsequent processing interests in order to find duplicated events.

Processing of Historical Event Streams. Similar to our initialization of the operator graph using historical event streams systems, like Aurora [CcC+02] or PSoup [CF02] process "ad-hoc queries" over historical event streams, before processing live event streams. Aurora processes all historical events that are buffered for a fixed amount of time, while PSoup allows users to specify a fixed time in the recent history that indicates the initialization point. However, those systems provide no notion of a maximum covering sequence and therefore no mechanisms to dynamically determine the number of historical events required by a specific operator graph.

Corrections of a stream, e.g., due to changes of a historic event, have been previously studied in the context of temporal streams [MC08]. To this end, they propose to replay streams and avoid duplicates by incrementally updating streams, similar to our duplicate detection technique that uses temporal overlaps. However, they do not provide operator classes and therefore specialized methods that are capable of dealing with spatial overlaps.

Dynamic reconfigurations of operator graphs mostly focused on adapting an operator graph to dynamic data rates. Typical solutions are changing the order of operators [AH00, HSS+14] or the access pattern on event streams [DFST11]. These solutions therefore do not support the adaptation to new spatial interests. Up to now, Borealis [AAB+05, SESFT11] 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 object updates the location with the system. However, the system does not provide methods to automatically decide how far the system has to rewind, limit their revision to non-spatial 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 failures [KMR+13, SM11] or to ensure event orderings by blocking operators and revising events [BGAH07]. This does not cover spatial inconsistencies as addressed by this work, which occur, for example, if an individual operator graph for each focal object is used and events of an updated spatial interest are streamed without updating the operator graph. Note that operators keep events in histories [AE04], i.e., a partial result of a correlation step [PSB04, ABB+03] 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.

Moreover, range queries provide spatial completeness of sensor data if they do not provide approximated results [AKTS11], 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.

Consistent Event Processing. Wang et al. [WRE11] propose transactional rules to adapt the state of a CEP system efficiently and consistently. These state changes, however, break down to information in a table of an external relational database. In contrast, our system needs to consistently adapt operator graphs, which involves adapting an operator's state and streaming historical events.

Our marker messages resemble notifications [MMI+13], tick-tags [NRB09] or punctuations [LCT+06, DR04, DRH03, TMSF03] that can also be injected into streams to indicate the end of a maximum covering sequence. However, although the systems that employ those mechanisms provide highly expressive options to process such marker-like messages, rules to merge dynamic markers from multiple incoming streams at operators, based on the sub-trees that need reconfiguration, are not directly supported and must be implemented for each individual operator. Moreover, methods to automatically decide when and how to adapt the operator's state, based on marker messages, are not provided.

A quality reduction through omitting or summarizing events has been proposed to reduce the streaming and processing load with respect to network and disk I/O costs [CF04]. Our duplicate reduction mechanism goes beyond these approaches by

considering a location-aware metric for the quality and proposing matching algorithms that interweave with this metric.

Timely event delivery. Typical CEP systems [PSB04, LJ05, AE04, CM10, KKR08] provide methods to efficiently detect patterns on a variety of sensor data for situation-aware applications. They even considered exploiting parallelism [CM12a, Hir12] or adapting the placement of operators [CM13, HLSD11, LLS08, RDR11, PLS+06] to achieve low latencies and a good system performance. However, they do not offer a notion of initializing operator graphs for new locations, since they are typically fixed to a set of sources. Moreover, hiding computing latency is not a big issue, since they typically process on live and not historic events.

Mobile publish-subscribe systems allow to adapt the streaming to new locations of consumers [JJE10, MPGJ05]. They even allow pre-subscriptions to new locations [CFH+03] before the consumer arrives at the location and loss-less event delivery for intermittent connectivity [HMLJ09, MUH04]. Furthermore, low-latency is achieved through parallelism [WZC+12] and adaptation of routing paths [TKK+10]. However, they do not have to consider processing of events and the time it takes to perform the computations as well as the quality of the result of computations, since they focus on individual events rather than processed complex events.

Large-scale situation awareness [RHI+12] is receiving increased attention due to the proliferation of sensors and advances in analytics such as computer vision. Target Container [HSS+11] is one of few distributed systems providing a high-level, handler-based programming abstraction that helps domain experts to write large-scale surveillance applications on camera network. SLIPstream [PMS+09] 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 huge amount of system resources in large scale. In contrast, our system performs analysis of sensor data based on user queries and cache the query results to reuse, resulting in efficient use of system resources.

Our work focuses on efficiently processing historical data. Work that is concerned with storing and indexing temporal event streams [RSW+07, DS07] or spatio-temporal event streams [MS05, AN08] efficiently is therefore orthogonal.

Predictive Query Processing. *Predictive queries* [ZJDR10, HM12a, HM12b] give results describing the future of the focal object, e.g., the taxis that will be in a 500 m radius around the consumer 5 min 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.

Proactive CEP systems allow users to predict a future event to prevent severe problems, e.g., to avoid traffic congestion by adjusting variable speed signs [ABB+14], which is orthogonal to our approach of reactively detecting situations.

3.7 Conclusion

In this chapter we have presented a mobility-aware CEP system that delivers complex event streams depending on a dynamically changing interest of a consumer. The system only processes event streams of interest to a consumer and this way avoids the overhead imposed by a predeployment of operators. Furthermore, we have introduced a mobility-aware delivery semantics that ensures for a query and each of its ranges to produce a maximum covering sequence. Event streams exhibit a well defined order in which events are delivered to consumers respecting the order of range updates and ensure the preservation of the chronological time order between updates. We have shown that the delivery semantics can be realized cost efficiently by relying on the properties of the proposed event execution model, which partitions incoming event streams in separate selection windows, and the spatio-temporal overlap of a consumer's interest.

Moreover, we also discussed the latency bottleneck of streaming and processing historical atomic events that need to be processed in order to detect historical situational information before it becomes of interest to a consumer. To this end, we have proposed methods to address the problems caused by this delay. The metrics of interest that we improved are the precision and recall with respect to atomic events for dynamic interest queries. To this end, we proposed a method for opportunistic computation of historical events, including a pipelining method to look several steps into the future and an opportunistic computing method that over-provisions operator graphs to compensate for partially inaccurate location prediction results.

The evaluation of our system by simulations indicate that our mechanisms can greatly improve the bandwidth usage, reduce the required computing resources, and increase the latency for delivering historical events in MSA scenarios. The number of streamed events—a key performance indicator for the bandwidth required to stream events and computing resources required to process events—in a traffic scenario can be reduced by up to 99.9% using our mechanisms of reconfiguring operator graphs. This result lets us conclude that more than 1,000 vehicles need to be interested in the very same situation in this setting to justify predeploying operator graphs. Moreover, our evaluation results suggest that by slightly reducing the quality of delivered complex events, in terms of precision and recall, it is possible to greatly improve the system performance of a CEP system. For instance, consumers accepting a precision of 0.8 in a traffic scenario allow our system to further reduce the number of streamed events by a factor of 2 and to achieve a near zero latency when delivering historical situational information.

4 Mobility-aware Migration and Placement Algorithms

This chapter's focus is on approaches to deploy operator graphs in large-scale, geo-distributed infrastructures. For example, virtualized computing infrastructures like clouds or fogs.

In *Distributed CEP (DCEP)* [PSB04], research [PLS+06, RDR11] has shown that the placement of operators—the concrete assignment of a computing node to execute an operator—has a significant impact on important performance metrics, e.g., end-to-end latency and network utilization. However, in *MCEP* systems, where consumers and sensors are mobile, bandwidth and latency of streams are expected to change frequently with continuous location updates. For example, the number of available camera streams close to a focal object or the latency between the access point of a mobile sensor and the computing node where an operator is placed varies. To ensure the system's performance it has to constantly adapt the placement through migrations to new computing nodes that are topologically closer to sources and consumers.

Each migration comes with a cost, because operators are associated with local state, e.g., a cached portion of the event stream or a street map. This state can accumulate up to several gigabytes of data [SM11] that have to be transferred during a migration. Frequently performing migrations to find better placements thus can significantly decrease system performance. For example, migrating gigabytes of state with each cell change of a consumer in a GSM network, while only several megabytes of data are streamed to and from operators increases the network utilization. Moreover, operators need to be stopped during the migration and are not operative during a migration from one computing node to another. In context of a video stream operator one can observe a delay of several seconds [OMK14].

We propose to address this problem with the MigCEP live migration system that supports operator migrations in our MCEP system. In particular, methods are proposed in order to maintain low end-to-end latencies while reducing the network utilization. These methods exploit application knowledge of the MCEP system and predicted mobility patterns to plan the migration ahead of time. First, it allows the system to amortize the migration costs by selecting migration targets that ensure a low expected network utilization for a sufficiently long time. Second, it allows the system to serialize the operator for the migration and migrating parts of the operator a priori in away where unnecessary events are not migrated and bandwidth is reduced. In more detail, the contributions, based on findings that are published in [OKRR13] and [OKR+14a], are:

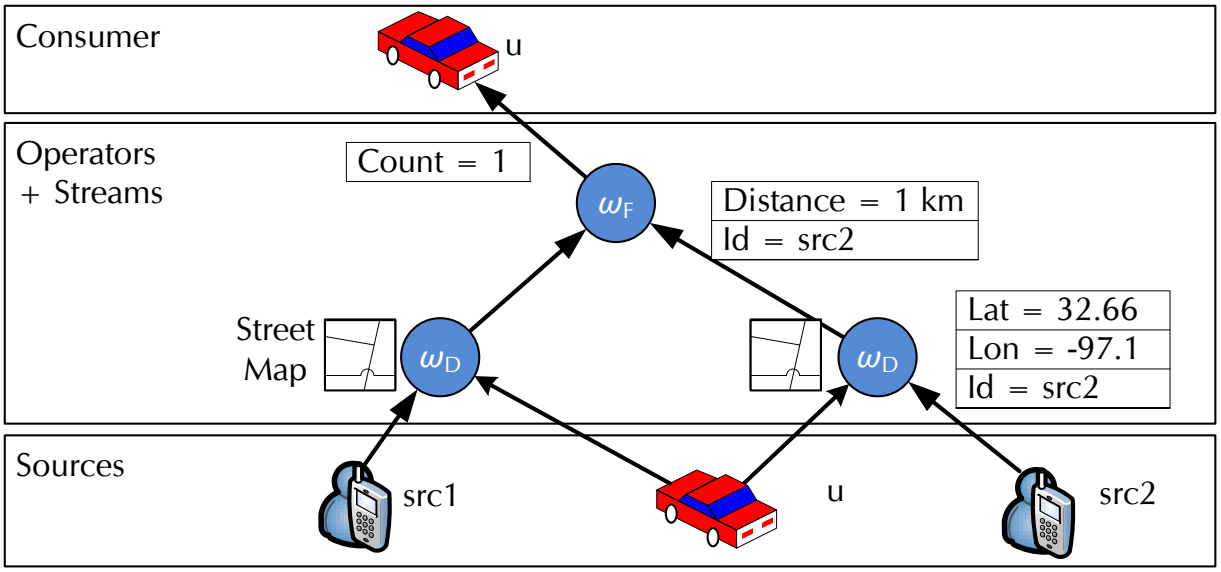


Figure 4.1: CEP operator graph that contains individual sources.

1. The definition of a probabilistic data structure, called *Migration Plan (plan)*, which defines future target computing nodes and times for the migration, and a distributed algorithm to create such plans.

2. A migration algorithm, which uses a plan to minimize the network utilization while keeping the end-to-end latency below a threshold.

3. An analysis and evaluation study of the cost imposed by the creation and execution of a plan and its benefits.

The remainder of the chapter is structured as follows: Section 4.1 introduces the underlying system model and Section 4.2 clarifies the problem. We present in Section 4.3 our plan-based migration approach and in Section 4.4 we describe how this approach is implemented in a demonstration scenario. In Section 4.5 we present findings from our analysis. The approach's evaluation is presented in Section 4.6. Section 4.7 discusses related work before we conclude the chapter in Section 4.8.

4.1 System Model

To support a broader range of CEP queries than MCEP queries, sources $src \in D$ of an operator graph G are not restricted to reflect processing interests as in previous chapters, but can also represent individual mobile sensors. This means that operator graphs do not necessarily need to be reconfigured according to the location of the consumer. For example, Figure 4.1 depicts an operator graph that can detect the number of friends that were closer than 1 km to a consumer u over the last hour regarding the distances given by

a street map. Atomic events are location updates of users, associated with a source-specific identifier (id), which are streamed to ω_D to calculate the individual distances between a friend and u. Operator ω_F then counts the number of distances smaller than 1 km with disjoint source-ids and publishes the result on a social platform or the mobile device of the consumer.

Recall, the correlation function f_ω of an operator $\omega \in G$ detects events on deterministically ordered event streams. Events of those streams are managed in a set of dedicated buffers B. The number of buffered events in B depends on the operators' semantics, which determines the selection and consumption of events from the queues. We refer to these dynamically changing queues as *mutable state*, while the *immutable state* of an operator is the part of the operator that is read-only for f_ω, rarely changed, and fixed in size. For example, the map of ω_D, a database for face recognition, or the executable code of the operator. Note, operators might dynamically load context-dependent immutable state, like the map depending on the friend's location, which, however, is also fixed in size. Intermediate results, i.e., states of variables in an operator's implementation, are denoted as *computational state*.

4.2 Problem Description

A placement Φ_{t_s,t_e} is the assignment of operators $\omega \in G$, of a given operator graph G, to brokers in between time t_s to time t_e. When a mobile source or consumer connects via new access-points to new leaf brokers the end-to-end latencies and the network utilization can increase since now more brokers are potentially involved in transferring event streams. For example, when the source in Figure 4.2 changes its connection from the leaf broker b_1 to b_2, it also means that from now on all streams for ω_D, placed on b_1, have to be transferred from b_2 to b_1, possibly over multiple steps in the broker hierarchy. In order to improve the system performance, the system has to adapt the placement $\Phi_i = \Phi_{t_s,t_e}$ at time t_e to a new placement Φ_{i+1} via migrations $M_j, .., M_k$ of one or multiple operators. To reduce the network utilization imposed by keeping ω_D at b_1 the system migrates ω_D to b_2, which can also affect the placement of ω_F.

Note, that also migrations themselves may impose a significant cost in terms of network utilization, given that a migration requires transferring the whole state of an operator to a new migration target which can be as large as several gigabytes. Therefore, it is important to account for those costs when performing a migration. Instead of always deploying the best possible placement, a placement should only be deployed if its migration costs can be amortized by the gain of the next placement. This gain depends on many dynamic parameters such as the mobility of sources and consumers as well as the actual workload of the event processing system.

The decision to initiate a migration may also be unavoidable in some cases for ensuring the responsiveness of the CEP system, e.g., to allow users to respond to traffic situations immediately. In this case, the global end-to-end latency has to stay, at least on average, below a consumer-defined restriction LR_u. In particular, it has to stay below the restriction

on all paths in G from any source $src_i \in D$ to a consumer $u \in U$. On these paths, the detection is delayed due to communication delays between neighboring operators and the computational delay, the time that an operator requires to process a new input event.

When preserving latency constraints it is also important to consider the downtime of operators during migrations, as a consequence of having to halt an operator and start it again at the migration target. When all state is already available on the migration target at the migration time we achieve a minimal downtime, since the operator can start processing immediately. Consider, for example, in Figure 4.2, the map and event streams can already be transferred to b_2 before the source connects to b_2 and the operator ω_D has to be migrated to b_2. However, uncertain future locations of the source may lead to a situation where the network utilization is increased because the state is copied to b_2 but the source never connects to b_2 and a migration is unnecessary. This requires to migrate the operator and its state to future placements where the expected network utilization is low.

The *mobile migration problem* addressed in this chapter is to find a sequence of placements $\Phi_1, \Phi_2, ..., \Phi_p$ and Migrations $M_1, M_2, ..., M_m$ for an operator graph G in a broker hierarchy, where the overall network utilization is low and the consumer defined latency restriction is met. More formally, we consider the average bandwidth-delay product during a time interval as the metric to express network utilization. Let $\text{bdp}(\Phi_i)$ be the sum over the average bandwidth-delay product of links in the broker network in a placement Φ_i. Then, the *placement cost* for streaming events and control messages like heartbeats in G between time t_s and t_e is $C_{pla}(\Phi_i) = \text{bdp}(\Phi_i) * (t_e - t_s)$. We can define the *migration cost* $C_{mig}(M_j)$ for transferring the operators' states for each migration M_j, accordingly. Furthermore, let $d(src_i, u)$ be the end-to-end latency from any source $src_i \in D$ to any consumer $u \in U$ over a path $(\omega_1, \omega_2, ..., \omega_n)$ in the operator graph.

An optimal sequence of migrations is found if the *overall costs* are minimal, i.e., :

1) $C_{tot} = \sum_{i=0}^{p} C_{pla}(\Phi_i) + \sum_{i=0}^{m} C_{mig}(M_i)$ is minimal

subject to

2) $\forall u \in U : \sum_{i=0}^{|D|} d(src_i, u) \leq LR_u$ at all time

4.3 Plan-based Migration and Placement

The mobile migration problem needs to be approached in an online-fashion, since many dynamics influence the overall costs, like the continuously changing locations and therefore access-points of mobile sources or the network latency between brokers. Any appropriate online approach requires that a MCEP system needs to repeatedly perform several tasks at run-time. Based on monitored values, e.g., a trajectory spanned by the most recent locations of a mobile source, the MCEP system needs to decide at which time operators of the operator graph need to migrate and to which broker they need to migrate to. Moreover, the MCEP system needs to actually implement the migration.

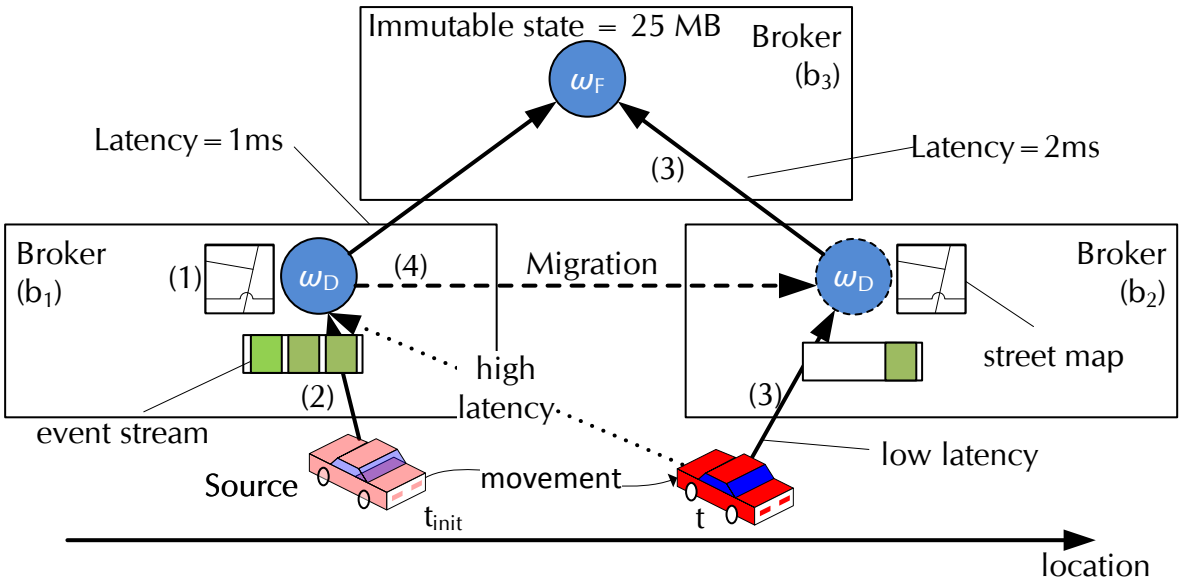


Figure 4.2: Overview of migration steps

Two basic approach directions can be considered when deciding at which time operators need to migrate to which broker: a reactive and a proactive approach. The *reactive approach* springs into action when the system performance is reduced due to recent changes in the MCEP system, e.g., when location changes indicate a new access-point of a mobile source and the network utilization has risen above a threshold. The MCEP system then reacts by finding a new placement which mends the inefficiencies. However, this means that the system performance remains low during the execution time of the migration. A *proactive approach* finds future placements based on predictions of consumer and source locations and the expected number of events streamed in an operator graph. This approach allows MCEP systems to initiate the migrations in time so that latency constraints are met and the anticipated network utilization is low. The downside of this approach is that predictions are uncertain. In a worst case, e.g., when consumers or sources move in the opposite direction of the prediction, the operator is migrated to a broker that does not ensure the latency constraint and increases the overall network utilization.

We use a proactive approach due to its prospects of keeping latencies and the network utilization low. MCEP's migration system assigns a dynamically updated *Migration Plan (plan)* to each operator. The plan defines for an operator a set of future brokers as migration targets, a deadline by when the 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. Furthermore, we incorporate aspects of the reactive approach by taking action when latency restrictions are no longer fulfilled or location predictions deviate beyond a threshold from actual locations of consumers and sources.

To avoid a downtime during the migration of operators, we use a live-migration approach to implement the migration. The original broker of an operator keeps processing

until all relevant state is transferred to the target broker of the migration. In particular, our system triggers transferring state to the target broker before the migration occurs. This way, our system can instantly switch to the target broker when needed.

In this section, we first give an overview over our approach (see Section 4.3.1). We then propose a simple model for a plan that triggers migrations at discrete time steps (see Section 4.3.2) and later show how to create an operator's plan based on the plans of its neighbors in the operator graph. Furthermore, we will show how to refine the basic mechanisms to deal with the inherent uncertainties which follow from the mobility of sources and consumers, variations in the workload, and resource usage in the broker hierarchy (see Section 4.3.3). We conclude this section by describing an algorithm that uses plans to execute a migration (see Section 4.3.4).

4.3.1 Approach Overview

We now give an overview over our plan-based approach to the mobile migration problem, which boils down to two operations: the *plan creation* to determine and update the operator's plans and a *plan execution* to implement the live-migration.

A plan can be executed and determined using either centralized or decentralized algorithms. Centralized algorithms have global knowledge about the network and the operator graph, i.e., its utilization, latencies between brokers, the placement of each operator, and the workload. Decentralized algorithms have only partial knowledge, i.e., for an operator ω hosted on broker b they only know about latencies between some brokers in the network and only know about the placement of operators that are connected over a stream to ω. Similar to the findings about centralized and decentralized placement algorithms [LLS08], we argue and show in Section 4.5 that centralized algorithms are vulnerable to scalability issues. To this end, we propose a decentralized algorithm.

In particular, each operator determines and executes its own plan locally at the broker it is currently placed at. Such plans, when executed, reduce the expected local placement and migration costs in order to reduce the expected overall placement and migration costs. The local placement costs of an operator ω in our previously defined cost metric are the sum of the average bandwidth-delay products between ω and each of its neighbors $\omega' \in \Omega_N(\omega)$ in the operator graph over a time interval $[t_s, t_e]$, i.e., $C_{Pla}(\omega) = \sum_{\omega' \in \Omega_N(\omega)} \mathrm{bdp}(\omega, \omega')(t_e - t_s)$. The local migration costs ($C_{Mig}(\omega)$) are the average bandwidth-delay products for transferring the operator's state during a time interval $[t_s, t_e]$ to a future migration target. Moreover, these plans adhere local latency restrictions, i.e., they ensure that the computational delay at the operator's future placements and the network delay to neighbors in the operator graph stays in average below a threshold.

Plan creation. Our system reacts to changes, like variations in the latency of an event stream due to a changed access-point of a vehicle, by creating a new plan for each operator connected to the event stream. In particular, it estimates the local migration and placement costs for multiple possible future migration targets of the operator. The plan is then

created by selecting the migration targets that expose the lowest expected migration and placement costs and ensure the overall latency restrictions. For example, ω_D depicted in Figure 4.2 anticipates that the expected costs are lower for migrating at time t to b_2 than for staying at b_1 since the vehicle will change its connection.

In order to be able to estimate these costs, an operator ω continuously anticipates the load of event streams and latencies between the operators connected to ω in the MCEP system. Operator ω also maintains a copy of the plans of each of its neighbors in the operator graph. The event load on links is estimated on average over the most recent traffic measurements or, in case of MCEP queries, based on the most recent atomic events in the expected future processing interest, while latencies can be estimated via regular ping messages between neighbors or using Vivaldi Coordinates [DCKM04]. The plan of the neighbor in the operator graph allows our algorithm to determine if a neighbor is placed on a different broker than ω in the future and causes network traffic, e.g., by streaming events from b_2 to b_3 in the example.

Leaf operators also need to account for the uncontrollable movements of consumers and sources when determining a plan. To this end, our system additionally considers *non-executable plans*, which describe the future connection patterns of mobile sources or consumers. In our example in Figure 4.2, this is the pattern of the vehicle, which is first connected to broker b_1 and then migrates its connection to b_2. Our system anticipates such patterns for sources in regular intervals by predicting future locations and therefore access points of the mobile consumer or source. Such predictions rely on well-known methods like dead reckoning [WSCY99].

A plan created by an operator like ω_D in our example directly affects the plan creation of neighbors like ω_F. For example, since ω_D plans to migrate at t to b_2, the network latency increases for the event stream connecting ω_F and ω_D. To this end, operators enter a recursive *plan coordination* step: Neighbors in the operator graph exchange plans until a stop criterion is reached. For example, ω_D sends the new plan to ω_F, which triggers a plan creation at the receiver. If ω_F's plan changes as a result, it also enters a coordination step and sends ω_D and other neighbors this new plan. This continues until both plans are stable, i.e., do not change in the plan creation step. Finally, operators reserve their anticipated resources (e.g., the bandwidth) on all migration targets in the plans to ensure the feasibility of the plan, e.g., ω_D reserves resources on b_2.

Plan execution. When a migration needs to be initiated for an operator according to the plan, the live migration system begins by first copying the execution environment and immutable state (step (1) in Figure 4.2). Then it transfers the relevant mutable state, which is required for future selections when the operator starts executing at the migration target (e.g., only one of three possible events in step (2)). Since this state allows to recompute computational state at the target, we don't need to transfer computational state. In the meantime, the operator will continue to execute at its original placement until the transfer of state has been completed. Finally, at the times indicated in the plan, events are streamed

to the new placement (step (3)) and the resources in use by an operator at its original host are released (step (4)).

4.3.2 Basic Migration Plans

The quality of a created plan depends on the accuracy of the anticipated information about the event loads, latencies, and locations; in fact unforeseen behavior may degrade the performance properties of our system and in the worst case even violate constraints. For now, however, to understand the basic principle of the proposed migration scheme we will treat this information as if it was accurate.

Defining and creating a plan is still challenging in this simplified setting. Each plan of an operator highly depends on the future placements and migrations covered by its neighbor's plan. For example, ω_F's plan can be found when our system knows that ω_D of Figure 4.2 migrates at t to b_2, stays there for at least 30 s, and sends events with a cumulative size of 30 MB to ω_F. In fact, this implies a migration of ω_F at t to b_2, since by migrating ω_F's state of 25 MB, the costs for streaming events from ω_D to ω_F can be averted. Observe, however, that knowing about more migrations of ω_D helps to make a better migration decision for ω_F. For instance, when ω_D migrates back to b_1 after 35 s, ω_F could have stayed at b_3 (the former optimal placement while ω_D was hosted at b_1) to avoid migration costs back and forth. Furthermore, for scalability, ω_F must locally decide if the network latency between ω_F and its neighbors like ω_D as well as its processing latency still obliges the global latency restriction LR_u. Moreover, in many cases it is practically not feasible to consider all brokers of a large-scale network at all times as future migration targets of an operator for a plan. In addition, plans of operators need to be deterministically coordinated. Recall, ω_F and ω_D exchange their plans, since the creation of each of these plans depends on the other plan. However, neighbors in the operator graph can create plans in parallel, which can lead to cycles if not carefully considered, e.g., ω_D and ω_F constantly decide to migrate to the others host.

In this context we will therefore answer:

- How far and at which granularity do we have to plan migrations ahead?
- What are possible migration targets that can be incorporated in the migration plan of a single operator without hurting the overall latency restriction?
- How should the migrations of multiple operators be best coordinated?

We address this by first proposing the basic Migration Plan model, which allows studying migrations at varying granularities. In order to select possible migration targets, we propose the *time graph model*, that allows finding migration targets, depending on the plans of other operators. The proposed coordination algorithm negotiates the plans with dependent operators and migration targets, in order to find minimum cost plans for each operator and reserve resources to ensure the execution of an operator at all times.

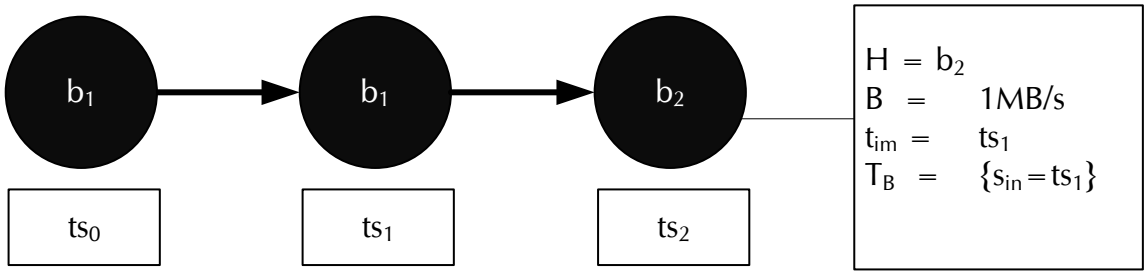


Figure 4.3: Basic Migration Plan for ω_D. At the time steps ts_1 and ts_2 it is hosted at b_1. The node for ts_3 is exemplary labeled with the expected placement. At ts_3 ω_D is anticipated to be hosted at b_2. The average bandwidth for streaming outgoing events between ts_1 and ts_2 is 1 MB. Immutable state and the incoming queues of ω_D need to be migrated from ts_1 on to b_2.

Data Structures

We now focus on the relevant models for the plan-based migration approach. First, we clarify the plan's model. Second, we introduce our model for local latency restrictions which allows our system to ensure the overall latency restriction LR_u. Third, we detail a data structure, called time graph, which is used to create an operator's plan. In a subsequent section, we will detail algorithms to create these data structures.

Migration Plan Model. A plan describes the upcoming migration behavior of an operator ω, determining when it is going to be hosted at which broker, as well as its expected resource requirements. In other words, it is the trajectory of an operator in the broker hierarchy. As depicted in Figure 4.3 for ω_D of the example, the plan provides an expected placement of an operator for a sequence of discrete time steps $ts_i \in TS = \{ts_0, ..., ts_{max^t}\}$. An expected placement is a quadruple $ep = (Ho, BW, t_{im}, T_B)$. Ho is the expected host of the operator, e.g., broker b_1 or b_2. BW is the average expected bandwidth of the operator's outgoing stream, e.g., 1 MB in the time period between ts_1 and ts_2. The time t_{im} is the one at which the transfer of immutable state has to be initiated, e.g., at ts_1 from b_1 to b_2. T_B is a set of deterministic starting points for each buffer in B from which events on the mutable state has to be transferred, e.g., all events with a timestamps greater than ts_1 from ω_D's incoming buffer for source streams.

Since consumers and sources are constantly connected to leaf brokers, we can use the same abstraction to model and share their movements. For example, the plan indicates for a mobile sensor to which leaf broker it will be connected to and the expected size of its sensor stream. The only difference for dynamic interest queries is that range queries with respect to processing interests can span several leaf broker, which requires extending ep by capturing Ho and BW as a set of hosts and bandwidths.

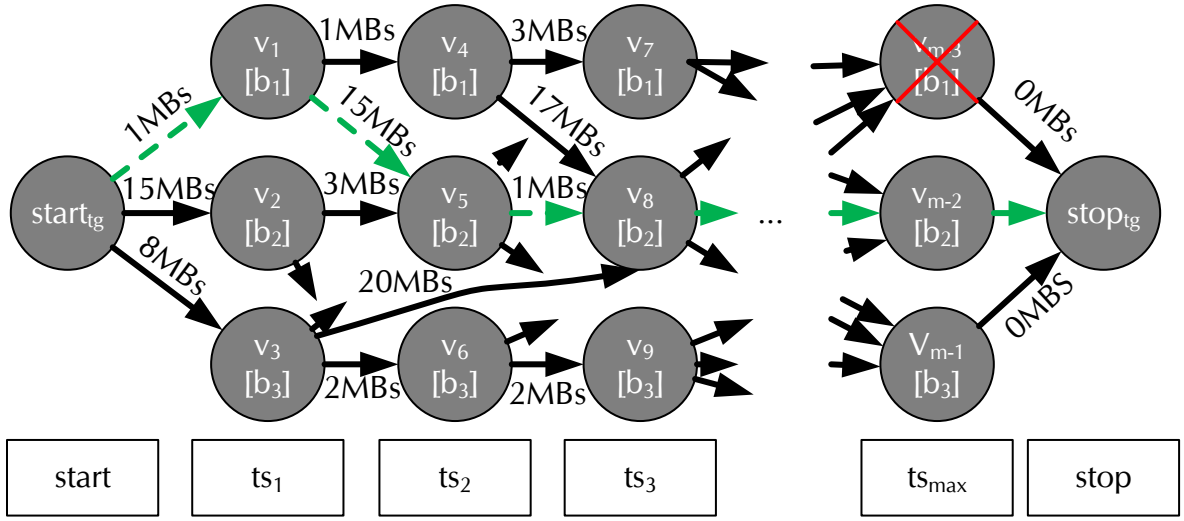


Figure 4.4: Timegraph Example for ω_D. Vertices represent possible future placements of operators at brokers. Edges represent possible future migrations between hosts. Vertices which represent future placements that are not expected to ensure the latency restriction like v_{m-3} are omitted from the time graph.

Latency Restriction Model. Our system supports a simple latency restriction model which allows operators to ensure locally that the global latency restriction LR_u is met. In particular, it relies on the observation that the sum of all delays on a path from a source to a consumer in the operator graph makes up the overall end-to-end delay. In turn, the global latency restriction can be decomposed into *local latency restrictions* for each operator and stream on an existing path from a source to a consumer. For example, let 9 ms be the overall restriction. Moreover, ω_F and ω_D from the example depicted in Figure 4.2 are assigned a local latency restriction of 3 ms and streams like the one from ω_D to ω_F are assigned a restriction of 1 ms. Any placement of ω_D where its host ensures an average processing delay below 3 ms, an average network delay below 1 ms to the host of ω_F, and an average network delay below 1 ms to the source ensures the global latency restriction LR_u. The model allows for sophisticated algorithms that trade latency restrictions which, however, are not in the focus of this work, e.g., if ω_D is hosted on a computational powerful host it could give 1 ms of its restriction to the stream.

Time Graph. The *time graph* $= \{V_{tg}, DE_{tg}\}$ (see Figure 4.4) is a data-structure that allows for each individual operator ω to identify costs and durations of future migrations and placements. Note, that even not performing a migration imposes a cost by streaming events over the in- and outgoing streams. The time graph for a single operator $G \in \Omega$ comprises migration targets which are suitable to fulfill the constraints of a placement.

Each vertex $v_{tg} \in V_{tg}$ represents the possible placement of ω at a broker b, starting at time step $ts_i \in TS$ and ending at the next time step $ts_{i+1} \in TS$. For example vertex v_1 in Figure 4.4 is labeled with b_1 at ts_1, which represents the placement at broker b_1 starting at ts_1. Two special vertices which do not indicate future placements, labeled as $start_{tg}$ and $stop_{tg}$ in Figure 4.4, represent the start and end of all possible sequences of migrations and placements. The absence of vertices at time-steps, e.g., v_{m-3}, indicates that the corresponding broker does not have enough resources to execute ω or is not expected to preserve the latency restriction at that time step.

Each directed edge $de_{tg} = (v_j, v_k) \in DE_{tg}$ represents the migration between brokers or no migration if de_{tg} connects the same broker. Furthermore, an edge indicates that a migration starting at time step ts_j of v_j is expected to be completed at time step ts_k of v_k. For example, when the migration from b_3 to b_4 is started at time ts_1, then all state is expected to be transferred by ts_3 and ω can start processing.

The edge weight $w_{j,k}$ of $(v_j, v_k) \in DE_{tg}$ represents the costs that are expected to occur in the time interval $[ts_j, ts_k)$—the average bandwidth-delay products weighted with that interval (see Section 4.2). This comprises the expected migration costs $C_{Mig}(v_j, v_k)$, the costs for transferring state from the broker represented by v_j to the broker represented by v_k. These costs depend on the anticipated size of ω's state as well as the bandwidth and the pair-wise network latency between all brokers involved in the migration. It also comprises the expected placement costs $C_{pla}(v_{tg})$ at the broker of v_j during $[ts_j, ts_k)$, i.e., the costs for transferring events over a network link from and to expected placements of neighbors of ω during that time. The latter costs depend on the anticipated size and number of events as well as the bandwidth and the pair-wise network latency between all brokers involved in transferring these events according to the expected future placements of neighbors in the operator graph. To this end, the costs vary with the placement and connection of a neighbor, e.g., it changes drastically for all vertices at ts_1 since this time step coincides with the time when the vehicle in Figure 4.2 changes its access point. The edge weight of 1 MBs between two subsequent vertices of b_1 are the placement costs during that time step, e.g., the accumulated size of transferred events is expected to be 1 MB when the time span between subsequent time steps is one second. Note that the placement costs are not only considered in the case that an edge represents no migration, but also if it represents a migration. This is because ω still processes events at the previous broker, while the migration happens in the background. The edge weight of 20 MBs between v_3 and v_8 comprises the placement costs of 4 MBs, since ω will still process events at b_3 until the migration of ω's state has finished at ts_3 on b_4. In addition, it comprises migration costs of 18 MBs for transferring state from b_3 to b_4.

Migration Plan Creation for Consumers and Sources

Recall, non-executable plans of mobile objects convey the predicted movement of consumers, sources, or focal objects in a unified way. These plans are the input to the plan creation of leaf operators like ω_D in Figure 4.2 and this way initiate a mobility-aware planning of migrations and placements. Our system creates the non-executable plans on

Algorithm 4.1 Creating a Migration Plan

```
1: Requires: ω //operator
2: Defines: stable_plan, tmp_plan, neighbor_plans, TG ← ∅

3: upon trigger generate_plan( )
4:  if not_in_await_state() then
5:      await not_wait_for_feedback() ∧ no_target_coordination() //in await state
6:      TG ← create_time_graph()
7:      path ← determine_shortest_path(TG)
8:      plan ← create_initial_plan(path)
9:      if plan deviates from tmp_plan then
10:         tmp_plan ← plan
11:         trigger send plan_updated_msg(plan) to all neighbors of ω
12:     else
13:         try_init_target_coordination()
14:     end if
15:  end if
16: end

17: upon timeout(timer in regular intervals)
18:  if check_monitoring() ∨ check_latency_restriction() then
19:     trigger generate_plan()
20:  end if
21: end
```

behalf of mobile objects. In particular, the plan creation for a source and a consumer is triggered after the initial deployment of an operator graph. Afterwards, the plan creation is triggered when $|TS|$ time steps have passed since the last plan creation, or connection patterns to leaf brokers and streaming patterns of sensor streams deviate from the ones recorded in the plan. TS is a set of time steps, a system parameter, and can be tuned by a system administrator for situations where knowing about more leaf brokers to which the consumer or source will connect to allows to create better plans for a leaf operator.

If the future connection patterns and streaming patterns are known beforehand, a plan for a consumer or source comprises for each time step $ts_i \in TS$, starting from the current time now on, exactly one expected placement ep. For each time step ts_i the expected host ($Ho(ep)$) is set to the leaf broker of the consumer or source at time $ts_i + now$ and the expected bandwidth is determined from the streaming patterns ($BW(ep)$). By approximating the time t_h for a hand-over between subsequent leaf brokers of the connection pattern $t_{im}(ep)$ can be set to $ts_i + now - t_h$, while each $t_B \in T_B(ep)$ is set to $ts_i + now$.

Migration Plan Creation for Operators

We now detail a distributed algorithm for the creation of a basic executable plan for an individual operator ω. We first outline the basic algorithmic steps by means of Algorithm 4.1. The plan creation algorithm finds a shortest weighted path in a time graph (*TG*) that is local to ω and generates an executable plan (*stable_plan*) from this path. We then focus on how the time graph is created to model various combinations of future placements and migrations (see Algorithm 4.2) and the rules to transform a shortest weighted path to a plan. The time graph is maintained using, among other predictions, the set of plans from all neighbors in the operator graph (*neighbor_plans*), which allows our system to calculate costs at discrete future time steps based on the neighbors' expected placements at that time. In turn, ω's plan is used to maintain a time graph for each of ω's neighbors in the operator graph. To this end, this section also details the exchange of plans between ω and its neighbors as part of the *plan coordination*. To ensure that migration targets have sufficient resources, our system also reserves resources at those migration targets.

Basic Algorithm. The first plan is created after the deployment of an operator. Afterwards, the plan creation is triggered concurrently when latency restrictions are no longer preserved on the current broker, predictions on the outgoing bandwidth and collected latencies deviate beyond a threshold (Lines 17-21) from previous predictions, or the plan of the neighboring operators changes (Algorithm 4.3).

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 6)—*TS* is a tuneable system parameter. The operator graph's organization allows us in a subsequent step to find an initial plan by determining the shortest path. This path contains a sequence of migration targets and times to migrate between them, where the expected overall network utilization is low and the latency restrictions are expected to be preserved (Lines 7-8). We replace a previous executable plan with the newly found plan only if this new plan deviates in the expected placements at any time step and has lower expected placement and migration costs for the next $|TS|$ time steps (Line 9). However, before implementing the new plan it is consolidated by coordinating it with the neighboring operators (Line 11) and the brokers that are selected as migration targets. Migration targets might not have enough resources to execute the plan or neighbors adapt their own plan due to the plan that is to be consolidated. In both cases another iteration of the plan generation is triggered (Algorithm 4.3).

Impact of TS on the shortest path: A large *TS* induces a high uncertainty for time steps that are far in the future. Note, however, in a time graph edge weights that include migration costs are typically higher than for edges between the same broker, since both migration costs and placement costs are included. For example, the migration represented by the edge from v_3 to v_8 in Figure 4.3 has a weight of 20 MBs, while the combined weight of the edges combining v_3 over v_6 to v_9 is 4 MBs. Therefore it is unlikely that an edge

Algorithm 4.2 Time Graph Maintenance

1: *Requires:* neighbor plans NP, anticipated event load AL, latencies LT, and broker BR
2: *Defines:* `Time-graph` $TG = \{V_{TG}, DE_{TG}\}$
3: **function** createTimeGraph()
4: $Targets \leftarrow$ get_possible_targets(NP, AL, BR)
5: $TG \leftarrow$ create_start_and_stop_vertex()
6: **for all** $ts \in \{t + now | t \in TS\}$ **do**
7: $enforce\_restrictions \leftarrow false$
8: **repeat**
9: **for all** $b \in Targets$: should_create_vertex$(b, ts, enforce_restrictions, NP, LT)$ **do**
10: $V_{TG} \leftarrow V_{TG} \cup \{(b, ts)\}$ //create vertex for b at ts
11: **for all** $b_p \in Targets$: edge_condition$(TG, b_p, b, ts, ts_{mig}(b_p, b))$ **do**
12: $de \leftarrow ((b_p, ts_{mig}(b_p, b)), (b, ts))$ //create edge spanning migration
13: $DE_{TG} \leftarrow DE_{TG} \cup \{de\}$ //add edge to time graph
14: $TS' \leftarrow \{t' | t' \in TS \wedge (ts_{mig}(b_p, b) \leq t' < ts + now)\}$
15: $w_{de} \leftarrow C_{Mig}(de) + \sum_{t' \in TS'} C_{Pla}(b_p, t', NP, AL)$ //calculate edge weight
16: **end for**
17: **end for**
18: $enforce\_restrictions \leftarrow true$
19: **until** $\exists(*, ts) \in V_{TG}$
20: **if** $\nexists(*, ts) \in V_{TG}$: reached$(*, ts)$ **then**
21: $TG \leftarrow$ add_from_preceiding_TS(TG, ts, TS)
22: **end if**
23: **end for**
24: **return** TG
25: **end**

from any broker b_i to another broker b_j is considered for the shortest path if only few time steps are included, even if a potential migration target promises lower placement costs. For instance, if the time graph in the example would stop after ts_3, the migration from b_1 to b_2 would not be captured by the shortest path, which would include v_4 instead of v_5. To this end, TS needs to be configured according to the scenario by a domain expert.

Maintaining the time graph. Algorithm 4.2 outlines the creation of the time graph TG. The algorithm relies on the current set of plans from neighbors in the operator graph (NP) and anticipated event loads (AL) for streams to these neighbors, which allows our system to calculate costs based on future placements of the neighbors. Additionally, pair-wise latencies (AL) between topological close brokers allow our system to anticipate if local latency restrictions are met. Furthermore, anticipated resources on these brokers (BR) allow our system to ensure that sufficient resources are available to host the operator.

At first, a set of brokers that have in general enough resources to execute the operator are selected as possible future migration targets (Line 4). Then it populates an empty

graph with the start and a stop vertex (Line 5).

In the following steps vertices for each combination of discrete time steps *TS*, starting at the current time *now*, and possible migration targets (*Targets*) are created (Lines 6-23). However, vertices are omitted when the associated broker comes not into consideration to host an operator at *ts* (e.g., v_{m-3} in Figure 4.4). This is the case when latency restrictions are not expected to be preserved according to the operator's neighbors placement in their respective plan. It is also the case if a local information on the migration targets expected available resources indicates that not enough resources to execute the operator at the potential migration target are available.

When the vertex $v = (b, ts)$ is created, the algorithm anticipates which migrations are possible and which previously created vertices are therefore connected to v over a directed edge (Lines 11-16). This requires to estimate the size of the mutable state, according to the average size of the buffers and the time t_{mig} that it takes to migrate all state between any possible host b_p and the host b of v. Since t_{mig} may span over more than one time step, the vertex that represents b_p at time $ts - t_{mig}$ and is labeled $(b_p, ts_{mig}(b, b_p))$, is selected as source of the edge *de*. However, the operator has to continue its execution at b_p during the migration, which means that all vertices labeled with b_p between $ts_{mig}(b, b_p)$ and ts have to exist (Line 15).

The weight of this new edge *de* is the sum of the placement costs $C_{pla}(v')$ of all vertices $v' = (b_p, t')$ representing b_p during the time steps spanned by *de*—the sum over the average bandwidth-delay product between b_p and the expected placement from the neighbors plan weighted with the time intervals during the given time steps—and *migration cost* $C_{Mig}(de)$ between b_p and b (see Line 15). Note that edges that connect vertices which are labeled with the same broker and therefore do not indicate a migration are considered a special case, where the migration costs are zero.

The time graph can be partitioned due to omitted vertices, and, as a result, no shortest path and plan can be found by Algorithm 4.1. To overcome such situations, we purposefully break latency restrictions. If no vertex is found that is expected to assure the latency restriction for a time step *ts* (Line 19), vertices are created for brokers that are either expected to preserve the latency restriction for at least one incoming or outgoing stream of ω or have been selected by previous plans to host ω. Moreover, to ensure that at least one vertex is reachable in each time step, i.e., at least one connected path from $start_{tg}$ to a vertex of the time step exists, the algorithm tracks the reachability starting from $start_{tg}$. To this end, vertices are marked reachable if they have an edge originating at $start_{tg}$ or an edge originating at a vertex that has been marked as reachable. If no vertex is marked as reachable for a time step, we ensure that at least one connected path from $start_{tg}$ to $stop_{tg}$ exists. In particular, we add a vertex v' for any broker that is represented by a reachable vertex v in the immediately preceiding time step and add an edge (v, v')(see Line 20).

Properties: The number of brokers that are selected in Line 4 of Algorithm 4.2 has a severe impact on the performance of the live-migration system. Modeling all brokers, in

a large-scale scenario with thousands of broker, is computationally costly, because the number of vertices and edges grows cubically. The number of vertices, per time step in the time-graph, grows linearly within each time step with the number of brokers n_b, thus the complexity is $\mathcal{O}(n_b\ |TS|)$. More severe is the number of edges, which grows quadratically per time step, because we connect each broker with all other brokers of the next time step. Thus the complexity is $\mathcal{O}(n_b^2\ |TS|)$, which is cubic if the number of brokers n_b equals $|TS|$. Furthermore, each broker has to acquire information about the latency, bandwidth and available resources of these brokers, to determine the placement costs, which can only be determined by regularly sending messages between the broker.

Although selecting all brokers as migration targets gives a near-optimal solution to the mobile migration problem if the prediction is perfect, we decrease the network and computation costs by only selecting the most relevant brokers. To increase the scalability, we utilize the spatial-temporal locality property. Mobile objects move only within local bounds, e.g., do not connect to the system from London and seconds later from New York. Hence, good future migration targets are found close to the current broker that hosts the operator. Hence, each broker keeps coarse grained information on brokers responsible for far away locations, and fine-grained information on nearby brokers.

Initial Migration Plan. In order to extract an initial plan (see Line 8 of Algorithm 4.1) we find the shortest path in the time-graph (the dashed line in Figure 4.4). At each time step of the shortest-path a broker is determined that represents the expected host ($Ho(ep)$) for an expected placement ep of the plan. Expected values for the bandwidths ($BW(ep)$) are taken from the bandwidth estimation for the operators outgoing stream. Furthermore, the starting time for the migration of immutable state ($t_{im}(ep)$) is computed for each time ts_i by subtracting the migration time t_{mig}. The first event of each buffer in B ($T_B(ep)$) is then captured according to the window-model (see Chapter 3). Those windows determine the set of events in each buffer that are required for the correlation at a discrete time, therefore if we model the windows position at ts_i we can approximate the first required event from the windows bounds.

Coordination. Recall, since plans of operators are created based on plans of neighbors in the operator graph, they inform one another about changed plans in a coordination step. Such plans can convey an increase in the network distance with respect to the expected placements in the neighbors' plan. To this end, operators and their neighbors reevaluate their plan based on the others changed plan until both found a plan with low expected migration and placement costs. As a result of this *neighbor coordination*, mobility related changes of leaf operators ripple deeper into the operator graph. For example, a changed plan from the vehicle depicted in Figure 4.2 triggers the creation of a changed plan for ω_D, in turn, ω_D informs ω_F about its changed plan. To ensure that sufficient resources are available at a migration target, ω needs to also coordinate the plan with them (*target coordination*).

Algorithm 4.3 Coordination of a Migration Plan

```
1: upon receive plan_updated_msg(plan) from neighbor ω'
2:   neighborPlans[ω'] ← plan
3:   trigger generate_plan()
4: end

5: upon receive plan_acceptedMsg()
6:   try_init_target_coordination()
7: end

8: function try_init_target_coordination()
9:   if allFeedbackReceived() ∧ noPlanGeneration() then
10:      check_need_send_plan_accepted_msg()
11:      check_need_send_reservation_request(tmpPlan)
12:  end if
13: end

14: upon receive reservation_reply_from_targets(tmpPlan)
15:  if tmpPlan is executable on all targets then
16:      stablePlan ← tmpPlan
17:  else
18:      tmpPlan ← ∅
19:      trigger generate_plan()
20:  end if
21: end
```

Neighbor coordination: An operator starts the coordination of plans by sending the new plan to each neighbor (see Algorithm 4.1),[1] e.g., ω_D sends the plan depicted in Figure 4.3 to ω_F. The operator waits (see Algorithm 4.3) until it received a feedback message from each neighbor which indicates that they accept the new plan, i.e., the new plan does not affect their current plan. Then it starts a coordination with the selected targets, subsequently implements the new plan, and leaves this wait state (Lines 16-19). If the new plan affects the current plan of a neighbor, their feedback is a changed plan. Note, that we consider a plan that was sent concurrently by a neighbor like ω_F and is still in transmission when the plan coordination was started by an operator like ω_D as feedback, which, as we will discuss, is a prerequisite to avoid deadlocks.

When a neighbor receives a plan, it reevaluates its own plan before it decides on a feedback (Line 3). If the newly determined plan deviates from the current plan, the operator starts to coordinate its own plan, by sending a changed plan of its own as feedback. Otherwise, the operator will send the feedback to all neighbors that recently updated their plans that it accepts their plans (Line 10). To reduce the number of coordination steps when neighbors concurrently generate plans, we ensure that a plan is

[1] Note, we assume a monitor semantic and each function is executed atomically.

only generated if all feedback is available and the plan is not coordinated with migration targets (Line 8). To avoid cycles when operators like ω_F and ω_D concurrently generate new plans and constantly decide to migrate to the other's expected placement due to the other's changed plans, we ensure that only one of the concurrently sent plans triggers a plan reevaluation. In particular, we assign all operators in the operator graph unique ids, e.g., id(ω_F) = 1 and id(ω_D)=2. This way, we can repress concurrently sent plans of the operator with the lower id at the operator with the higher id. Only when ω_F sends a feedback to ω_D as a result of the concurrently sent plan of ω_D, ω_D will reevalute its plan. In the analysis we show that this coordination terminates.

Target coordination: The coordination with migration targets happens in two steps. First, a request is sent to all migration targets, asking if (i) they have enough resources available to host the operator (ii) they can ensure the local latency restrictions. If this is not the case, the plan is rejected and re-evaluated at the initiator of the coordination, with a time graph that does not create vertices for brokers that rejected the plan.

Although, these conditions are already checked when the time-graph is created, other plans of other operators compete for the resources. Therefore, plans have to reserve the resources at the migration target in a second step. This enables a broker to approximate its future resource utilization. If the future migration targets specified in the plan can execute the plan the resources are reserved and previous reservations of the previous plan are revoked.

Stability: Updated plans of neighbors bear the potential to make the plan of an operator unstable if the communication characteristics keep changing (e.g., diverging bandwidths) and can not be perfectly predicted. For example, if the operator generates a new plan with each location update of a source that instantly triggers a migration of an operator to a new host, we arbitrarily increase the network utilization. To ensure that a plan does not continuously change, only because of a small deviation in the measured bandwidth or latency, edge weights are only replaced in a time-graph if these measurements change beyond a threshold.

4.3.3 Uncertainty-aware Migration Plans

So far, we presented plans as sequences of definite placements. However, since these sequences represent the predicted future, it comprises various uncertainties. Sources and consumers can change their movement pattern, e.g., vehicles changing their routes, or they hand-off between adjacent leaf broker at unexpected times. Data rates of event streams can also vary, e.g., when an operator detects less events than expected.

The system's possibilities to capture the future movement of mobile objects, highly influences the accuracy of the anticipated connection pattern to a leaf broker. In the remainder, we consider three possibilities to capture the future movement: a dead reckoning based mechanism, a navigation system based mechanism, and a learning based mechanism. Simple dead reckoning mechanisms predict future locations based on the

direction and speed of the last few reported locations, which is typically only accurate for nearby locations. Recall, leaf brokers are responsible for mobile objects in a distinct spatial region (see Chapter 2). Therefore, the next sequence of leaf brokers and when a source or consumer connects to them is highly uncertain. If the future path of a mobile source or consumer from a navigation system is at hand, the system can accurately determine the sequence of leaf brokers. However, it needs to predict the times when a mobile object changes its connection to another leaf broker based on the expected velocity of the vehicle. Learning typical connection patterns of vehicles between neighboring leaf brokers also gives a more accurate view than dead reckoning. For example, while driving on a highway vehicles will connect with high probability to the same sequence of leaf broker that cover subsequent sections of that road and with a low probability take the next exit that may require to connect to another leaf broker. A concrete exit, however, can be known by a navigation system, making its determined path more accurate than the learned connection pattern.

Since data rates of event streams can vary, it is possible that placing an operator ω at its planned migration target imposes higher costs compared to an optimal placement. In case the placement costs dominate over the migration costs, it is beneficial to stall the decision for a concrete migration target until it is certain that the placement can be improved. However, dependent operators, e.g., with large state, now require the information about all possible migration targets of ω to plan their migrations.

In this context we have to answer:

- What is a meaningful representation of an uncertain connection and migration pattern that can be used to model an uncertainty-aware plan?
- What are the best migration targets to incorporate into a plan that reflects uncertain migrations?

We address the question of the uncertainty-aware representation by expanding the plan to a *Markov chain*. The second one is addressed by a set of heuristics that aim to minimize the expected overall migration and placement costs of a single operator.

Uncertainty-aware Migration Plan Model

An uncertainty-aware plan is represented as *Markov chain*, as depicted in Figure 4.5. It allows for more than one expected placement at the same time step, connecting adjacent time steps with probabilistic state-transitions. We distinguish two kinds of uncertainty-aware plans. First, *temporally skewed migrations*, describe that instead of migrating at a fixed time step a mobile consumer, mobile source, or operator will initiate the migration within a temporal interval of discrete time steps. For example, a vehicle's connection change from broker b_1 to b_2 according to the plan in Figure 4.5 is performed with probability 0.5 at time ts_1 and with probability 0.5 at ts_2. Second, *spatially skewed migrations*, allow operators to describe multiple placement possibilities at the same time step, e.g., if a transition from b_1 at ts_0 to b_3 at ts_1 in Figure 4.5 were included.

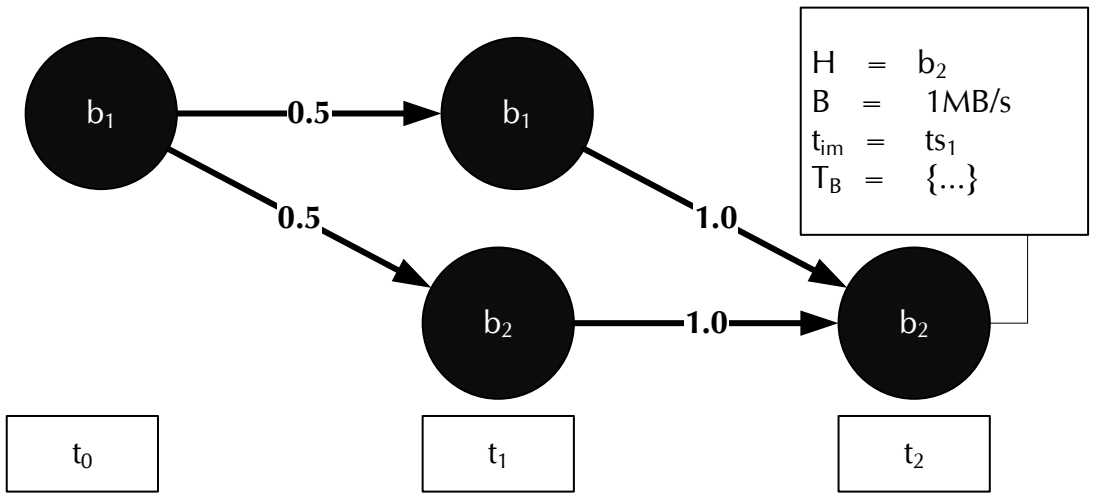


Figure 4.5: Uncertainty-aware Migration Plan

Creating an Uncertainty-aware Migration Plan

This section covers how the system creates uncertainty-aware plans for a consumer or source to indicate the possible future leaf brokers and its data rates. Furthermore, upon receiving uncertainty-aware plans, an operator has to adapt its own plan. To this end, it can select from a pool of policies that aim to minimize the expected bandwidth-delay product but vary in their complexity.

Planning for consumers and sources

Dead reckoning methods and navigation systems provide a set of time-stamped uncertain future locations[2] for mobile sources and consumers. This allows us to determine the expected placements ep of the plan for a set of future time steps $ts_i \in TS$. Expected hosts ($Ho(ep)$) are all future leaf brokers responsible for the future location at ts_i. The average over the monitored recent event traffic determines the expected bandwidth ($BW(ep)$). Probabilities for state transitions from previous time steps ts_{i-1} are estimated based on the proportion of the overlap of the uncertain location with the leaf brokers area weighted by the extent of the uncertain location.

This method changes for learned connection patterns of consumers and sources. These learned patterns describe how probable it is for a source or consumer to change its connection to another leaf broker after it stayed connected for a certain amount of time to its current leaf broker. The key idea is to maintain these patterns in a graph data structure which allows our system to determine expected placements for plans by traversing the graph.

In a graph representing learned connection patterns, each vertex represents a broker and has edges to all vertices representing neighboring brokers the source or consumer

[2]Note, this can be a typical GPS normal distribution

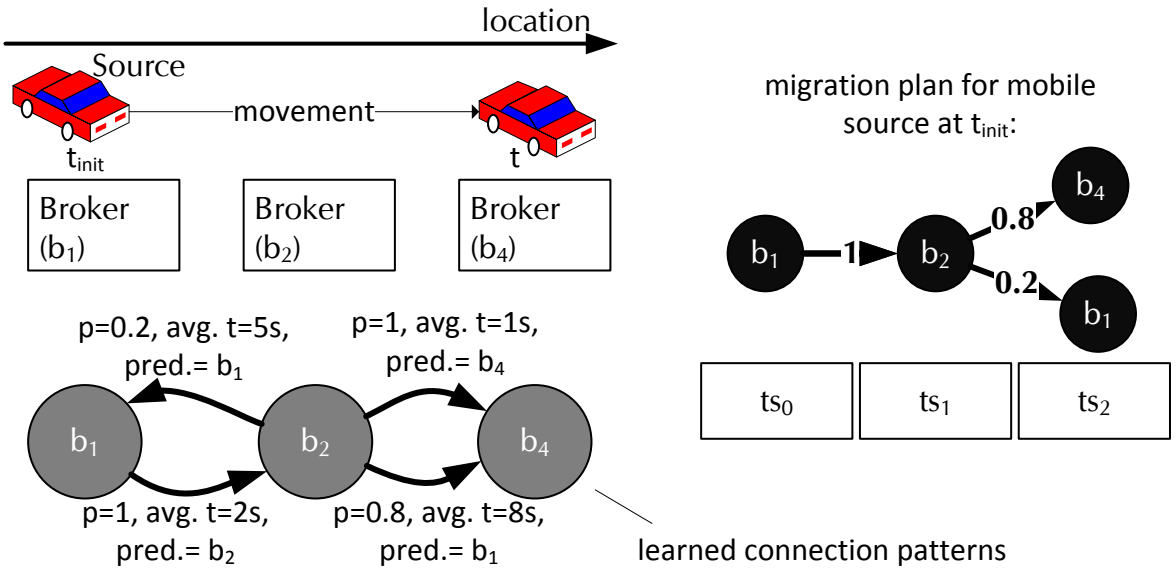


Figure 4.6: Learned connection patterns between b_1,b_2, and b_3 are represented in a graph. For example, all vehicles change their connection from b_1 to b_2 after an average time of 2 s. A plan for the vehicular source can be created by traversing the graph from the vertex representing b_1.

can connect to next. A graph for an example depicted in Figure 4.6 would comprise three vertices, one for each leaf broker (b_1, b_2, and b_4), where b_2's vertex would be connected to both neighboring brokers since passing vehicles first connect to b_1, then b_2, and then b_3. Each edge in the graph is annotated with the probability to change the connection and the time a source or consumer is connected to a broker before it connects to the neighbor, e.g., a mobile object in our running example stays on average 2 s connected to broker b_1 before changing the connection to b_2. To represent directional information, we also maintain several edges with respect to previous brokers. This way, we can indicate for a connection change from b_2 to b_4 that a mobile object's connection must have changed to b_2 from b_1 and not b_4 when vehicles drive along a road.

Such a graph can then be exploited to find a sequence of expected hosts $Ho(ep)$ for an uncertainty-aware plan. A simple algorithm follows paths in the graph sequentially from broker to broker for $|TS|$ time steps, starting from the current leaf broker of a mobile source or consumer. For each following time step ts, the algorithm uses the annotated times of the graph's edges to identify which broker the mobile source or consumer can be connected to and adds corresponding expected placements to a probabilistic plan. In our example, the source is currently connected to b_1. After a 5 s time step it is expected to be connected to b_2, since the vehicle stays connected with b_1 for an average of 2 s. Since after another 10 s time step the vehicle is expected to have changed the connection to eihter b_1 or b_4, two expected placements are added. State transition probabilities of the probabilistic plan are derived form the probability that a connection changes.

Policies of operators

In the following section we present several policies to process and create uncertainty-aware plans.

Naive Policy: Our naive approach exploits that neighboring operators in G of an operator ω commit themselves to a finite number of expected placements and migrations with their uncertain plans. Therefore, we identify for an operator ω all possible placements and migrations based on the uncertain plans of its neighbors in the operator graph. In particular, our approach is to decompose the probabilistic plans ω received from its neighbors into basic plans. These plans allow our system to calculate near optimal basic plans for ω for all distinct sequences of migrations and placements of the neighbors. A composition of ω's basic plans then represents the probabilistic plan for ω.

Observe, each sequence of state transitions in an operator's probabilistic plan describes exactly one possible sequence of placements and migrations. For example, if the vehicle from our running example created the plan shown in Figure 4.5, it indicates to ω_D to either connect to b_2 at time step ts_1 or ts_2. A probabilistic plan can therefore be decomposed into individual basic plans for each sequence. For example, the vehicle's decomposition results in basic plans for the sequence $seq_1 = \{b_1,b_2,b_2\}$ and sequence $seq_2 = \{b_1,b_1,b_2\}$. The latter basic plan is structurally similar to the one depicted in Figure 4.3. The product of all state transition probabilities of the sequence defines the probability that the neighbor is actually following the specific plan, e.g., 0.5 for both example sequences.

For each combination of decomposed basic plans which comprises exactly one plan of each neighbor of ω our system then determines a basic plan for ω using the method described in Section 4.3.2. For example, consider ω_D of our running example also received a basic plan from ω_F, comprising the sequence $seq_3 = \{b_3,b_3,b_3\}$, in addition to the probabilistic plan depicted in Figure 4.5 from the vehicle. In this case, plans are created for ω_D by (i) using the combination of a plan represented by seq_1 and a plan represented by seq_3 as well as (ii) using the combination of a plan represented by seq_2 and of a plan represented by seq_3. Our system creates multiple time graphs to achieve this; in particular, one time graph for each possible combination.

All shortest paths in the time graphs are then combined to a probabilistic plan for ω, where each shortest path represents one sequence of state transitions in the resulting plan. Each sequence's probability depends on the probabilities of the decomposed plans used to create it. Consider that the two resulting basic plans in the example comprise the sequences $seq_4 = \{b_1,b_1,b_2\}$ and $seq_5 = \{b_1,b_2,b_2\}$. Since the probability for ω_F to follow the basic plan is 1 and the probability of the source to connect according to the sequence seq_1, respectively seq_2, is 0.5, each of those resulting sequences has the probability of 0.5 to represent the actual migration sequence of ω_D. When both are combined to a single plan it looks structurally similar to the plan Figure 4.5.

This policy is targeted at operators with small state, where short migration times allow an operator to accurately follow the uncertain movement of a neighbor by creating an

uncertain plan. However, it yields a high computational cost, because it requires to maintain many time graphs.

Weighted Sum: Our second approach finds a basic plan for an operator ω, which yields the best expected overall costs (see Section 4.2) based on the uncertain plans of its neighbors in the operator graph. To this end, we determine the basic plan as discussed in previous sections, however, we modify the cost function to determine the placement costs C_{pla} for the time graph in order to comprise expected costs.

Recall, the weight of each edge in the time graph of an operator ω is determined by the costs for the event stream connecting ω to its neighbors in the operator graph. This basic approach exploits that neighbors commit themselves to be hosted at a distinct broker between subsequent time steps, say ts_u and ts_v, as indicated by the corresponding expected placement in the basic plan. However, a probabilistic plan allows neighbors to blur the concrete placement between time steps, i.e., it allows to indicate more than one expected placement, e.g., an operator with the plan depicted in Figure 4.5 indicates that it is either hosted at broker b_1 or b_2 at t_1. These expected placements are then reflected in the calculation of an expected placement cost as follows:

Using the plan's Markov chain, the system can calculate the probability of occurrence $P(ep_n, ts_u, ts_v)$ for each of the expected placements ep_n of a neighbor $\omega_n \in \Omega(\omega)$ between ts_u and ts_v, e.g., 0.5 for both expected placements at t_1 in the example. In addition, the system determines for each ep_n of the neighbors plan mp_n placement costs $C_{pla}(ep, b, ts_u, ts_v)$ for placing ω between ts_u and ts_v on broker b. For example, if operator ω_D received the plan depicted in Figure 4.5 from a neighbor, it calculates costs for both ep at t_1. The expected placement costs $C_E(b, ts_u, ts_v)$ between time steps ts_u and ts_v are then calculated by summing up all costs and weighting them with the probability of the expected placement (ts_u and ts_v have been omitted for clarity):

$$C_E(b) = \sum_{\omega_n \in \Omega(\omega)} \sum_{ep_n \in mp_n} C_{pla}(ep_n, b) * P(ep_n)$$

The computational costs are lower than for the naive approach, since only one time graph needs to be maintained. However, at the drawback of sacrificing the accuracy in following the uncertain movement of a neighbor. This policy is therefore targeted at operators with larger state, where migrations need to be started when the neighbors movement is still uncertain.

Changes in data rates: The last policy is the only one that creates an uncertainty-aware plan without previously receiving such plans from neighbors. Since a time-graph comprises estimated costs on future placements and migrations, the actual costs at the time of an estimated migration might differ. Recall that an operator with small state can defer its migration to such a time. In order to reflect these scenarios in the plan, we select the k shortest paths from a single time-graph, since they are the most likely paths to actually

have the lowest costs. For example, due to a deviation in the bandwidth it can be good to also consider the edge from b_1 to b_2 at ts_2 in Figure 4.4.

Each of these k paths represents a sequence of transitions in the plan. The probabilities that these paths represent the actual shortest path are determined by estimating the probability that the estimated cost for one of these paths is lower than for all the other paths and therefore describes the actual migration behavior. We therefore understand the sum of all weights W_{tg} on these paths as random variable X_{tg}. The confidence interval $[\min(X_{tg}), \max(X_{tg})]$ is given by the interval that determines how far bandwidths and latencies can deviate before the time-graph is created anew. However, without the knowledge of the real distribution within that interval, we rely on hints from the operator on the actual distribution, e.g., that the values are distributed according to a rather typical distribution, such as poison or uniform. The continuous solution for the probability, that the j^{th} path has the lowest cost is then:

$$P_{min}(X_j) = \int_{y=\min(X_j)}^{\max(X_j)} P(X_j = y) \prod_{x \neq j} \text{Prob}(X_x, y, j)$$

with

$$\text{Prob}(X_x, y, j) = \begin{cases} P(X_x \geq y) & x < j \\ P(X_x > y) & else \end{cases}$$

For example, the weights for all depicted paths in Figure 4.4 including only brokers of b_1 and b_2 accumulate to the same costs. Therefore the probability for both paths in the interval $[t_1, t_2]$ is 0.5.

4.3.4 Executing a Migration Plan

We now discuss how to execute a plan in order to perform a migration. At first we discuss the basic approach which only migrates immutable and mutable state to build up processing state at the target, then we discuss an alternative approach which requires that operators expose their internal process state.

Basic Migration Algorithm

Algorithm 4.4 allows the operator to implement the migration according to the probabilistic state transitions in a plan. It decides on one of the possible placements from the plan and informs the live-migration system about when to transfer and initialize an operator at the migration target. The algorithm serializes the migration of immutable and mutable state, while it continues processing at the previous broker until the migration target starts processing.

At each time represented by the time step $ts \in TS$ (see Line 1) of the plans, the algorithm checks if any possible migration needs to be initiated. It starts at the state that models the current placement, e.g., b_1 at ts_1 in Figure 4.5. For the next possible migration according

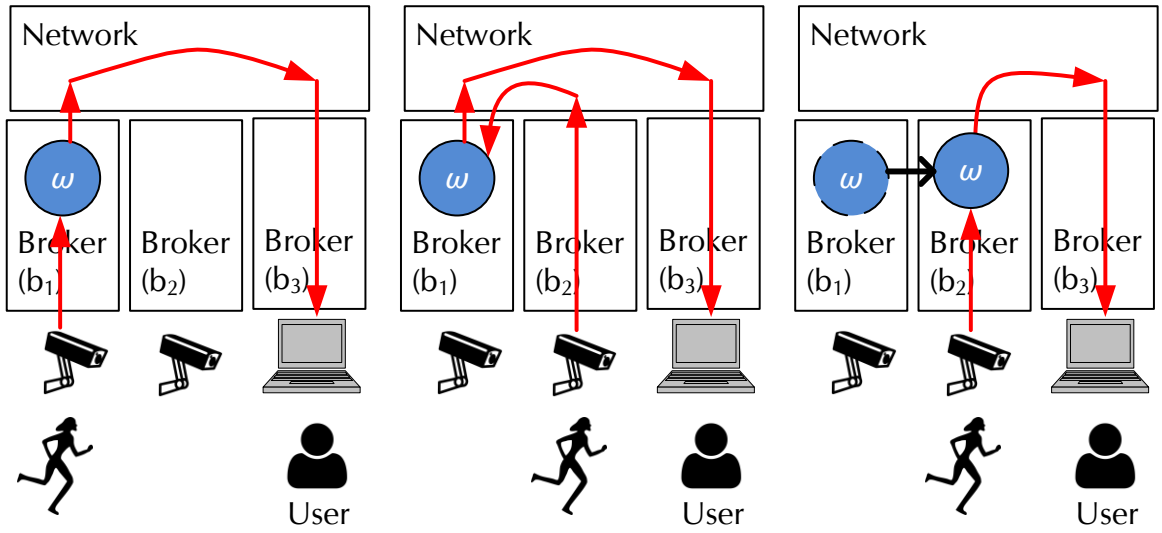
Algorithm 4.4 Execution of a Migration Plan

```
1:  upon timeout migration_timer(time t)
2:   nextMigration ← find_next_target(t)
3:   if nextMigration.t_im = t ∧ ¬stateMigrated then
4:       trigger send immutalbeState to nextMigration.H
5:   end if
6:   for ∀tq ∈ nextMigration.T_B do
7:       if  should_migrate(tq) then
8:           initiate_mutable_state_transfer(tq)
9:           initiate_streaming(tq)
10:      end if
11:  end for
12:  if all_state_at_target(nextMig.t_i)  then
13:      stop_operator()
14:      trigger send start() to nextMigration.H
15:  end if
16: end
```

to this plan, e.g., from b_1 to b_2 at ts_2, the operator checks if it has to start transferring the immutable state before the next time step. To determine the next possible migration for probabilistic plans our algorithm needs selects one sequence of state transitions. Since operators with small migration state do not need to amortize migration costs, they can greedily select the sequence that currently has the lowest expected bandwidth-delay product, e.g., at ts_1 either b_1 or b_2, Other operators can randomly draw one sequence of state transitions where the next migration has not yet started (see Line 2-3). The migration itself starts out by sending the immutable state to the selected migration target (see Line 4). This is skipped if the immutable state already started to be transferred in a previous time step.

In the next step (see Line 6-11), we determine if parts of the mutable state have to be transferred before the next step. This is done by estimating for each buffer B the size of the mutable state that is currently available in all buffers starting from T_B on and how long it takes to migrate these states. In this step, we also inform the neighbors in the operator-graph that they have to start streaming to the migration target. They in return inform the migrating operator about the first event they transferred to the migration target, which indicates when to stop the streaming of mutable state from the current host to the migration target.

In a final step (see Line 12-15), the previous broker stops the processing at the time when mutable and immutable state is available at the migration target to process the next selection (see Section 3.2.2). In addition it informs neighbors in the operator graph that they should stop streaming to the current host, respectively that they will receive streams from the migration target. Finally, it initializes the operator, in a rollback-recovery like fashion [KMR+13] at the target.



(a) Initial setup, optimal placement of operator ω.

(b) Focal object moves, high latency and network utilization.

(c) Migration of the operator ω, the latency and network utilization improves.

Figure 4.7: Migration of friend detection operator, following a mobile focal point.

Including Processing State

Observe that the basic algorithm duplicates event streams during the migration to have all mutable state available on the previous host that is required to process its last selection before the migration, while the migration target has all mutable state available to process its first selection after the migration. Depending on their implementation, however, operators may expose their internal state for each selection (e.g., [CFMKP13]), which can be smaller than the combined size of all events in that selection. Hence, we can allow to partially process a selection at the previous host, only transfer processing state, and then finish processing of a selection at the migration target. To this end, we allow operators to choose between the basic migration strategy and this one.

4.4 Demonstration

The MigCEP live-migration approach has been implemented in a demonstrator [OMK14]. In our demonstrator, we show the impact of different operator migration approaches on the latency and the perceived customer quality (frame rate) with a *video friend finder*.

Friend finder application. Our video friend finder allows users to specify a query that comprises a set of friends, images of these friends, and an initial placement for operators

that can be migrated in a target region. In turn the application provides live video streams on which the friends are highlighted when they are detected in the target region.

The friend detection can be modeled with a set of MCEP operator graph that comprises well-known operators from the visions domain. This operator graph takes as input a set of video streams from cameras, where each frame of the camera is treated as individual event. In a next step a filter operation is performed on each video stream, selecting and forwarding any frame that comprises a face. A succeeding operator collects all filtered video streams and performs a face recognition to identify faces of friends on the frames. To this end, models of the friends faces are kept as the internal state of the latter operator.

Setup and timeline. Our MCEP middleware setup comprises two major software components: (i) a set of *brokers* and (ii) a *controller*. Those software components are executed on resources at the edge of the network, in particular several laptops which are connected over a standard router. Each laptop is equipped with a video camera.

The broker allows to capture faces of users that act as friends and to specify a query for videos of friends over a graphical interface. Moreover, it allows users to choose between several migration approaches. In addition live video streams that contain friends can be viewed.

In order to demonstrate the effects of the selected migration strategies, the friend moves in front of a camera that is connected to a broker and then moves to another camera that is connected to another broker (see Figure 4.7). The consumers can now observe a stuttering video if the operator that performs the friend detection is not placed and migrated optimally.

A visualization component provides in addition a real-time monitoring of key metrics. This visualization is connected to a controller component which runs on one of the laptops. The *controller* is connected to all broker in order to organize them in an overlay topology and manage the placement of operators on those broker.

4.5 Analysis

Locality of Plan Generation. We now compare a centralized approach for the plan creation with MigCEP's decentralized approach for the plan creation. The overhead in terms of messages to generate a plan depends on where the plan is generated. Which is either locally at the current host of each operator ω for a decentralized approach or at a central coordinator for a centralized approach, e.g., at a dedicated node in the cloud. Note that a time graph with vertices for all combinations of placements of operators increases the total number of vertices to the power-set. This is why it is more scaleable, even for a centralized approach, with a global view on all operators, to maintain a single time graph for each operator.

A locally created plan imposes the overhead to exchange plans (and feedback) with neighbors in the coordination step. We assume that this happens in average at a frequency, say $\nu_p(\omega)$, and with an average size, say m_p, for each message. In the central case all

changes in measured and estimated bandwidths have to be sent from the operators to the coordinator in order to allow the coordinator to create a time graph for each operator. We assume that this happens in average at a frequency, say $\nu_m(\omega)$, and with an average size, say m_m, for each message. No network traffic occurs to exchange plans, since all plans are created at the central coordinator. As a result of the plan creation at the central coordinator, stable plans are sent to individual operators with a frequency, say $\nu_s(\omega)$. Other messages, i.e., to reserve resources at targets and collect information of latencies on links between neighbors, are nearly the same for both possibilities. A locally generated plan pays off, iff:

$$\sum_{i=0}^{|\Omega|} \nu_p(\omega_i)\ m_p < \sum_{i=0}^{|\Omega|} \nu_m(\omega_i)\ m_m + \nu_s(\omega_i)\ m_p$$

Threshold for plan generation. In this section we derive a theoretical threshold, that determines when a potential second path in the time graph is shorter than the selected shortest path. This gives a bound *SB* on when it is safe to not generate a new plan. Let $(v_1, ..., v_{TS})$ be the currently selected minimal path and $(v'_1, ..., v'_{TS})$ any other path. Let *DW* be difference in their overall costs of the edge weights w.

$$DW = \sum_{i=0}^{|DE|-1} w((v_{i+1}, v_i)) - \sum_{i=0}^{|DE|-1} w((v'_{i+1}, v'_i))$$

Only if *DW* is positive, a second path can be shorter. Thus if the change in all weights stays in the bounds of the following threshold at the i^{th} edge, *DC* is guaranteed to be negative and the selected path is shorter.

$$SB < \frac{|w((v_{i+1}, v_i)) - w((v'_{i+1}, v'_i))|}{2}$$

Termination. An important property of the coordinated plan generation algorithm is that it eventually terminates, i.e., no new shorter path is selected, iff, the estimation on the available resources at the migration targets does not change, as well as the estimated latencies on links and bandwidths incorporated in a time graph do not change while the plan is generated. We claim that:

Claim 1. Let, all operators have selected a time-graph for the same set of time steps starting from time *now*. Let, path *path* be a newly selected shortest path for the plan generation of operator ω and *tmpPlan* be the previously selected plan for the same set of time steps starting from time *now*. With each newly selected shortest path for the plan generation of an operator the overall costs C_{tot}, the sum over all placement and migration cost, monotonically decreases when plans are not exchanged in parallel between neighbors in the operator graph.

Proof. We show the termination property by a proof of contradiction. Assume that the total costs C_{tot} increase to C'_{tot} after a new shortest path *path* is selected for an operator ω.

The only links that can now increase the placement costs are those directly involved in sharing event streams with neighboring operators of ω. The only migration costs that can be increased are those imposed by ω. Since these costs are used to calculate the edge weights of the time graph, at least the previous shortest path that led to *tmpPlan* would have been shorter, or no plan *tmpPlan* exists that covers the same time steps. □

Deadlock Freedom. An operator starting a coordination will enter a wait state that is left when all neighbors either sent an accept message of the plan or a changed the plan of their own. Each operator needs to leave the wait state eventually in order to ensure deadlock freedom.

To proof freedom from deadlocks we have to consider the wait graph which models at any point in time which operator is waiting for feedback from another operator. A wait graph comprises a vertex for each operator of an operator graph and has a directed edge from one vertex for an operator to another if the first operator waits for feedback from the second. Algorithm 4.1 is message driven if locations of consumers and sources, data rates, and latencies don't change. Thus, changes in the wait graph are triggered by sending and receiving plans and accept messages. Since entering a wait state at operator ω is always followed by sending plans to all neighbors in the operator graph ω', an edge from ω to ω' can be added to the wait graph when sending a plan. Moreover, since each plan is also a feedback any edge from ω' to ω can be removed. Concurrently sent plans are also feedback which cause the removal of an edge from ω' to ω when the id of ω' is smaller than the id of ω. Sending an accept message from ω to ω' also causes to remove such an edge from ω' to ω.

Claim 2. Each linearization of sending and receiving plans and accept messages leads to a cycle free wait graph.

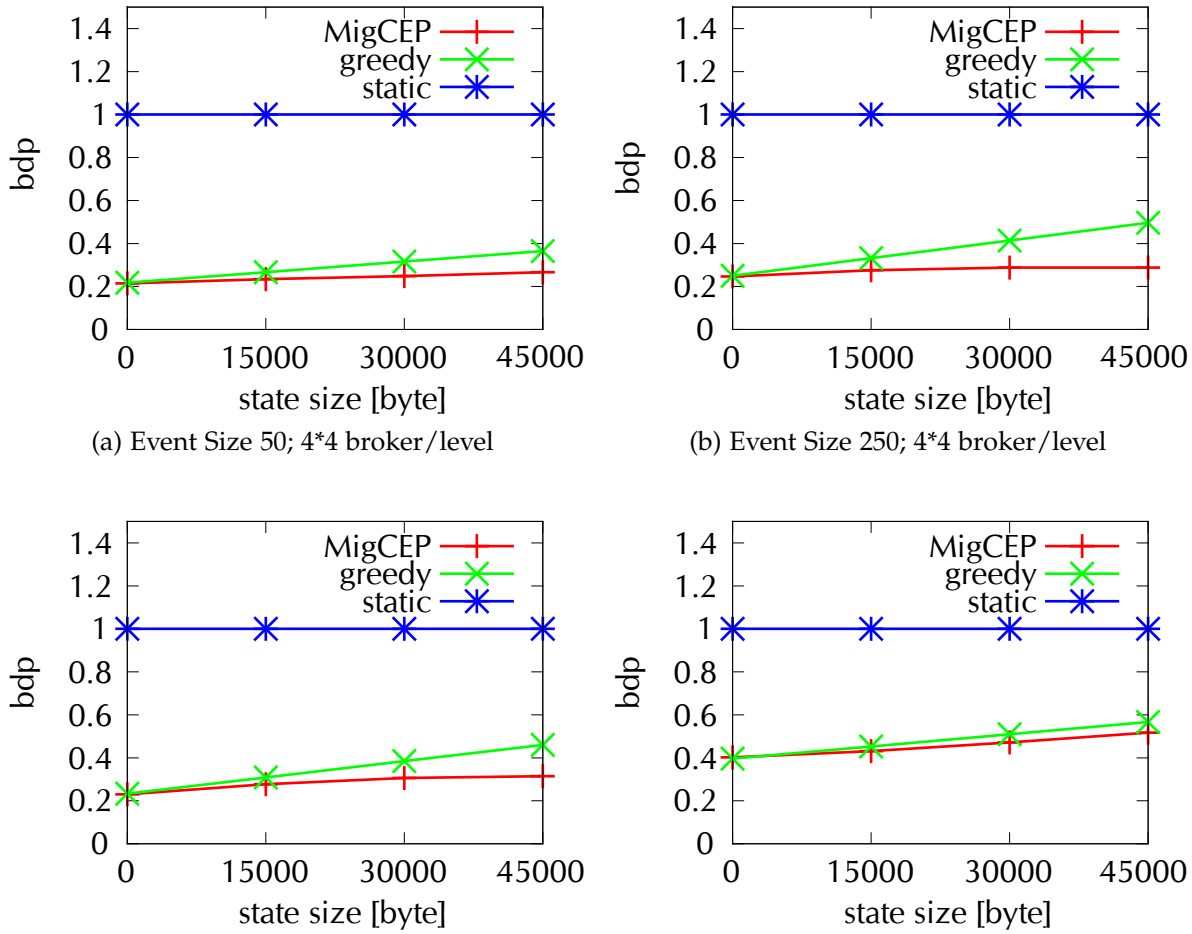
Proof. We proof by contradiction. Assume that a path $\omega^1, \omega^2, \ldots, \omega^1$ in a wait graph exists that is a cycle. Consider the pairs of neighboring operators ω^i, ω^j and ω^k, ω^l in the cycle. Moreover assume that the cycle is closed by ω^j when sending a plan to ω^k. However, since ω^j always sends plans to all neighbors, the edge from ω^i to ω^j would have been removed from the wait graph which breaks the cycle. □

Claim 3. Any edge will eventually be removed from a wait graph.

Proof. Since the wait graph is cycle free, there is always a vertex in the wait graph with only incoming edges. Each vertex in the wait graph that only has incoming edges is not in a wait state or is at least eventually not in a wait state, since only messages sent or received from another operator removed a previously existing corresponding edge. Thus, the operator represented by such a vertex will eventually generate a new plan or send a new accept message and remove any edge to vertices of neighbors in the operator graph. □

The latter claim implies that all operators will eventually be removed from the wait graph, since our coordinated plan generation eventually terminates. Hence, our coordinated plan generation is deadlock free.

4.6 Evaluation



(a) Event Size 50; 4*4 broker/level

(b) Event Size 250; 4*4 broker/level

(c) Event Size 450; 4*4 broker/level

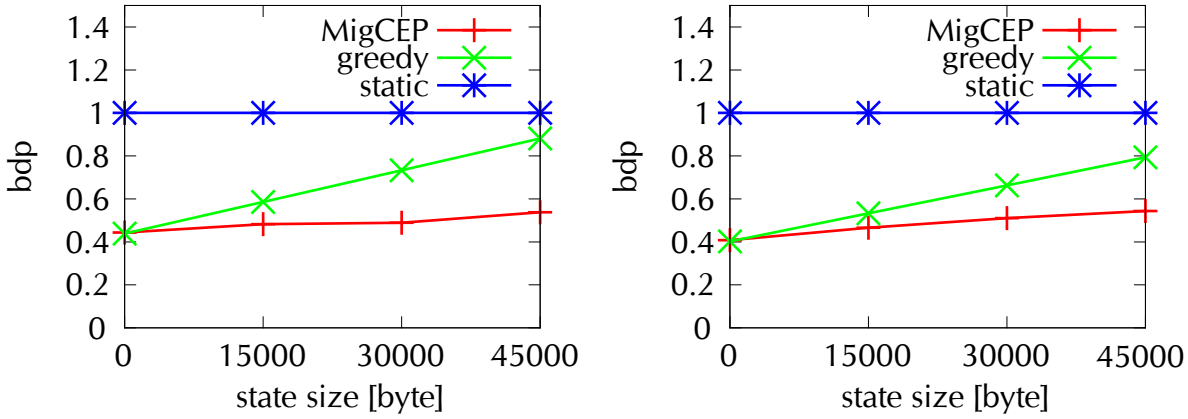
(d) Event Size 50; 10*4 broker/level

Figure 4.8: Evaluation of basic time graph based approach

We implemented the migration approach with the Omnetpp simulator [Var01]. The traffic simulation package SUMO [BBEK11] enabled us to model realistic traffic patterns of vehicles on an OpenStreetMap graph [HW08]. The approach was tested with a generic operator graph resembling Figure 4.1, that allowed us to extract meaningful thresholds.[3]

Similar to typical mobile infrastructures, as discussed in Chapter 2, brokers were organized in a hierarchical data structure. To explore the effects of the infrastructure's density in terms of number of nodes, we generalized the model depicted in Figure 2.2: each tier contained a dynamic number of simulated cloud or fog-nodes. The latency between parents and children in the hierarchy were similar. Vehicles were always connected to a leaf-node of the tree that managed the area where the car was located in. Over the

[3]For simplicity we omitted source u. We parameterized the operator graph regarding the state size, event size, and detection rate.



(a) Event Size 250; 10*4 broker/level

(b) Event Size 450; 10*4 broker/level

Figure 4.9: Evaluation of basic time graph based approach (cont'd)

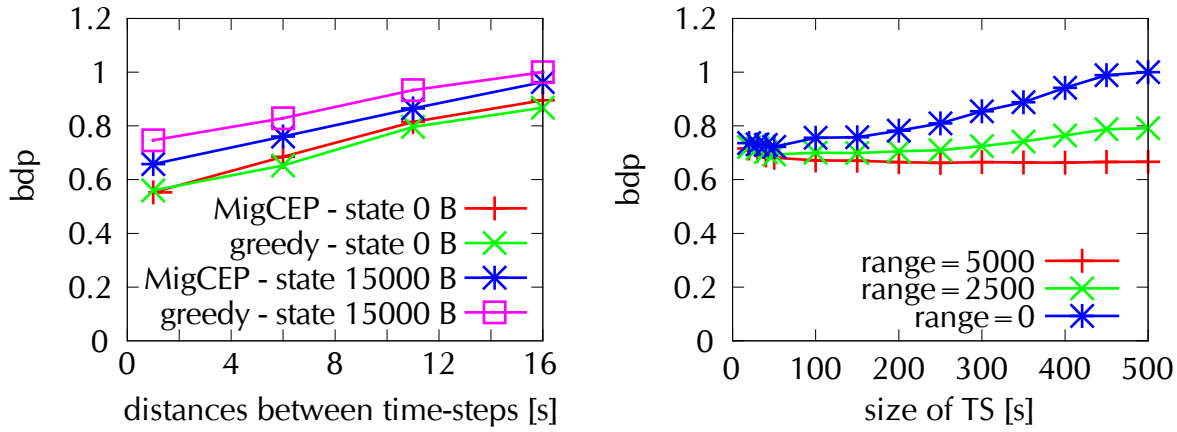
course of 1,000 simulated seconds approximately 1,000 vehicles drove in an area of the size 7.7x3.5 km. Each connected vehicle published approx. one event per second. Each measurement was taken approximately 5 times.

We initially distributed all virtual brokers in the broker hierarchy randomly and tested three possible migration approaches: i) the *static* approach that didn't perform any migration, ii) the *greedy* approach that greedily selected every few seconds the broker with the best placement cost, and iii) our *MigCEP* approach.

Impact of state and streaming size

Since the *MigCEP* approach is designed to consider the state and streaming size for an optimal sequence of migrations, we studied their influence on the main performance criteria, the network utilization. Furthermore, an increase in the number of leaf brokers increases the number of hand-overs in the system, the main source of dynamics. Therefore, we also evaluated the impact of the number of brokers in the hierarchy on the network utilization.

In the experiment, we gradually increased the *event size* of one of the sources in the operator graph from 50 to 250 bytes and the immutable *state size* of the operator that received those events from 0 bytes to 60,000 bytes. The number of brokers varied from 21 fog-nodes in a quad-tree with a single cloud data-center root to a tree with one simulated cloud data-center as root and a flat topology of 40 fog-nodes. In particular, in the first case the infrastructure resembled the one depicted in Figure 2.2 with three levels in the hierarchy, while in the second case the infrastructure comprised only two levels in the hierarchy. The information from the simulated navigation system of the vehicle was used to generate plans for vehicles. On the y-axis in Figure 4.8 and Figure 4.9 the results of the average bandwidth-delay product (bdp) is depicted, relative to the *static* approach. The x-axis depicts the immutable state size in bytes.



(a) Distance between TS

(b) Varying ranges and |TS|

Figure 4.10: Evaluation of time graph parametrization

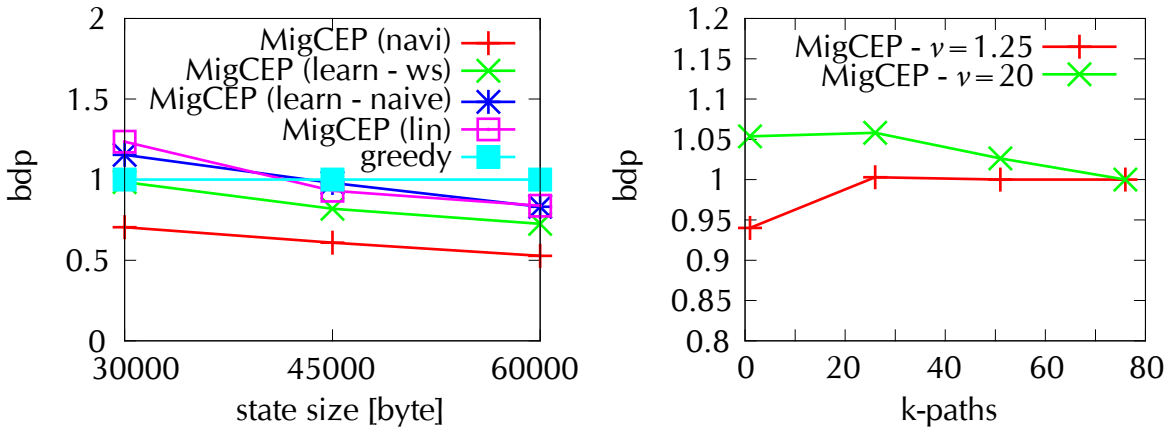
The results demonstrate the strength of our approach in both broker hierarchies. Our approach outperforms by far the static approach, since it adjusts the placements. While the resulting migration and placement costs are comparable to the greedy approach when the size of immutable state is low, the benefit increases the larger the immutable state is, since our approach amortizes the migration costs. The system had to adapt the placement of the operator more often in the case of larger event sizes (Figure 4.8(c) and Figure 4.9(b)) which is why the benefit is more distinguished in those cases. The overhead averaged on low 27 additional coordination messages (resource reservations, plan updates, and feedback) per coordination — independent of event and state size.

Impact of vertices in the time graph

The performance of the plan generation depends on the number of vertices in the time graph. This number changes with the granularity of migration times in a sequence of time-steps *TS* and the number of selected brokers.

The distance between time-steps dictates the system on when it can perform a migration. The longer the distance, the less vertices in the time graph; however, the less chances to find the optimal time for a migration. Figure 4.10a depicts the results, relative to the maximal measured *bdp*, for varying distances between 1 and 16 seconds, effectively reducing the number of vertices by $\frac{1}{4}$ with each larger step. It demonstrates that the increase in the *bdp* is linear, while the number of vertices multiplicativly decrease.

Moreover, the number of time teps of *TS* dictates how far into the future a prediction is made. In the experiment we increased the number of time steps from 100 to 500. We also restricted the number of brokers that are considered for the time graph, by only selecting neighbors of the broker hosting an operator that manages ranges that are no more than 0, 2,500, or 5,000 meters away. Figure 4.10b shows that a larger range gives a better prediction since more brokers were involved. However, the performance only



(a) Spatially Skewed Plans

(b) Changing data-rates

Figure 4.11: Impact of uncertainty

gracefully degrades with fewer brokers. Few time-steps limited the opportunity for planning migrations and too many time-steps reduce the chance to place the operator on the right broker in the future.

Impact of uncertainties

The future placements typically encounter a lot of uncertainties, depending on how the mobility pattern and communication characteristics are captured. The general setup to test the policies dealing with these uncertainties (Section 4.3.3) was to deploy both ω_Ds as state-less operators and ω_F as state-full operator. For the *naive* and *weighted sum* policy (ws), we tested the three methods to capture mobility patterns, uncertain locations from the *dead reckoning* approach (linear), certain locations that could stem from a *navigation* system (navi), and *learned* transitions between leaf broker (learned). To test the *k*-shortest path approach we also randomly increased and decreased the event-sizes. Hence, the placement of one of the ω_Ds had to be constantly adapted for an optimal placement.

The x-axis of Figure 4.11a depicts the varying state size. The average bandwidth-delay product depicted on the y-axis shows the effect of different methods to capture the mobility using different policies on the network utilization; using the greedy approach as baseline. Each policy increased the network utilization less severe than the greedy approach after the state size crossed a specific threshold. That threshold is the point where the reduction in the migration rate makes up for less optimal anticipated placements. Depicted in Figure 4.11b are the results for different numbers of sequences selected form the k-shortest path approach, where the frequency ν in the change in the event-size ranged from every 20 seconds to every 1.25 second. The results are depicted relative to the *bdp* value of the largest *k* for each frequency. For the high-change rate, more paths increased the possibility for wrong placement decisions, however, for slow change rates the system had a better chance to adapt itself.

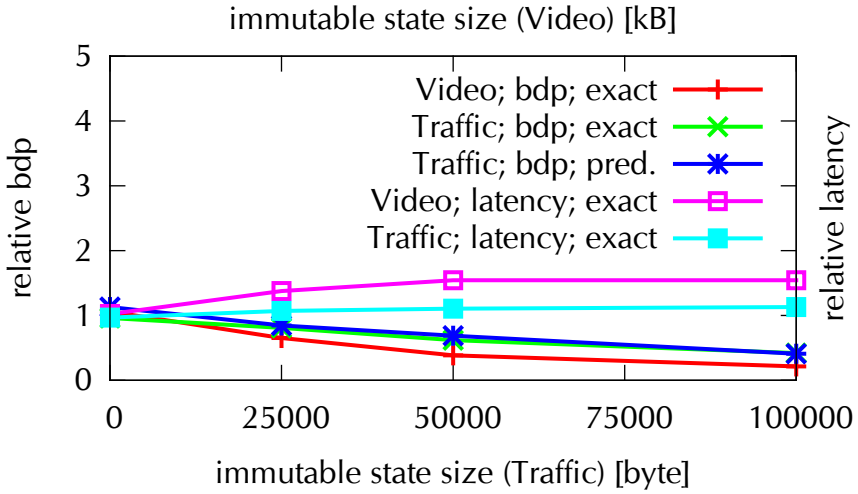


Figure 4.12: Impact Migration on MCEP Query: Immutable State Size

Impact on the performance of MCEP queries

To evaluate the migration approach in context of MCEP queries, we used the video friend finder (*Video*) and accident application (*Traffic*) described in Section 3.5. 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 Note that with the greedy approach in a realistic scenario with 100,000 kB 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 cloud data-center as root and 40 fog-nodes as children. The y-axis in Figure 4.12 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 4.13 show the bdp relative to the greedy

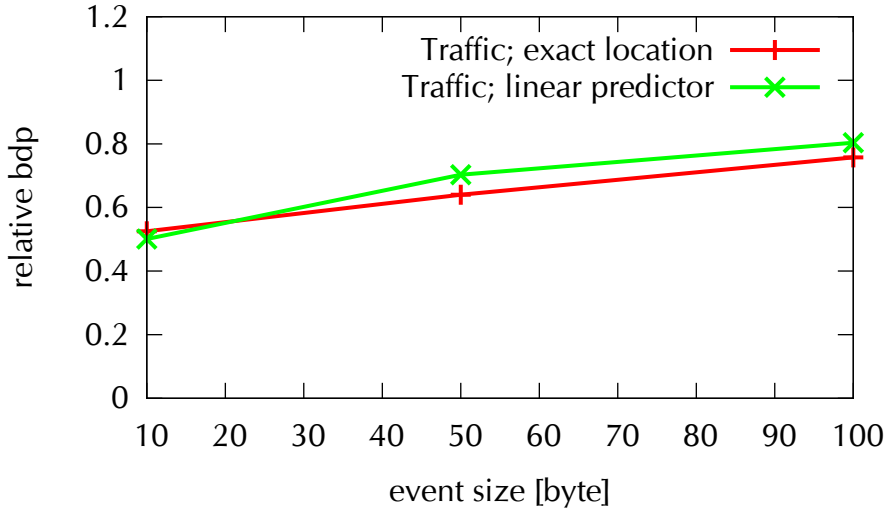


Figure 4.13: Impact Migration on MCEP Query: Event Size

approach over the event size. Since higher streaming costs outweigh the immutable state size, the benefits of our approach decrease with larger event sizes.

Discussion

Our simulation results indicate the huge potential of the plan-based migration and placement mechanisms implemented in the MCEP system to reduce the overall network utilization. In addition, the mechanisms proof to be latency-sensitive, since migrations are not stopped during the migration. In an optimal scenario, where the sources' and consumers' locations are well known and the workload is fixed, our approach drastically improves the overall network utilization compared to approaches that do not consider migrations. More than 80% of the network utilization can be saved compared to a scenario without migrations. Furthermore, more than 20% of the network utilization can be saved compared to a greedy approach which finds an optimal placement every few seconds.

The size and distance between time steps in a time graph have been confirmed as crucial parameters for the efficiency of the approach. While the distance between time steps decreases the number of vertices of the time graph multiplicatively, it increases the network utilization linearly. Moreover, too little time steps increase the overall network utilization, since no shortest path can be found in a time graph that amortizes a migration. Too many time steps include too much uncertainty which also increases the network utilization. We also observed incorrect plannings due to uncertainty, however, the corresponding costs are amortized for large operator states, since constant migrations of the greedy approach outweigh the plan's incorrect placements.

4.7 Related Work

The placement of operators has already been studied in context of DCEP [SHCF03, ZOTW06, PLS+06, YKPS07, YLTX08, CSM11, RDR11, CBD+12, CM13]. However, their main focus are systems where sensors and consumers aren't mobile and the communication characteristics rarely change. Therefore, none of the systems considered the impact of migration costs on the placement, nor do they have to consider the mobility of a consumer, which influences the time when it is required to trigger a migration.

Previous work in DCEP focused on uncertainty in detecting events, e.g., on noisy events captured by sensors [KKR08] or predicted future events [EEF12]. However, none of them studied the impact on the system resources when planing future placements with uncertain location information.

Mobility-aware publish-subscribe systems [HMLJ09, JJE10] dynamically adapt filter operators to new access points of publishers or subscribers. However, filter operators are stateless and therefore none of these systems considers the migration cost itself in the optimization.

Live migration [HDG09, MW12] in cloud computing environments is used to increase data-access locality, energy saving, or load balancing. A typical goal is to reduce the downtime, the time during which the execution is stopped. On the one hand, pre-copying techniques [BKFS07] copy disk-images, i.e., the complete state of the operator, and memory-pages in advance before the processor state. This can arbitrarily increase the bandwidth, because events that are potentially deleted from the queues Q as soon as the Virtual Machine (VM) starts at the target are also migrated. On the other hand, post-copying [HDG09] techniques transfer first the processor state and then the pages. This can increase the latency if the next required page is not migrated yet, even to such an extent, that latency restrictions are not preserved.

Content-delivery networks improve end-user performance, data availability, and reduce server load by migrating data. They have been considered for mobile environments [BTJK04] and cloud environments [WJFR10]. However, they do not have to consider dependencies between different operators.

Distributed location Management [TDSC07] for mobile devices typically stores data where it occurs, as the number of location updates is expected to exceed the number of location look ups. This is equivalent to migrating operators to the nodes where the data occurs. This, however, can lead to a huge number of, possibly unwanted, migrations.

Advances in mobile cloud environments [MW12] allow for transparent migration of tasks, e.g., from a mobile system to the cloud. For example SOD [MW12] only migrates the parts of a Java Code that is actually needed. Which does not suffice for CEP applications, where we want to hold certain end-to-end latencies, because loading code from the source of a migration on demand can take too much time.

4.8 Conclusion

This chapter presented methods to improve the overall network utilization for real-time applications with defined end-to-end latency restrictions in future mobile infrastructures. Using a MCEP system as example, we demonstrated that it pays off to plan migrations ahead of time according to the mobility of users or dependent migrations—more than 80% of the network utilization can be saved. Costly migrations of state, in terms of network utilization, can be amortized by selecting suitable targets in a time-graph data structure that models the expected costs. Furthermore, we presented how application knowledge improves live-migration systems. Execution environments that are typically used in DCEP provide the possibility to find a better serialization of operators, because they allow a live-migration system to infer which state has to be transferred.

5 Multi-Query Optimizations for Mobile CEP Queries

The previous chapters focused on high quality, consistent, and resource-efficient detections of situational information by deploying an individual operator graph for each MCEP query. These operator graphs can be flexibly placed in a mobile infrastructure which utilizes computing resources at the edge and in the core of the network. This way MCEP systems can establish low latency paths between mobile consumers and producers. Note, however, that supporting a large number of consumers in a dynamic environment becomes highly challenging, since a significant amount of resources is needed to execute all individual operator graphs. Consider a route in California [Cal14], where more than 1,500 vehicles/h are driving and a traffic scenario which processes regular speed updates of size 60 B. If the spatio-temporal interest comprises the whole route for one hour, then at least 1 MB of speed events are streamed for each consumer, while for $1,500$ consumer more than 1.4 GB are streamed.

To address this resource problem and make MCEP systems scalable, we propose a reuse component—the Reuse-aware Complex Event Processing (RECEP) framework—which reduces computational and communication load by sharing computations and streams between operators. In particular, the framework exploits two inherent characteristics of MSA applications. The first characteristic is that many consumers have overlapping interests because consumers are typically interested in the same or similar situational information of their surrounding areas. More than 150,000 cars per day are counted on average on the highways in the county of Los Angeles in 2014 [Cal14] (Annual Average Daily Traffic (AADT)). Moreover, according to recent surveys [Cap13], up to 80% of the interviewees were interested in similar situations like "monitor condition of [vehicle] parts and give automated warnings [about roadblocks, ice, traffic jams, accidents]". If there are more consumers in a certain area, then there will also be more overlaps between the consumers, since their locations are similar. Another important characteristic is that for many situational information MSA applications do not require a perfectly accurate set of input data to generate meaningful situational information. This characteristic is well established in CEP systems and used to reduce the system load through load shedding [AN04, TcZ07], sample-based aggregations [BCD03], or sample-based joins [KNV03]. For example, the average speed on a highway based on seven out of ten vehicles may still be meaningful to infer that the traffic is slowing down. Since applications already deal with uncertainty, i.e., the location of the consumer itself is often uncertain [LWG+09], we expect them to be robust to slightly degraded qualities. By

reusing approximate processing results based on slightly mismatching input streams, the system dramatically increases system scalability in terms of consumer queries and input streams.

In this chapter the following contributions are detailed:

- System mechanisms for fine-grained reuse of computations and streams between operators to support scalable processing of consumer queries.
- Methods to control the quality of the approximate results and the associated overhead of the reuse mechanism.
- Evaluation results which show the benefits of the proposed reuse approach in terms of system scalability and resource utilization.

Main parts of this chapter were previously published in [OKR+14b].

The remainder of the chapter is structured as follows: We present the extended CEP model in Section 5.1. The problem is discussed in Section 5.2. Our fine-grained reuse mechanism is detailed in Section 5.3. In Section 5.4, we then evaluate our system before we present the related work in Section 5.5. We conclude in Section 5.6 with a short summary and outlook.

5.1 System Model

Like in previous chapters, we rely on the general operator graph and infrastructure model which is discussed in Chapter 2. Operators in this model perform correlation steps on individual selections s. At times, the start and end of a selection s needs to be referred, where $t_s(s)$ is the smallest and $t_e(s)$ is the largest timestamp of an event comprised in the selection. Moreover, we say a pair of operators ω, ω' is of the same type if they and their induced sub-graphs $G(\omega), G(\omega')$ implement the same logic and receive and produce events of the same type. We further rely on the query model of Chapter 3 which allows to specify an changing spatial interest R defined by a focal object fo.

5.2 Problem Description

In this work, we explore the potential of reusing correlation steps of operators and this way reduce the number of resources that need to be allocated by the CEP system. In mobile environments, consumers are interested in similar types of events, e.g., an accident in a traffic monitoring application. Therefore, we expect that in many cases two consumers will utilize, if not the same, highly similar operator graphs—comprising the same type of

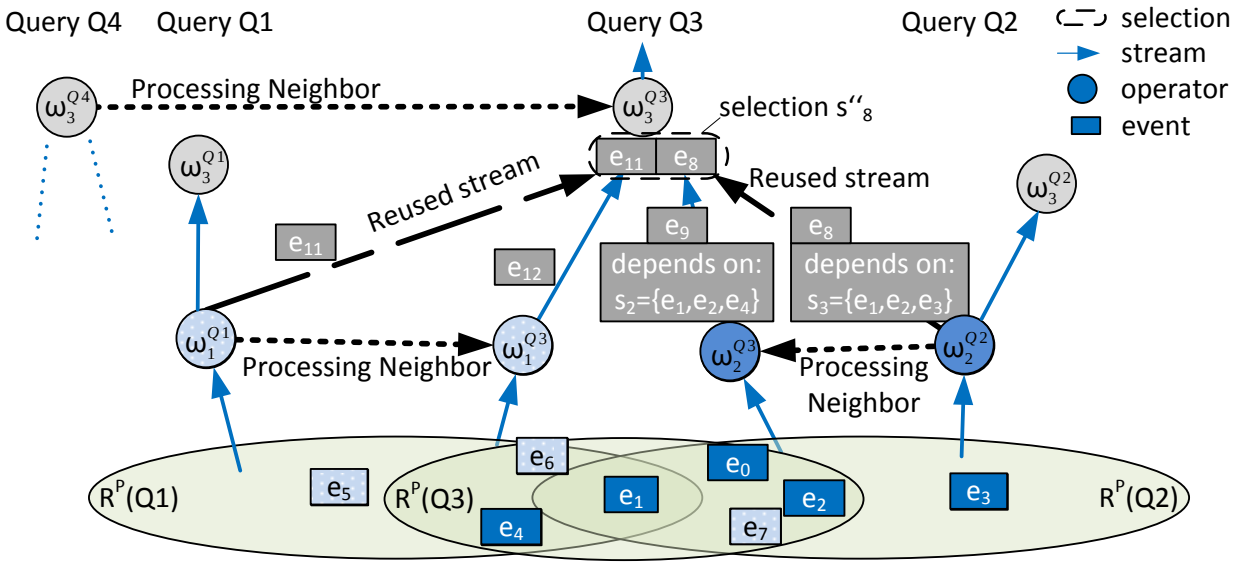


Figure 5.1: Example for reuse-aware operator graph: ω_2^{Q2} processes on behalf of its processing neighbor ω_2^{Q3}. Moreover, ω_1^{Q1} processes on behalf of its processing neighbor ω_1^{Q1}. Hence, event e_8 is delivered to ω_3^{Q3} instead of e_9 and event e_{11} is delivered instead of e_{12}. The operator's selection s_8'' therefore comprises reused events.

operators and the same dependencies to their predecessors and successors.[1] Since the detection of situational information is performed with respect to a focal object centric processing interest, two mobile consumers can only share the result of an operator's correlation step if the processing interest of their queries is overlapping. In particular, the result of an operator can only be reused by another operator if both intend to perform a correlation step with respect to matching selections, i.e., both selections comprise the same set of events. In the example of Figure 5.1 (for its selections see Table 5.1), $\omega_2^{Q_3}$ and $\omega_2^{Q_2}$ both process $s_1 = \{e_0, e_1, e_2\}$ and one of the operators may reuse the result of the other, whereas only $\omega_2^{Q_3}$ processes $s_2 = \{e_1, e_2, e_3\}$. Hence, the execution of the RECEP framework can be characterized by a reuse-aware operator graph (see Figure 5.1) in which some operators perform correlation steps on behalf of other operators. For an operator ω multiple operators—named *processing neighbors*—may exist on whose behalf ω can process and stream produced events. In the example $\omega_1^{Q_1}$ can process on behalf of $\omega_1^{Q_3}$.

The dynamically changing partial overlap in the spatial interests of mobile consumers makes it challenging to find matching selections betwen pairs of operators. The execution of the MCEP system can even be significantly interrupted and hence the time until

[1] Consider that many vehicles are equipped with navigational systems that may display live-information about traffic jams and also of accidents. In both cases, corresponding situational information can be detected using an operator to calculate the average speed in an area in addition to other, disjoint operators.

Operator	Selection s	$f_\omega(s)$	Disparity	Comment
ω_1^{Q1}	$s_4 = \{e_5, e_6\}$	$\{e_{11}\}$	0	prec.$(s_4, s_5) = \frac{\lvert\{e_6\}\rvert}{\lvert\{e_6, e_7\}\rvert} = \frac{1}{2}$
ω_1^{Q3}	$s_5 = \{e_6, e_7\}$	$\{e_{12}\}$	0	
ω_2^{Q3}	$s_1 = \{e_0, e_1, e_2\}$	$\{e_{13}\}$	0	identical to s_1' of ω_2^{Q2}
	$s_3 = \{e_1, e_2, e_4\}$	$\{e_8\}$	0	prec.$(s_2, s_3) = \frac{\lvert\{e_1, e_2\}\rvert}{\lvert\{e_1, e_2, e_4\}\rvert} = \frac{2}{3}$
ω_2^{Q2}	$s_2 = \{e_1, e_2, e_3\}$	$\{e_9\}$	0	
	$s_1' = \{e_0, e_1, e_2\}$	$\{e_{13}\}$	0	identical to s_1 of ω_2^{Q3}
ω_3^{Q3}	$s_8 = \{e_9, e_{12}\}$	$\{e_{14}\}$	0	no reuse from $Q2$ or $Q1$
	$s_8' = \{e_8, e_{12}\}$	$\{e_{15}\}$	0.2	Reused e_8 from ω_2^{Q2}
	$s_8'' = \{e_8, e_{11}\}$	$\{\}$	0.5	Reused e_8 and e_{11}

Table 5.1: Possible selections identified by operators depicted in Figure 5.1. While s_1 and s_1' are identical, s_2 and s_3 are highly similar with a precision of $\frac{2}{3}$. When either reusing events from ω_2^{Q2} and ω_1^{Q1}, from either ω_2^{Q2} or ω_1^{Q1}, or none, the disparity of ω_3^{Q3} result changes due to the disparate processing interests of $Q1$, $Q2$, and $Q3$.

events will be detected is increased. For example, the processing of s_2 has to be blocked until all events comprised in s_2 are locally available to the operator to decide if any other operator may process s_2 on behalf of $\omega_2^{Q_3}$. Here, we approach this problem by providing mechanisms to relax the quality at which the CEP system detects situational information. This offers two major benefits: (i) By utilizing observed event and mobility patterns, the overlap between selections can be estimated before receiving all events which are included in the selection thus reducing the time of interruption caused by finding matching selections. (ii) The gain through reusing can significantly increase, in particular the reuse of an operator can be performed in many cases over a sequence of subsequent correlation steps.

Note that relaxing the quality is for many mobile applications completely acceptable, even if this means to sacrifice strong temporal completeness and spatial consistency guarantees. For instance, an average speed based on seven out of ten speed events can still give sufficient insight to detect that the traffic is slowing down. However, the quality degradation should be kept within acceptable limits. In order to quantify the quality degradation for utilizing a selection s' instead of s within a correlation step, we will compare the *similarity* of the sets of atomic events $A(s)$ and $A(s')$ on which s and s' depend. This allows us to apply a variation of the previously defined metrics *precision* and *recall* which is tailored to selections. The precision defines how noisy the input of a selection is. The recall defines how relevant the input of a selection is.

$$precision(s, s') = \frac{|A(s) \cap A(s')|}{|A(s')|} \tag{5.1}$$

$$recall(s, s') = \frac{|A(s) \cap A(s')|}{|A(s)|} \tag{5.2}$$

Since in a reuse-aware operator graph an operator's selection s can comprise reused events which are detected by operators of several distinct MCEP queries, the quality of a detected event also depends on the disparity of the queries' processing interests. In the example depicted in Figure 5.1, ω_3^{Q3} processes s_8'' comprising reused events with slightly degraded qualities from ω_1^{Q1} and ω_2^{Q2}. Both, ω_2^{Q2} and ω_1^{Q1}, are processing neighbors of ω_3^{Q3}'s predecessors in the operator graph, yet the disparate processing interests of $Q1$ and $Q2$ hardly overlap. For instance e_7 is not selected as input to an operator graph by $R^p(Q1)$ but by $R^p(Q2)$. In a worst case, situational information may not be detected on disparate processing interests. For instance, situational information, like an accident, is detected at ω_3^{Q3} when events in the operator's selection depend on spatially close atomic events of different types, like e_7 and e_2, which is not the case in our example. More formally, let $\mathfrak{R}_u$ denote the set of all processing interests utilized by operators of the reuse-aware operator graph G_r in detecting events of a selection. Then we measure disparity by determining for each atomic event $a \in A(s)$ how many $R \in \mathfrak{R}_u$ match the corresponding location l over all possible combinations of atomic events and processing interests, i.e.,

$$disparity(s) = 1 - \frac{\sum_{a \in A(s)} |\{R | (R \in \mathfrak{R}_u) \wedge (l(a) \in R)\}|}{|A(s)||\mathfrak{R}_u|} \tag{5.3}$$

A disparity of 0 means that all processing interests are comprising all relevant atomic events while in the example of Figure 5.1 the disparity is worse and closer to 0.5 for ω_3^{Q3}, since it reuses results from ω_1^{Q1} and ω_2^{Q2}. Table 5.1 depicts several possibilities of the disparity at ω_3^{Q3}.

The *"selection reuse problem"* addressed in the following sections is therefore to maximize the number of reused selections in the system in order to minimize the required computing and bandwidth consumption of the CEP system, while preserving a high quality. This means that precision and recall should be preserved above an individual threshold q_δ^Q for each query Q in the system while inducing low disparity.

5.3 Selection-based Reuse

We now describe our basic approach to the selection reuse problem by detailing the RECEP framework. The key idea of our solution is to orchestrate the selection management for operators of the same type and find selections which can be reused in the reuse-aware operator graph to save resources. We introduce a component denoted *selection manager*, which distinguishes RECEP from traditional distributed CEP systems. Each operator of the same type receives streams from and forwards streams to the same selection manager (see Figure 5.2). This way, our system is able to determine a selection s that is similar to a set of selections S_c from other operators—we say that s covers all selections in S_c. The result of a correlation step with respect to such a selection s—we call a *covering selection*—can be reused by all operators whose selections are covered.

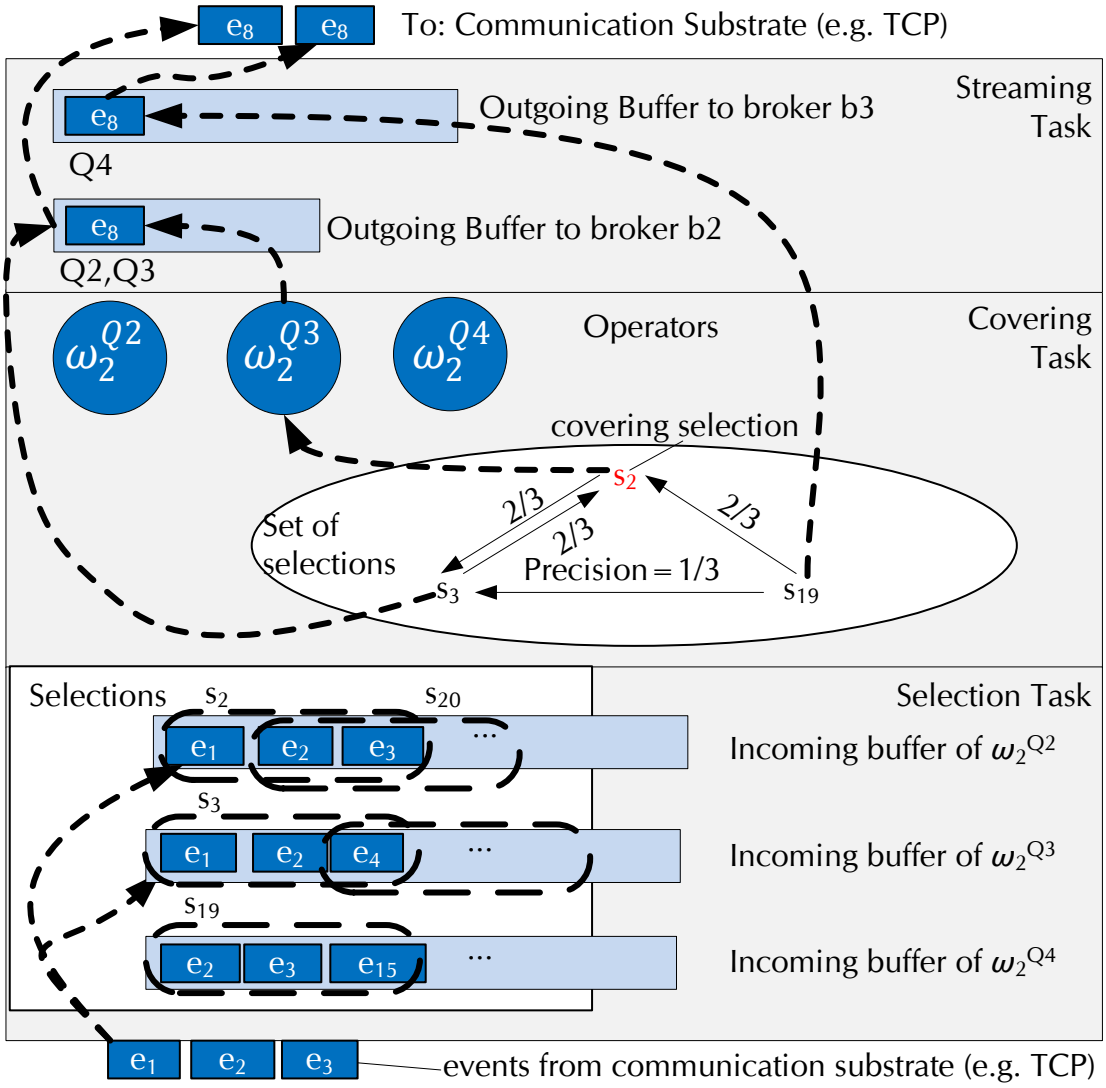


Figure 5.2: Overview over the three main tasks of a reuse manager which is responsible for three operators ω_2^{Q2}, ω_2^{Q3}, and ω_2^{Q4}. In a selection task, the reuse manager identifies selections for these operators on buffered events from their predecessors in the operator graph. This way, the reuse manager's covering task can identify in regular intervals similar selections like s_2, s_3, and s_{19} which can cover each other. From this set of selections, the covering task finds exactly one covering selection, e.g., s_2, which will be processed by an operator. Computational resources for processing s_3 and s_{19} are saved since both reuse the resulting event e_8 of processing s_2. The streaming task ensures that this event is sent exactly once over a network link towards succeeding operators who are managed by the same selection manager; in the example events are sent once to a selection manager hosted on b_2 and once to a selection manager hosted on b_3.

To increase scalablility in face of many operators connected to the same selection manager, our system dynamically assigns groups of operators according to their type and a location attribute to distinct selection manager. The rationale behind this partitioning is that operators of the same type from two distinct operator graphs are only able to share the processing of selections if their processing interests overlap spatially. The same rationale allows our system to limit the search space for covered selections to pairs of operators which depend on atomic events from highly overlapping processing interests. The selection manager therefore maintains for each operator ω a set of processing neighbors, this is, a preselected set of operators connected to the selection manager which can be provided by ω with covering selections.

In Section 5.3.1, we first cover the basics of the selection manager. To increase the scalability of our approach, we illustrate in Section 5.3.2 how to perform a general grouping of operators. Since the mechanisms for managing selections crucially depend on the mechanisms for monitoring the similarity of selections, we will detail the similarity monitoring in Section 5.3.3.

5.3.1 Selection Management

The selection manager performs three tasks (see Figure 5.2) in order to determine and process covering selections: It (i) determines independently processable selections on the incoming streams (*selection task*), (ii) decides which selections are covered by so called covering selections and which operator needs to process covering selections (*covering task*), and (iii) decides to which target the results of a correlation step needs to be streamed to (*streaming task*). For now, we will detail the three tasks of the selection manger to identify a covering selection in a simplified setting where a grouping of the operators is already given and each operator has already been assigned a set of processing neighbors.

Selection Task

The selection task analyzes incoming streams of all connected operators in order to build up the selections on which operators perform their next correlation steps. As part of this analysis, the selection manger identifies according to the selection policy specification when a new selection can be *opened*, e.g., for a sliding window specification a selection can be opened when the first event matching the window boundaries arrived over the incoming event stream. Furthermore, the selection manager identifies with respect to the selection policy when the selection can be closed, e.g., when all events matching a sliding window specification have arrived.

For each opened selection, the selection manager keeps a buffer comprising references to the events that match a selection (see Table 5.2). Moreover, each selection carries two additional attributes: The first attribute marks whether a selection is a covering selection, the second attribute marks whether the processing of a covering selection can be started.

Attribute	Description	Condition	Example
event set	The set of events that are comprised in a selection with their respective provenance information.	Opened, filled, closed and according to selection policy	$\{e_8 \cup \{id(e_1), id(e_2), id(e_4)\}, e_{11} \cup \{id(e_5), id(e_6)\}\}$
set of covered selections	References to selections that are covered by corresponding selection	Initialized with corresponding selection. Selection task adds and removes selections.	$\{s_8\}$ initially, $\{s_8, s_{22}\}$ when covering selection s_{22}, $\{\}$ when covered by other selection
processable	Marks whether processing is enabled or not	Flag is set after covering task has finished	false

Table 5.2: Buffered information for a selection like s_8

Initially, every selection will be marked as a covering selection, i.e., covering itself, and the processing of events is disabled. Since events of a selection can be asynchronously processed, disabling the processing imposes a buffering delay. This buffering delay imposes a trade-off, it allows the selection manager to identify covering selections without imposing processing cost.

Events of the incoming stream are filtered according to the selection policy of opened selections and inserted to the buffer of all matching selections while respecting the order relation imposed by the timestamps. Furthermore, all events in the buffer of a selection need to comprise information about their dependency to atomic events. This information will be later used to find a covering selection that yields a given similarity following the definition of precision and recall. For now, assume that each event is annotated with the complete provenance information [GSEFT13], i.e., the provenance information comprises identifiers (ids) of all atomic events on which the event depends; which can be a lot of information. How to provide such information efficiently is discussed in Section 5.3.3.

Covering Task

The covering task is periodically initiated and performed with respect to all selections for which processing is disabled. After a covering task is performed, covering selections are determined, scheduled for processing, and their processing attribute is enabled. The goal of the covering task is therefore to minimize the number of covering selections. For now, we will disregard the fact that some selections may be incomplete and not all relevant events are available yet.

The problem to find a set of covering selections that is minimal can be reduced to the minimum set covering problem [CSRL01, Chapter 35.3], which is known to be NP hard.

In this work, we therefore build on heuristics to find the minimum number of covering selections. In a nutshell, the set covering problem is the problem of finding for a set of elements denoted $\mathbb{X}$ and a family of subsets of $\mathbb{X}$ denoted F, the minimum number of subsets in F covering all elements in $\mathbb{X}$. In our setting, an element of $\mathbb{X}$ is a selection. Moreover, a subset in F is given by a set of selections which are covered by the same selection with acceptable level of similarity.

We will propose and later evaluate two ways to approach the problem of finding the minimal number of covering selections. In the first approach, we generate the family of feasible subsets and then solve the set covering problem by applying Johnson's greedy heuristic [Joh73], which is known to give a logarithmic ratio bound with respect to the maximum size of a subset (for details we refer to Algorithm 3.11). As alternative, we propose a heuristic to reduce the computational cost which stems from the generation of subsets. The heuristic prioritizes covering selections which are determined by operators that depend on processing interests with high overlaps with processing interests of other operators in the same group, since those selections have a higher chance to cover many selections.

Our previous work in Section 3.4.2 used a set cover heuristic to select a minimal number of processing interests for a single query, in contrast to this work, where we select a minimal number of reusable selections for multiple queries. Note, that the mapping to the set covering problem and therefore the solution of the problem discussed in Section 3.4.2 differs from this one, e.g., in the constraints.

Maximum Similarity Coverage Heuristic (MSH). The MSH algorithm, which is depicted in Algorithm 5.1, takes as input a set of operators Ω. All operators in Ω are of the same type and are grouped by the selection manager. MSH returns as a result W, comprising the set of covering selections as well as the selections covered by each covering selection. To this end, MSH generates in a first step the family of subsets F that jointly cover all selections (Line 4-Line 14). This is achieved by iterating over all selections $s \in S_\Omega$ for which processing is still disabled (Line 4). Let s_ω be a selection comprised in the buffer maintained for operator $\omega \in \Omega$, then MSH builds w.r.t. s_ω a subset as follows: MSH will add for every processing neighbor ω' of ω (Line 6), the selections $s_{\omega'} \in S_\Omega$ to the subset if $precision(s_{\omega'}, s_\omega)$ and $recall(s_{\omega'}, s_\omega)$ meet an acceptable similarity threshold (Line 9). The subset is then added to F (Line 13).

After MSH iterated over all selections, the greedy heuristic is applied to select the subsets in F that yield the largest coverage. Once a subset is chosen, its elements are removed from the remaining subsets in which they are comprised, in particular also the subset for which they are chosen as the covering selection is removed (Line 20).

Maximum Neighbor Heuristic (MNH). The maximum neighbor heuristic presented in Algorithm 5.2 also takes a set of operators Ω as input and returns the set of covering selections W. Each operator ω in Ω is assigned a priority level which is given by the overlap of the dependent processing interest with ω's processing neighbors. MNH then

Algorithm 5.1 Maximum Similarity Coverage Heuristic

```
1: function find_covering_selection(Set of Operators ω)
2:   W ← ∅ //Set of covering selections
3:   F ← ∅ //Family of subsets
4:   for all s_ω ∈ S_Ω do //iterate over selections of operators in Ω
5:     ℂ ← {s_ω} //each selection covers itself
6:     for all ω' ∈ neighbor(ω) do
7:       for all s_ω' ∈ S_ω' do
8:         if precision(s_ω', s_ω) > precision(q_δ^ω) ∧ recall(s_ω', s_ω) > recall(q_δ^ω) then
9:           ℂ ← ℂ ∪ {s_ω'} //increase subset for s_ω
10:        end if
11:      end for
12:    end for
13:    F ← F ∪ (ℂ, s_ω)
14:  end for
15:  L ← S_Ω
16:  while L ≠ ∅ do //greedy set covering heuristic
17:    choose (ℂ, s) ∈ F s.t. ℂ ∩ L is maximal
18:    W ← W ∪ (ℂ, s)
19:    L ← L \ ℂ
20:    ∀(ℂ', s') ∈ F : update ℂ' ← ℂ' \ ℂ
21:  end while
22:  return W
23: end
```

iterates over the operators in order of their priority by extracting in each iteration the operator with maximal priority (Line 5). Let ω be the operator with the maximal priority extracted from the priority list PL, then MNH checks for each selection of ω, say s_ω, whether the set W already comprises a selection $s_{\omega'}$ which can cover s_ω by determining their similarities (Line 9). If this is the case, then s_ω is added to exactly one subset in W (Line 10). If no covering selection exists in W, then s_ω is added to W as a new covering selection with s_ω itself as covered selection. The algorithm terminates after it has iterated over all operators.

Properties. To understand the different behavior of MSH and MNH and their influencing parameters, let us briefly compare the complexity of both approaches in finding the covering selections. Let n_Ω be the average number of processing neighbors, n_s the average number of selections the selection manger maintains per operator, and n_q the average number of comparisons to determine precision as well as recall of a pair of selections. The complexity of MQC is caused by (i) generating the subsets (Line 4) and (ii) applying Johnson's heuristic (Line 16). Generating all subsets has an expected cost of $\mathcal{O}(|S_\Omega| n_\Omega n_s n_q)$ since the similarity of a selection is determined against $n_\Omega n_s$ other selections. Furthermore, Johnson's greedy heuristic will impose a cost of $\mathcal{O}(|W||S_\Omega| +$

Algorithm 5.2 Maximum Neighbor Heuristic

```
1: function find_covering_selection(Set of Operators Ω))
2:   PL ← priority_list(Ω) //Operators are sorted by priorities
3:   W ← ∅ //Set of covering selections
4:   while PL ≠ ∅ do //cover all selections of all operators
5:     ω ← extract_max(PL) //Extract oper. with max prio.
6:     for all s_ω ∈ S_ω do //iterate over selections of ω
7:       covered ← false
8:       for all (ℂ, sω') ∈ W with ω ∈ neighbor(ω') do
9:         if precision(s_ω, s_ω') > precision(q_δ^ω) ∧ recall(s_ω, s_ω') > recall(q_δ^ω) then
10:          ℂ ← ℂ ∪ {s_ω} //increase subset for s_ω'
11:          covered ← true
12:          break
13:        end if
14:      end for
15:      if covered = false then
16:        W ← W ∪ ({s_ω}, s_ω) //covering selection s_ω
17:      end if
18:    end for
19:  end while
20:  return W
21: end
```

$|S_\Omega| n_s n_\Omega$). In each round—from beginning until termination—one element is inserted to W. Extracting the maximal element costs $\mathcal{O}(\log |S_\Omega|)$ while updating the priorities of $|F| = |S_\Omega|$ elements can be achieved amortized in $\mathcal{O}(|S_\Omega|)$ using a Fibonacci Heap. Finally, overall at most $|S_\Omega| n_s n_\Omega$ elements are to be removed from F if all selections of processing neighbors are covered by all selections. Hence, the expected time to calculate the set of covering selections is $\mathcal{O}(|S_\Omega| n_\Omega n_s n_q + |W||S_\Omega|)$. In contrast, MNH iterates over all selections $|S_\Omega|$ and makes in each iteration at most $|W| n_q$ comparisons. The cost for calculating the priorities of the operators depends on the frequency of changes in the processing interests, however, for now we disregard this cost and consider the frequency of covering tasks to be dominating. Hence, the time complexity for MNH is $\mathcal{O}(|W||S_\Omega| n_q)$.

The performance for MNH is therefore expected to yield performance gains when the generation of subsets is the dominating factor for the MSH. We can influence this factor at run time when grouping the operators maintained by Ω. If the grouping of operators ensures all operators of Ω perform processing with respect to the same region, we expect the number of selections covering the grouping, namely $|W|$, to be by far smaller than $n_\Omega n_s$. The performance gains of MNH comes also at a potential cost in the similarity and number of found subsets, since MNH restricts the number of comparisons to be performed with other selections.

Independent of the decision whether to choose MNH or MSH, the complexity analysis

shows that the computational overhead of the selection manager requires mechanisms to increase scalability with appropriate mechanisms. In this context we have to answer:

- For how many pairs of operators should the selection manager compare selections to generate the subsets?
- How many pairs of selections should be compared to generate the subsets? In particular, can we make a coarser decision on the coverage by comparing many selections of a pair of operators at once?
- How much information about atomic events should be examined to make a good estimation for precision and recall when building the subsets?

We will give answers to these questions when introducing the run-time aspects of the RECEP system in the next sections.

Streaming Task

When an operator ω produces events as a result of processing a covering selection s_ω, the streaming task is initiated. While the covering task is mainly concerned with reducing correlation steps, the streaming task is mainly concerned with saving bandwidth by reducing the number of events sent over a network link. Note that the produced events of ω need not only to be forwarded to a destination defined by ω, but also to the destinations of operators whose selections were covered by s_ω. The destinations of these events are those selection managers to which the successors of operators with covered selections are connected to. RECEP ensures that events are sent only once over a network link if two distinct destinations reside on the same broker, the succeeding selection managers of operators associated to these selections are placed on the same broker. To assign these events at the destinations to operators, they are annotated with all ids of operators to which they need to be forwarded to.

Upon arrival, each incoming event is analyzed and for each annotated destination operator the selection task is initiated. Note that because preceding operators can reuse different asynchronously processed selections, events might not arrive in the desired order. Each selection is therefore assigned a sequence number and we inform successors about sequence numbers of selections for which the processing is finished. Such information can be piggy-backed with the produced events or regular heart-beat messages, which allows the successors to first buffer and sort events according to the sequence number before they are added to the incoming buffer.

5.3.2 Selection Grouping

This section details how a selection manager groups operators in order to connect them to the same selection manager and based on that decision assigns processing neighbors.

Distributed Execution of a Selection Manager.

Operators of the same type which process on behalf of consumers with overlapping interests have to be connected to the same selection manager to find covering selections. The more operators can be connected to the same selection manager, the more selections may be covered and a gain in saving resources can be achieved. Moreover, note that the selection manager and its connected operators do not need to be hosted by the same broker. Therefore, it is easy to integrate RECEP with placement algorithms to optimize latency and bandwidth usage (see Chapter 4). However, the number of operators connected to a selection manager influences the throughput of the selection manager in processing selections. RECEP avoids overload situations for the selection manager by introducing a grouping mechanism. This way RECEP can share the load in processing selections between multiple selection managers. A natural way to establish a grouping is to classify the operators by their type as well as their spatial location, i.e., two operators are only connected to the same selection manager if they are of the same type and the corresponding location of the focal object is comprised in the same spatial area. This way, operators that belong to queries with a high overlap in the spatial interest are automatically grouped.

Dynamic Group Management. For the RECEP system, the geographical region in which events are produced and consumed is partitioned into disjoint spatial regions. Each of the spatial regions will be assigned a selection manager by the MCEP controller if at least the focal object of one query is comprised within this area. A new selection manager will be deployed once the first operator of a given type is comprised within the spatial region and is discarded after the last operator of a specific type has left the spatial region. To this end, the MCEP controller is a distributed component that scalably keeps track of the location of mobile consumers similar to location services [LR01b, ZZL07]. Moreover, we bound the number of operators that can be connected to a selection manager. If this bound is exceeded, the number of selection managers deployed in a spatial region will be increased, and each selection manager takes an even share in processing the selections.

The requirements regarding the size, location, and shape of such predefined spatial areas can vary for different applications, and therefore can be specified for RECEP by the system administrator. For example, for rectangular shaped interests Quad-Trees [MS05] can be used to efficiently update the deployment of the selection managers and find selection managers to which an operator can connect to. This connection is dynamically adapted with location updates provided by the consumers. All selection managers are initially hosted by the same broker. Their placement can then be dynamically adapted with the previously mentioned operator placement mechanism by treating selection managers in the same way as operators.

Dynamic Processing Neighbor Selection

After a new group is established and operators are connected to selection managers, RECEP needs to track and update the processing neighbors for each operator of a group. Recall, that the set of processing neighbors, assigned to each operator, is a key parameter for MNH and MSH in efficiently finding highly similar covering selections. Further observe, even though operators are grouped according to type and spatial region, the number of viable processing neighbors can differ for each operator. In the example depicted in Figure 5.1, most of the selected atomic events lie in the overlap of the processing interests $R^p(Q3)$ and $R^p(Q2)$. Therefore, it is likely that similar selections are identified if operators of $Q3$ and $Q2$ are processing neighbors. However, the similarity of selections identified by operators of $Q3$ and $Q1$ is likely to be low as $R^p(Q3)$ and $R^p(Q2)$ hardly overlap. In fact, there is no similar selection and the overhead of executing MNH and MSH is avoided if operators of $Q3$ and $Q2$ are not processing neighbors. Moreover, since consumers can choose distinct similarity thresholds, the processing neighbor relationship is in general not even symmetric.

Besides efficiently updating the set of processing neighbors, we address how to achieve a low disparity which results from reusing selections over several levels in the operator graph. For example, in Figure 5.1, ω_2^{Q2} and ω_1^{Q1} receive event streams from processing interests with low spatial overlap which imposes a high disparity for s_8'' at ω_3^{Q3} which reuses their results. Observe, however, from Table 5.1 that disparity can be reduced by trading off the opportunity to reuse the result of a preceiding operators against disparity, i.e., the disparity of s_8' is lower than that of s_8'' since the result of ω_1^{Q1} is not reused.

To this end, our key idea is to utilize the overlap between processing interests (i) to find processing neighbors for operators that are expected to process many similar selections and (ii) to ensure that processing interests that correspond to operators with the same processing neighbor are highly overlapping to reduce disparity. There are two approach directions to find queries with a high spatial overlap in the processing interest, either a *query clustering* or a *moving range queries* approach. A typical clustering algorithm like k-means [Mac67] or DBSCAN [EKSX96] allows us to efficiently assign queries according to their similarity to distinct groups, e.g., according to the pair-wise distance between the locations of focal objects .[2] Operators of members of the same cluster can then be used as processing neighbors. However, when the maximum distance between focal objects in the group is too small, spatially overlapping members of different clusters with the prospect of highly similar selections are not considered as processing neighbors. If the maximum distance is too large, then processing interests may not overlap which results in a high disparity. The moving range query approach allows us to find individual processing neighbors for each MCEP query and is therefore the approach chosen by MCEP. In particular, a moving range query over indexed locations of focal objects reveals corresponding MCEP queries that are spatially close to each other. This way, all operators

[2]Note, for brevity, we leave out a discussion of the relation between similarity and the spatial distance, since previous sections, e.g., Section 3.3.2, discussed a similar relation for QoR.

of MCEP queries with the prospect of highly similar selections are found and can be used as processing neighbors, at the additional cost of maintaining range queries over a spatial index.

Coordinated Neighbor Selection. In the remainder, we detail our solution to maintain an operator's processing neighbors in a way that allows MNH and MSH to find highly similar selections without inducing a low disparity: the *coordinated neighbor selection* (see Algorithm 5.3). We utilize the overlap of processing interests to consistently restrict the number of processing neighbors for each operator over all levels of an operator graph G. To this end, we annotate each operator of a query with the current location of the focal object. Moreover, each selection manager is responsible to ensure for each connected operator $\omega \in \Omega$ (e.g., ω_3^{Q3}, ω_3^{Q1}, and ω_3^{Q2} in our example in Figure 5.1) the following:

1. That the distance between the respective focal objects of the query that ω is a part of and those queries whose operators have ω as processing neighbor remains below a threshold ($dist_n(\omega)$). This way, we achieve a high spatial overlap between the corresponding processing interests, which promises highly similar selections. In the example, if $dist_n(\omega_3^{Q3})$ equals $R(\omega_3^{Q3})$'s diameter, ω_3^{Q1} and ω_3^{Q2} can be processing neighbors of ω_3^{Q3}.

2. That focal objects of queries whose operators have ω as processing neighbor are themselves confined in a spatial region such that their pair-wise maximum distance remains below a second threshold ($dist_o(\omega)$). A consistent spatial region over all levels of the operator graph allows us to achieve a high pair-wise spatial overlap between the processing neighbors which promises a low disparity. In the example, if $dist_o(\omega_3^{Q3})$ is smaller than the distance of $Q1$'s and $Q2$'s focal object either all operators of $Q1$ or all operators of $Q2$ can be selected as processing neighbors of operators from $Q3$.

3. That the number of operators that select the same operator ω as processing neighbor remains below a threshold ($k(\omega)$) to give a bound to the number of comparisons of selections by the MNH and MSH. In the example, if $k(\omega_3^{Q3})$ is 1 either ω_3^{Q1} or ω_3^{Q2} can be selected as processing neighbors of ω_3^{Q3}.

To account for individual movement patterns of focal objects and similarity requirements, we need to perform the coordinated processing neighbor selection for each individual operator graph of a MCEP query in regular time intervals. Moreover, the coordinated processing neighbor selection consists of two steps. In the first step (see Lines 4-8), the system determines at the level of a query Q's root operator ω_r a so-called *reference operator graph* H, e.g., $G(Q1)$, $G(Q2)$, or $G(Q3)$ at ω_3^{Q3} in the running example. The location of the focal object corresponding to the reference operator graph centers all operators that might select one of $G(Q3)$'s operators as processing neighbor. This minimizes the chances for disparity in the second step (Lines 9-16), by assigning operators

Algorithm 5.3 Coordinated Neighbor Selection

1: *Requires:* Ω //Operators managed by selection manager, index: focal object's location
2: $dist_n(\omega), dist_o(\omega), k(\omega)$ //allowed distance between neighbors and number of neighbors for each $\omega \in \Omega$
3: *Defines:* $\forall \omega \in \Omega$: a set of processing neighbors(neighbor(ω)), a reference graph ($H[\omega]$)
4: **upon** update timeout(`Operator graph` G *with root* ω_r)
5: $N(G) \leftarrow$ range_query($fo(G), dist_n(\omega_r), \Omega$)
6: $H(G) \leftarrow g \in N(G)$ with max $\sum_{g' \in N(G)} |R^p(g) \cap R^p(g')|$)
7: $\forall \omega \in G$: **trigger** neighbor_selection($\omega, H(G)$)
8: **end**
9: **upon** neighbor_selection(`Operator Graph` G, `Operator Graph` $H(G)$) at ω
10: $H(\omega) \leftarrow H(G)$
11: $\forall \omega' \in \Omega$: neighbor(ω') $\leftarrow$ neighbor(ω') $\setminus \omega$
12: $\Omega_a \leftarrow$ k_range_nearest_neighbor($k(\omega), dist_o(\omega), fo(H(G)), dist_n(\omega), fo(G), \Omega$)
13: **for all** $\{\omega' | \omega' \in \Omega_a \wedge dist_o(\omega) \geq dist_o(\omega') \wedge (|H(G) - H(\omega')| < dist_o(\omega))\}$ **do**
14: neighbor(ω') $\leftarrow$ neighbor(ω')$\cup\{\omega\}$
15: **end for**
16: **end**

of $G(Q3)$ as processing neighbors to operators whose focal objects are spatially close to the reference operator graph's focal object.

Each operator graph with a focal object closer than $dist_n(\omega)$ to an operator graph with a root ω_r is a potential candidate for a reference operator graph. To find these candidates, each root $\omega_r \in \Omega$ managed by a selection manager issues a range query in regular time intervals. Note, an extension of the algorithm to use continuous range queries is straight forward and left out for brevity. The range query returns new operators ($N(G)$), e.g., ω_3^{Q1} and ω_3^{Q2} for ω_3^{Q3}, whose operator graphs' focal objects are close to G's focal object (Line 5). To increase the reuse at distinct operators, i.e., both ω_3^{Q1} and ω_3^{Q2} reuse from ω_3^{Q3}, we enforce that all operators managed by the selection manager choose a reference operator graph with similar focal object. Particularly, we assign each root operator a priority with respect to the spatial overlap over all processing interests corresponding to operators in Ω, e.g., in the example the priority of ω_3^{Q3} is 2.1, if the overlap to the processing interests of ω_3^{Q2} and ω_3^{Q1} is 0.53 and 0.57. Since the priority of ω_3^{Q1} is higher than the priority of ω_3^{Q2} and ω_3^{Q1}, which hardly overlap with each other, ω_3^{Q1} is selected as reference graph (Line 6). To be able to assign processing neighbors, $H(\omega)$ is then forwarded to all selection managers which are connected to an operator $\omega \in G$, i.e., to all predecessors of ω_3^{Q1} (Line 7).

Once the selection manager knows $H(G)$ for an operator ω, it determines the set of operators to whom ω is a processing neighbor by performing a range query centered in the location of $H(G)$'s focal object. The query returns a maximum of $k(\omega)$ operators whose focal objects are comprised within a radius of size $dist_o(\omega)$, respectively $dist_n(\omega)$, around the focal objects of $H(G)$ and $G(\omega)$ (Line 12). For example, if ω_2^{Q3} is restricted

to one neighbor, it would select ω_2^{Q2} as its processing neighbor, since its focal object is closer than ω_2^{Q1}'s focal object. From the result we select those operators ω' that (i) have a reference operator graph with at most a distance of $dist_o(\omega)$ to $H(\omega')$, and (ii) have at most the same restrictions as ω' ($dist_o(\omega')$) to reduce indirect disparity (Line 13). Indirect disparity is induced by reusing results with a degraded similarity at processing neighbors.

Update Frequency and Priorities. In order to reflect the mobility driven changes in processing neighbor relationships when there are no changes in the grouping of operators, the selection of the reference operator graph is performed in regular intervals. A low interval will comprise often the same set of processing neighbors for slow mobile consumers and a high interval will often lead to processing neighbors with non-overlapping interests for faster mobile consumers. Moreover, finding processing neighbors itself imposes a processing overhead which is amortized in scenarios with a high event rate and thus a high potential for resource savings.

The priority list for the MNH can also be updated at the granularity of the temporal interval. The costs for doing so at a frequency ν_u are bounded by $\mathcal{O}(\nu_u * \log(|\Omega|)$, where $\log(|\Omega|)$ is the amortized cost for updating the priority. Since overlaps are calculated anyway to find potential reference operator graphs, we ignore those costs for the analysis. However, it becomes clear that if the the update rate of operator graphs and number of operators is low compared to the number and calls of the covering tasks, those costs for updating the priority list become neglectable.

5.3.3 Selection Batching and Monitoring

We now detail two optimizations. First, we describe how grouping several selections together into so-called *selection batches* can increase scalability. Second, we describe how we can efficiently calculate and predict the similarity.

Batching Selections

Reusing individual selections can be costly. Consider computational weak operations like parameter filters that have a small selection, e.g., one event per selection, and can be performed with few instructions. Finding k other selections that might cover the selection requires at least k instructions to decide that these selections are potential covering selections. The solution is thus to group selections into sets of subsequent selections, denoted as *selection batches*. A selection batch can then be processed on behalf of another selection batch by a processing neighbor.

Processing of Batches of Selections. Selections can be batched on a temporal basis or for a number of n_β subsequent selections. In the first case a selection batch β is assigned a time window $(t_s(\beta), t_e(\beta))$ and comprises all selections s where $t_s(s) \geq t_s(\beta)$ and $t_e(s) \leq t_e(\beta)$. For the latter, beginning and end of these number-based selection batches

can be referred to by the time-stamps of the first (s_1) and last selection (s_n) in the batch, i.e., $t_s(\beta) = t_s(s_1)$ and $t_e(\beta) = t_s(s_n)$. Furthermore, there are two special selection batches: $n_\beta = 1$ denotes the individual reuse of selections as discussed until now and $n_\beta = \infty$ denotes the case where all selections of an operator are grouped to exactly one batch, i.e., an operator does reuse all results of a processing neighbor or none at all. Partitioned execution environments by sensor id or location and locality preserving operators (see Section 3.3.3) allow our system to partition batches by sensor id or location. The choice of the parameters are operator specific and depend also on the expected mobility pattern of the application and therefore need to be specified by the domain expert.

The MNH and MSH algorithms for batches require little changes to their previously presented versions. Instead of comparing and iterating over individual selections, batches of selections are compared. The core difference to reusing individual selections is that we lose a fine-grained control on the reuse. The system either cannot cover another batch if at least one selection of the batch does not yield an acceptable similarity, or the consumer accepts that at least some selections in the batch have a low similarity and only the average similarity over all selections yield a higher similarity than q_δ. It depends on the application how important it is to ensure similarity, yet, only the latter approach can be efficiently implemented.

Scalable Run-Time Similarity Monitoring

In this section, we discuss how we reduce the overhead for calculating the precision and recall if all events of a selection or batch are present at the operators incoming buffer. We also discuss how to predict the similarity if not all events of a selection or batch are present at an operators incoming buffer.

Determining the similarity metrics. Both heuristics of Section 5.3 calculate the precision and recall for a pair of selections or batches based on estimates about atomic events they depend on. For example, by annotating the ids of atomic events to processed events similar to [GSEFT13]. The better these estimates are, the better the calculated similarity. However, since calculating the similarity according to the metrics given in Section 5.3 requires a pairwise comparison of these estimates, the imposed overhead can be high. We detail three approaches to determine the atomic events and their influence on the overhead and the similarity.

Spatial Interest Similarity: The similarity can be determined by intersecting the processing interests of operator graphs that correspond to a pair of selections or batches. For instance, two congruent processing interests result in a precision of 1, while disjoint processing interests result in a precision of 0. To this end, operators keep track of the current processing interest of their corresponding operator graph. This approximation of atomic events assumes that all atomic events are evenly distributed in the spatial interests of queries, which can result in a high inaccuracy, since none of the events of a selection might actually lie in the spatial overlap. Moreover, it disregards the fact that events of a selection might already depend on reused selections.

Per Neighbor Similarity: The number of events that are sent to both an operator ω and its processing neighbor, i.e., their predecessors reused the processing of the same batch, can be used as indicator for the overlap in atomic events. In our running example depicted in Figure 5.1, e_{13} and e_8 are received by ω_3^{Q3} and ω_3^{Q2}. In a rough estimation, this means that at least 2 atomic events stem from the spatial overlap of their corresponding processing interest. These events can easily be identified, since they are streamed only once for a pair of processing neighbors to the selection manager during the streaming task. The system determines, for each pair of selections, say s and s', the number m_o of events that are produced by a common predecessor and the number of events in those selections m_s and $m_{s'}$. Precision and recall thus evaluates to $\frac{m_o}{m_s}$ and $\frac{m_o}{m_{s'}}$. Observe, this method disregards events that stem from already reused selections and therefore already have a degraded similarity, i.e., e_8 is reused with a precision of $\frac{2}{3}$ at the predecessor. Consider a selection s_8'' comprising e_{11} and e_8 for ω_3^{Q3}. For the sake of the example, consider a second selection s' for an operator ω_3^{Q4} comprises e_8 and e_{11}. In the example m_o, m_s, and $m_{s'}$ is 2, which means precision evaluates to 1, although the selection of ω_3^{Q3} relies on reused events. Hence, we annotate the calculated similarity and number of actual atomic events on which it depends to each outgoing event for each selection that reuses the result. The system can now calculate m_o as the sum of all annotated similarities of events from a common predecessor comprised in the selection, each weighted with the annotated number of atomic events. To this end, m_s, m_s' are also weighted with the corresponding sum of annotated numbers of atomic events. In our example, e_8 is annotated with a precision of $\frac{2}{3}$ and that it depends on 3 atomic event (e_1 through e_3), while e_{11} is annotated with $\frac{1}{2}$ and depends on 2 atomic events, thus $m_o = 3\frac{2}{3} + 3$. Since similarity degrades over all levels in the operator graph and the probability to have events from common predecessors also degrades, this method is designed for operator graphs with few levels in their operator graph.

Coarse-grained spatio-temporal Similarity: This approach annotates each event with the information about atomic events it depends on. In particular, it coarsens the annotated information about atomic events over time and space in a spatio-temporal grid data structure. The grid divides the spatial interest into distinct cells. Each cell then comprises the number of occurred atomic events during a time-span, e.g., during the start and end of a selection. Note, a sequence of time-spans can further partition the cell temporally. Atomic events can thus be aggregated over the course of a selection at the leaf operators and the grid is then propagated in a leaf to root direction with the events. Consider for our running example the grid depicted in Figure 5.3, the spatial interest of $Q2$ would be roughly divided by a grid in two halves, e_3 is comprised in one cell, e_1 and e_2 would be comprised in another cell. At each level of the operator graph, all annotated grids of events that are comprised in a selection are then joined and annotated to the outgoing events to represent the estimates for the similarity calculation. In particular, e_8 would comprise a grid where e_3 would be represented by the count 1 in one cell, and e_1 and e_2 are represented by a count 2 in a second cell. Congruent cells are joined by summing up the corresponding numbers of atomic events. At ω_3^{Q3}, the grid of e_8 is joined with one

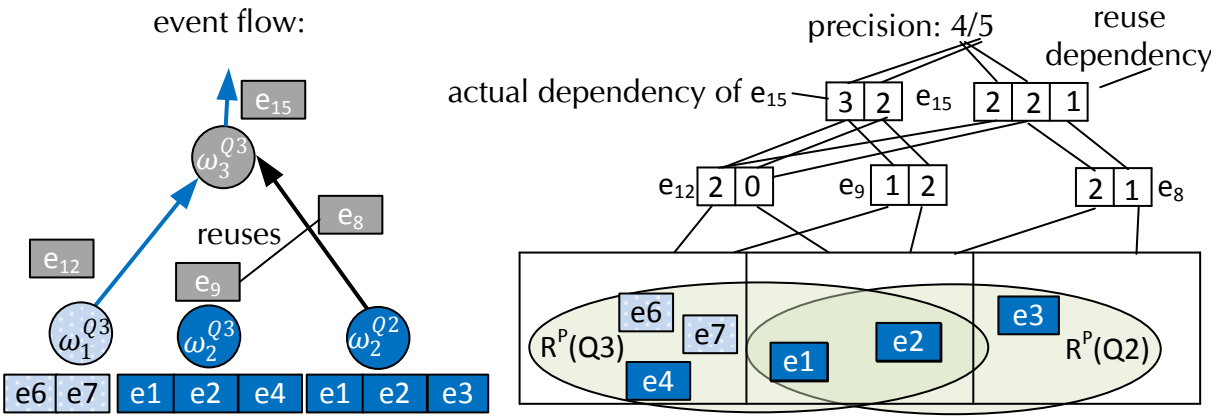


Figure 5.3: Example for coarse-grained spatio-temporal similarity. Operator ω_3^{Q3} detects event e_{15} based on the reused event e_8 from ω_2^{Q2} and e_{12}. Each event is assigned a grid which represents the aggregated number of atomic events on which the event depends on in the specific region. This allows us to estimate the similarity at operators on a higher level like ω_3^{Q3}.

from e_{12}; the congruent cell aggregates e_1 and e_2. In the following, we denote these grids as reuse dependency. Moreover, events comprised in a selection can already depend on reused selections. This requires that the actual information about atomic events that a selection would depend on without reusing must also be annotated in the same way—denoted as actual dependency. In the example, e_1, e_2, e_4, e_6, and e_7 are aggregated as the actual dependency of the selection. For a pair of selections s, s', the system can now use the actual dependency of s and the reuse dependency of s' to calculate the similarity according to the metrics given in Section 5.2. Cells with the same spatial coordinates comprise the number of events in the overlap. For example, we can determine e_{15}'s precision, the aggregate of the overlap in atomic events at ω_3^{Q3} is 4 (all aggregated events in the leftmost cells of e_{15}'s reuse dependency), while the selection depends on 5 atomic events in the actual dependency, implying a precision of $\frac{4}{5}$. Depending on the granularity of the size of the cells, more or less atomic events are compressed with the spatial grid. The programmer can thus trade-off accuracy in the calculated similarity against overhead for maintaining estimates with the grid size.

Prediction. Due to the asynchronous processing of selections (see Section 3.2.2), not all events that are comprised in a selection or batch might be present in the incoming buffers when calculating the similarity. In this case, we propose to predict the similarity. Moreover, it can be computationally less intensive to predict the similarity from a sample of annotated estimates, instead of calculating precision and recall over all annotated estimates.

Pessimistic and optimistic approaches are stateless and assume that either no event (pessimistic) or all events (optimistic) that arrive in the future for a batch lie in overlaps

of spatial interests of queries. For example, consider that the covering task's regular execution is triggered and some events already arrived for a selection s, e.g., e_1 and e_2 for s_3 in the example, while e_4 is missing (see Table 5.1). Moreover consider e_1, e_2, and e_3 already arrived for s_2 and the selection can be processed by the operator. The system can now compare s_3 and s_2. In particular, it determines that e_1 and e_2 lie in the spatial overlap of the corresponding processing interests. In the pessimistic case the system assumes that the missing event of s_3 does not lie in the overlap, the similarity would therefore evaluate to $\frac{2}{3}$. In the optimistic case, our system assumes that the missing event of s_3 lies in the overlap and evaluates the similarity to 1.

Event pattern prediction. This approach assumes that the past event pattern of selections indicates the future event pattern. For instance, all vehicles typically slow down when approaching a crossing at a distinct location. Therefore, the most recent received estimates about atomic events are buffered, e.g., the ids and the locations of e_0, e_1 and e_2 are buffered for our running example Figure 5.1. Consider again the case where event e_4 is missing to complete selection s_3. The estimation of future atomic events comprised in a pair of selections can then be determined based on the buffered estimates using an urn model. In order to estimate a similarity metric for a selection s, we distinguish events that lie in the spatial overlap of queries as one type of balls and events that do not stem from the overlap as another type of balls, e.g., three balls (e_0, e_1 and e_2) from the overlap of $Q1$ and $Q2$ are in one urn. Let $m_{s'}$ be the number of missing events in a selection, e.g., 1 in the example, then the event arrival can be modeled as drawing $m_{s'}$ balls from the urn. In particular, $k = 1$ balls have to be drawn of the type representing the spatial overlap to achieve a similarity of 1. The probability to draw k balls can be determined using a hypergeometric distribution. Let K be the atomic events in the interest overlap of the buffered estimates (3 events in the example) and M all events of the buffered estimates of the processing neighbor (also 3 events in the example), then the similarity can be predicted to $\frac{k_e}{m_{s'}}$. This results from the expectation value of $k_e = m_{s'}\frac{K}{M}$. This means since $m_{s'}$ atomic events are missing for the determination in the example, we expect that $k_e = 1$ will be drawn from the overlap. The corresponding probability to draw k or more atomic events can then be calculated using a hypergeometric distribution:

$$p_t < P(X > k) = \sum_{j=k}^{n} \frac{\binom{K}{j} * \binom{M-K}{n-j}}{\binom{M}{n}}$$

Location prediction. To support long batches, this approach considers the mobility-patterns of consumers. In particular, it anticipates the next processing interests $R^p(\Omega)$ of operators $\omega \in \Omega$ using location prediction methods, similar to Section 3.4. Based on recent atomic events in the predicted processing interests and the event pattern prediction method, we can anticipate an average similarity.

Discussion. To avoid vast amounts of annotated ids in order to estimate the similarity at each operator, we proposed three methods. (i) A light weight method based on the

spatial intersection of overlaps, which tends to be highly inaccurate since actual event distributions are not considered. (ii) An approach that approximates the similarity based on the similarity of selections at predecessors, which gives a better approximation, yet increases the size of events, since similarity related information need to be annotated to events. (iii) An approach that aggregates atomic event information in a grid data structure, which allows us to dynamically trade-off accuracy for a minimized size of annotated similarity information by selecting different grid sizes.

While pessimistic and optimistic approaches are naive and expected to highly deviate from the actual similarities, they do not need to maintain additional state and are computational inexpensive. Event pattern predictions need to maintain state to keep information about the past event patterns, however, can predict with a higher accuracy if the average similarity of selections if the past event pattern is similar to the future event pattern. Location predictions allow our system to even compare selections for future locations.

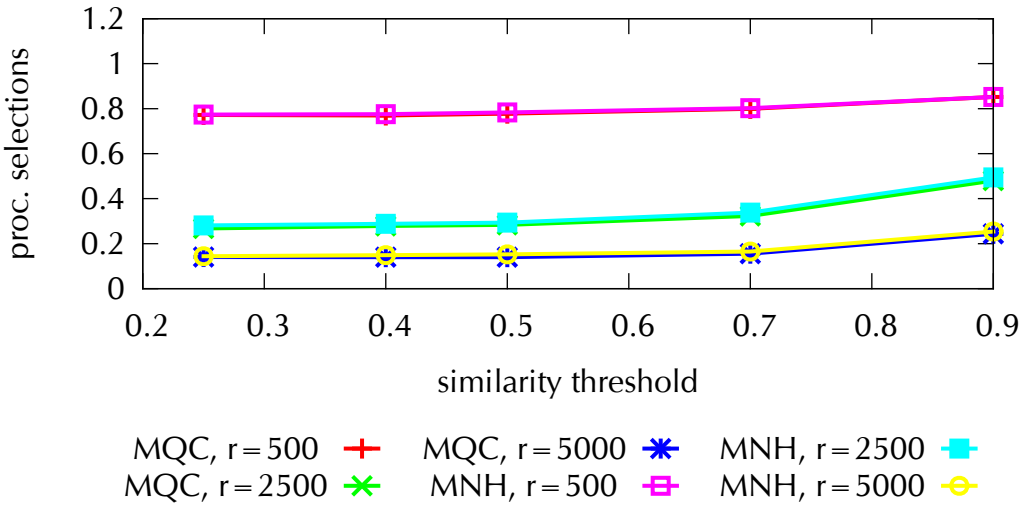
5.4 Evaluation

To show the benefits of our RECEP approach we evaluated our reuse methods with an operator graph that resembled the one of the traffic scenario (see Chapter 3) and compared it to the *baseline approach*—the basic MCEP approach. Moreover, to measure the benefit in terms of realistic processing costs, we considered a second setup in contrast to this *mobility setup*. In the second setup (*basic setup*) we determined the overhead of finding covering selections for different complex operators without mobility.

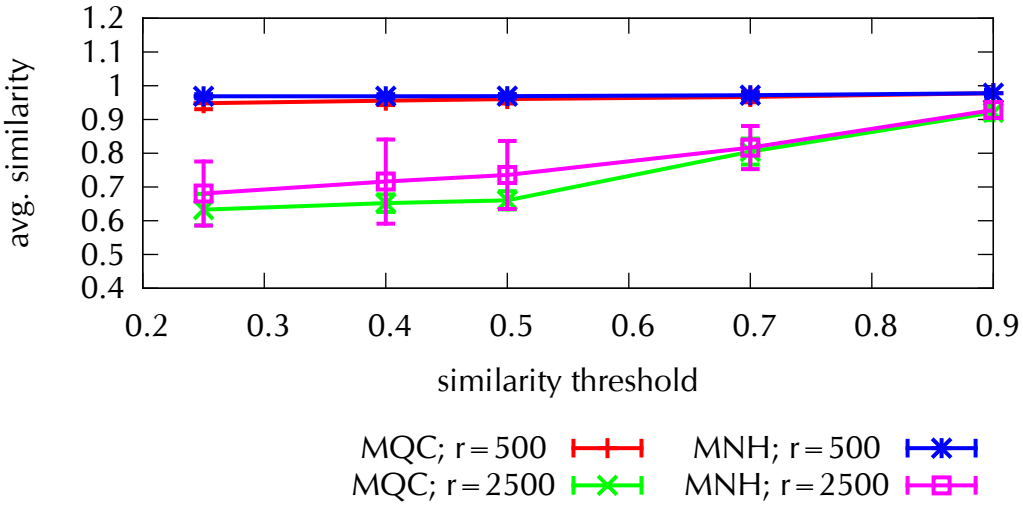
In the mobility setup, a number of vehicles (*numCars*) was selected at random as consumer with a square sized spatial interest. In order to test our approaches with different parameterizations for the operator, we used a generic count-based window to select the inputs to the operators. If not stated otherwise, we used all operators of the same type as processing neighbors, which allowed us to evaluate the full potential of RECEP before analyzing how the restrictions imposed by processing neighbors affect the system.

In the basic setup, we systematically derived a meaningful threshold for the throughput under synthetic workloads. We deployed one operator per query that implemented an average function at a random location in the service area. To emulate computational more complex operators, e.g., an operation that scans every pixel of an image that is comprised in an event, we assigned a parameter to the operator that allowed us to repeat the average operation several times on the same selection. We generated an evenly distributed workload of atomic events and spatial interests had a size of $\frac{1}{4}$ of the service area.

Since our main goal is to reduce the number of processed selections in order to save computations, we measured our computational savings in terms of actually processed selections (*proc. selections*). The output similarity was always determined according to the formulas given in Section 5.2 by annotating each event with the set of atomic events it



(a) Computational savings; batch execution; 25 cars



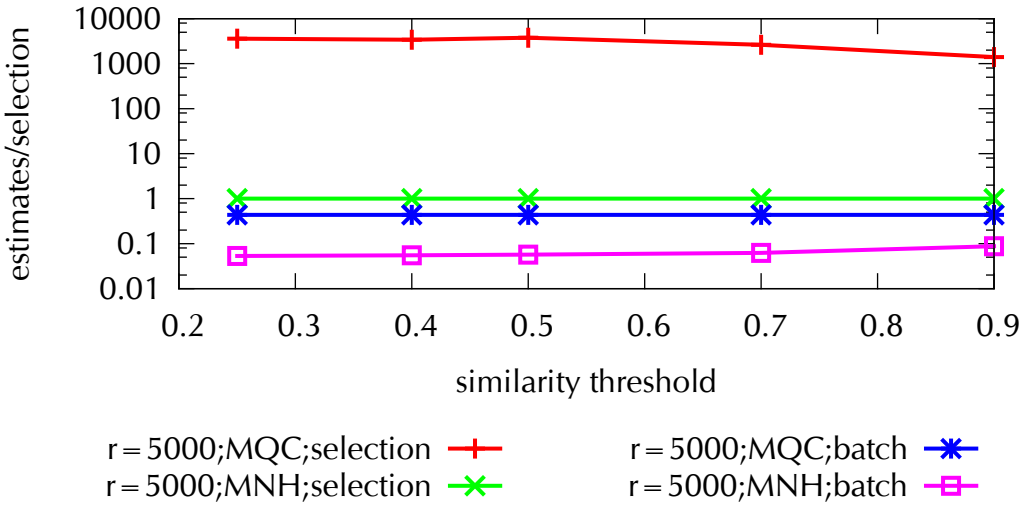
(b) Similarity; batch execution; 25 cars

Figure 5.4: Batch Execution Evaluation of the RECEP System

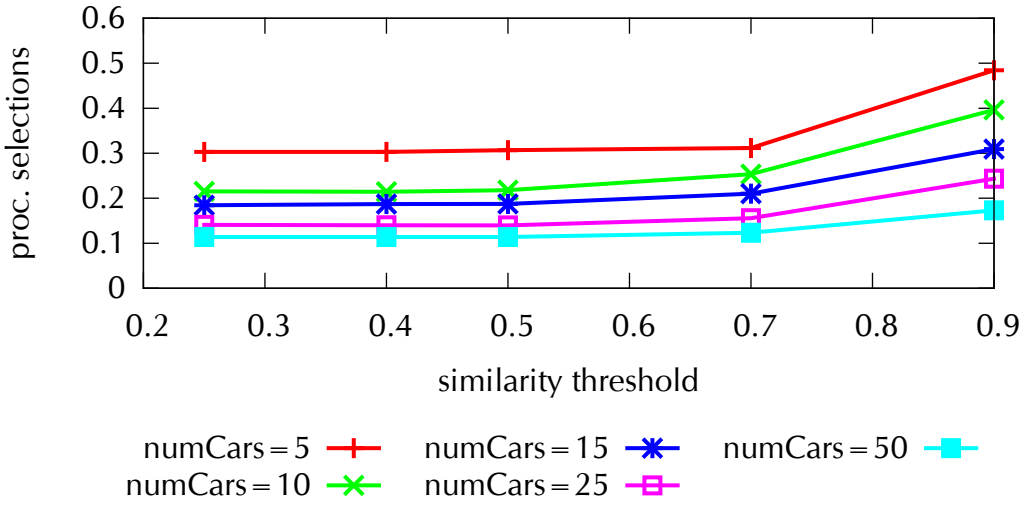
depends on, or in case of batches, with the average similarity of the batch. The actual measured similarity is presented as the average precision and recall over all detected situational information (*avg. similarity*). Most results are presented as relative results, in particular relative to the baseline, MCEP, without reuse.

Scalability of the Set Cover Heuristics.

With the mobility setup, we tested how much computational savings we can achieve with MQC and MNH (our basic approaches of Section 5.3.1), how much these approaches affect the similarity, and how significant the overhead is. We varied several control parameters. At first, the similarity threshold. Secondly, the side length (in m) of the spatial interest



(a) Overhead; 25 cars

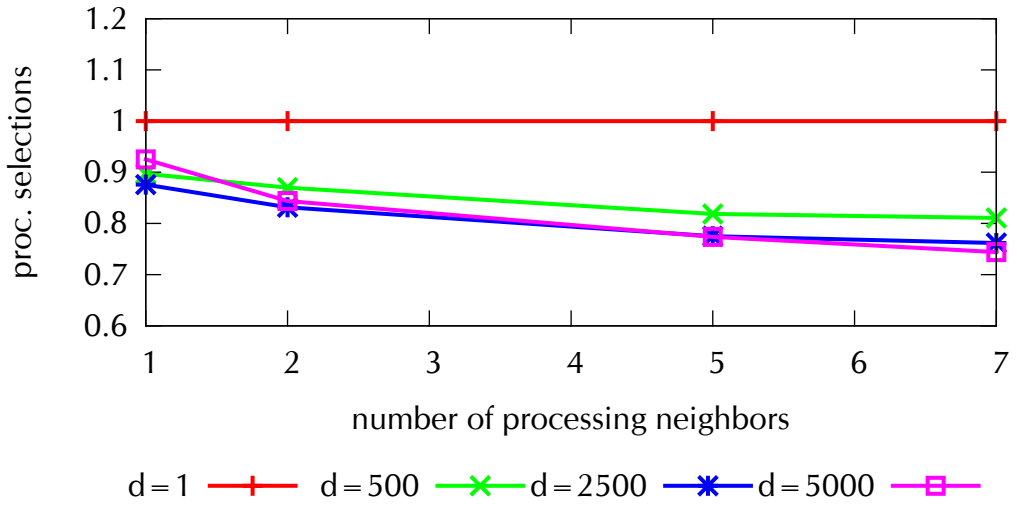


(b) Scaling; batch execuiton

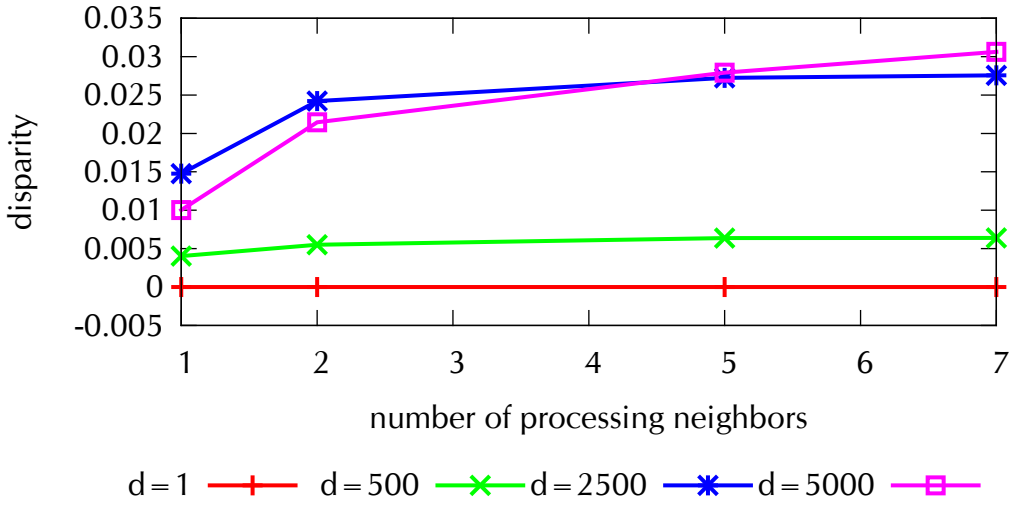
Figure 5.5: Scalability Evaluation of the RECEP System

(r). Note that with larger spatial interests the overlap of the interests increases. For these experiments we fixed the size of the temporal batches to 10 s and the number of cars that queried for traffic to 25, the frequency of initiating the processing phase was set to 1 s.

Figure 5.4a presents the computational savings when reusing batches of selections. The x-axis depicts the similarity threshold and the y-axis the fraction of actually processed selections in comparison to the number of processed selections in the baseline approach. When decreasing the similarity threshold, the system is able to reuse by far more selections since more selections can cover each other. However, it can only reuse if the interest overlap is high enough, e.g., in the case for $r = 500$ m the interests hardly overlap and nearly no reuse is possible. Figure 5.4b presents the effect of the reuse on the actually measured average similarity. Since all operators of an operator graph cooperatively keep



(a) Impact of processing neighbors on computational savings; 15 vehicles



(b) Impact of processing neighbors on disparity; 15 vehicles

Figure 5.6: Impact of Processing Neighbors

the threshold, the average similarity always remains above the threshold with a low standard deviation (depicted with the error-bars). The overhead imposed by comparisons of selections is depicted in Figure 5.5a on the y-axis as the number of similarity estimations performed by the heuristics over the number of selections that were actually covered by another selection. Here, we also compare the results for reusing individual selections to the case when reusing batches. A key observation over all evaluations is that the computational savings of the MQC are slightly better than for the MNH, however, with the downside of incurring more overhead.

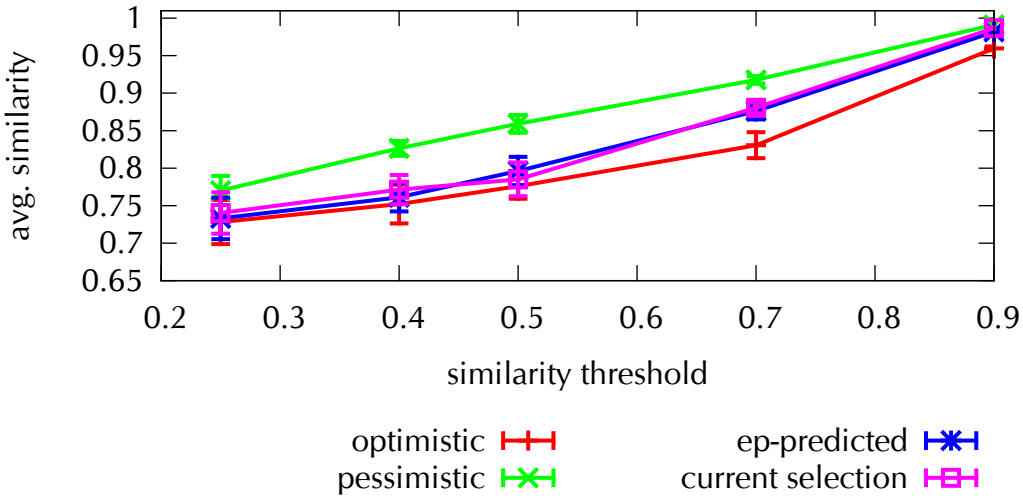


Figure 5.7: Impact of Similarity Prediction; 5 cars

Number of Overlapping Queries.

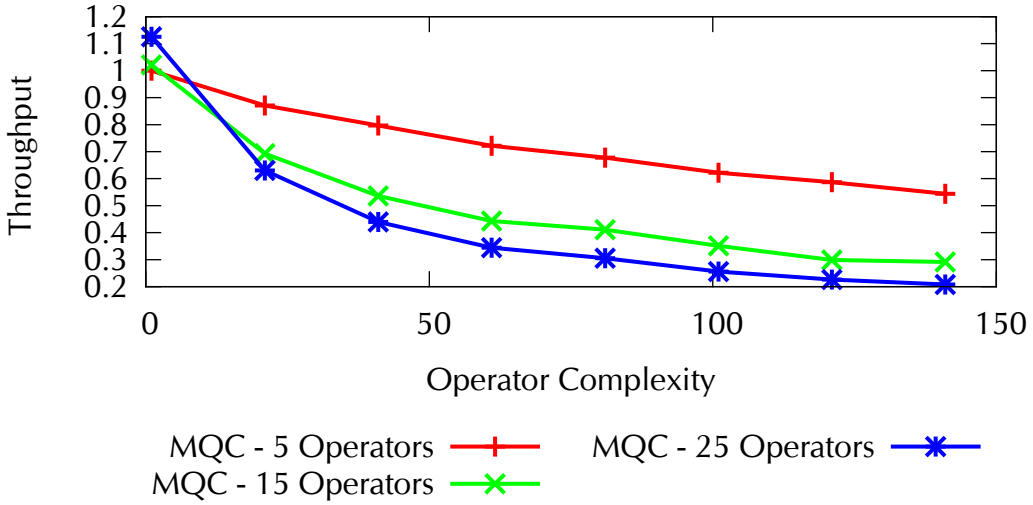
We also varied the number of vehicles (*numCars*) that queried for traffic. The higher *numCars*, the more operators are deployed and more queries overlap on the very same or similar spatial interest. We used the MQC for selection batches, while fixing the spatial interest to 5000 m.

The results in Figure 5.5b depict the relative computational savings in comparison to the baseline approach (y-axis) for different similarity thresholds (x-axis). Due to high overlaps in the interest of queries, selections can be reused by more operators if more queries are deployed, resulting in a lower percentage of processed selections.

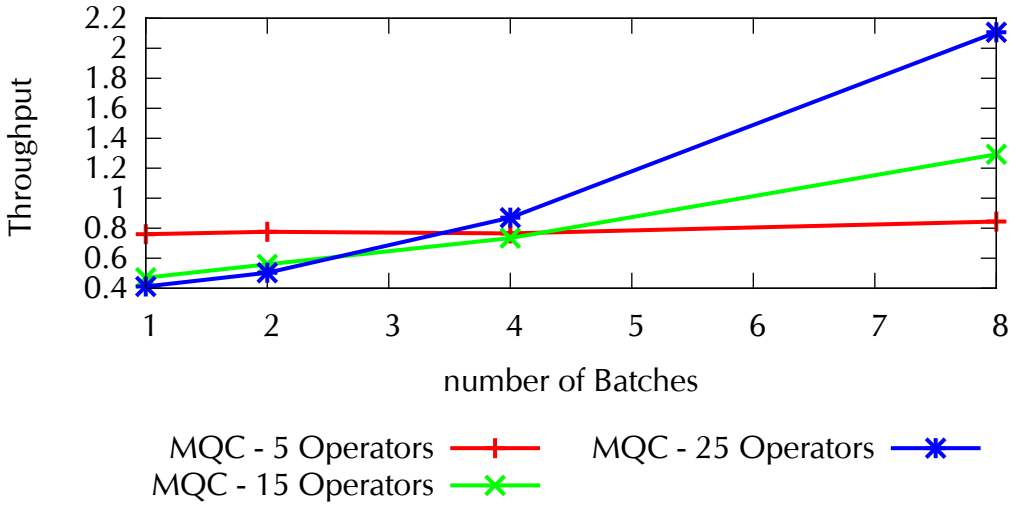
Impact of Processing Neighbors

Since increasing the number of processing neighbors (k_p) increases the potential to find operators to reuse with and the spatial distance d between processing neighbors affects the disparity (see Section 5.3.2), we also tested the effects of both parameters on the computational costs with the mobility setup. In fixed time intervals of 2 s new processing neighbors were selected. In this experiment, we fixed the similarity threshold to 0.5, the spatial interest to 5,000 m and reused individual selections with the MNH heuristic among 15 cars.

Figure 5.6a depicts on the x-axis the maximal number of selected processing neighbors, on the y-axis the number of processed selections relative to the baseline approach. The savings gradually increase with the number of neighbors since more reuse is allowed, however, as depicted in Figure 5.6b, the disparity drops with the number of neighbors and their relative distance to each other.



(a) Throughput ratio over operator complexity



(b) Throughput ratio over batch sizes

Figure 5.8: Efficacy of the RECEP System

Impact of Similarity Prediction Methods

To study the impact of the different similarity prediction methods (see Section 5.3.3) we conducted an experiment with the mobility setup, where the number of cars that queried for traffic was fixed to 5 and r was set to 5,000 m.

Figure 5.7 shows how the different similarity predictions affect the actually measured similarity (y-axis) for different similarity thresholds (x-axis). The optimistic approach reused many selections with actually bad qualities, since it always overestimated the similarity at run-time. The pessimistic approach missed many reuse opportunities, but only reused selections with good qualities. The event pattern-prediction, and an approach that evaluated the similarity over the events in *current selection* settled between both approaches.

Efficacy of the Set Cover Heuristic

The basic setup was used to determine how much delay is induced by applying the set cover heuristics before actually processing the selections. The operators had to process a set of 10,000 selections and sliding windows that each comprised 30 events. We used the event pattern prediction method to reduce the overhead for estimating the similarity and a naive implementation of the MSH.

The results depicted in Figure 5.8a show how the ratio of the throughput without reusing to the throughput by applying the MQC changed with the complexity of the operator, i.e., how often the average was computed (*Operator Complexity*). We also varied the number of deployed operators in that area. Our system clearly benefits from a high number of operators, since many operators can reuse the processing. Moreover, the higher the complexity of the operator, the better the performance of our approach, since the overhead of finding covering selections is always the same and eventually amortizes. Figure 5.8b depicts how the throughput ratio changed with the number of batches. More batches require more comparisons and estimations of the similarity, hence our system preforms especially well when the number of batches that have to be reused per operator is small.

Discussion

Our simulation showed the benefits of our approach in reducing the number of actually processed selections by up to 90% while maintaining a high average similarity for situational information of above 0.65. We also confirmed our analytical results, that the MNH algorithm reduces nearly the same amount of processed selections (around 1% worse than MSH) while reducing the overhead, the number of comparisons between selections, by an order of magnitude of 100. Moreover, we showed that the more queries are deployed and overlap in our system the higher the reduction in processed selections, since the same selection can be reused by more operators. Furthermore, we showed that the number of processed selections shrinks with the number of processing neighbors, as more possibilities for reuse can be exploited. However, we also show that an increased number of processing neighbors increases the disparity.

5.5 Related Work

Reuse has already been studied for a broad variety of queries. For example, the shared execution of queries and reusing partial results is a common technique in location-based queries to improve the query latency, scalability, computational load, or bandwidth [HM12a, GL06, XMA05]. For example, if two range-queries that provide a consumer with objects overlap in their spatial interest, the result of the overlap can be cached and reused. However, these reuse techniques are highly specialized for the individual queries and not tailored for a more general reuse approach. In CEP, reusing results

from an overlap can lead to false negatives and false positives, e.g., when aggregates are computed separately on the overlap and the non-overlapping area.

Multi Query Optimization for one-shot queries have been studied in data-base systems [PS11] and Data Mining [JSA05]. Moreover, CEP and (streaming) data base systems [XLT06, HFAE03, CDTW00, RSSB00] typically enable reuse by finding completely overlapping sub-sets of operators that process the same input events. However, this is not suitable for overlapping interests in sensor data. The similarity can degrade arbitrarily, e.g., if no event of interest lies in the overlap. Other methods that can work on streams that comprise similar input event [AG12, HRK+09, KWF06, KFHJ04, BDF+07] or share processing in a setting with parametrized operators [YRW09] are mostly tailored towards specialized operators and not tailored towards a general solution as presented in this thesis. Consider that aggregates could be processed on the overlapping and non-overlapping interests, and then combined by applying the same aggregation again. However, this is not applicable for all operators, e.g., averaging the temperature over two averages from two sub-ranges is not the same result as the average temperature of the whole range. Moreover, while these methods are tailored to provide exact results, our approach allows to control the degradation in the similarity.

Filter operations [MSHR02, TKK+11, XJ09] allow for reuse with respect to containment relations. For instance, in distributed publish/subscribe systems routing paths are merged if filters overlap. This reduces the bandwidth and computing costs. Take for example two subscribers, one that filters for events with temperatures > 20 and another one that is connected to the same host for events > 30. If on a preceding host an event with temperature 33 arrives, only one filter (> 20) has to be evaluated reducing the computing costs and the event has only to be sent once. However, this is only possible because filters are per-event operations and do not change the content of an event.

The sharing technique for cyber foraging presented in [VSDTD13] allows to efficiently share computations of several components in a video application. This works well, since these components are designed for incremental updates. However, our goal was to provide a more general sharing technique. Similar to our selection manager, a MJoin operator [RZRD09] takes all streams of join operators of the same query. However, with a different goal: the reordering of operators.

5.6 Conclusion

In this chapter, we presented a method for sharing computations between stateful operators in distributed CEP systems—the RECEP framework. In the context of a MSA application we showed the potential of RECEP in decreasing the computational overhead and resources needed by CEP systems in meeting quality requirements of consumer. Our method exploited two inherent characteristics of many CEP systems: overlapping interests in sensor data and the fact that slightly inaccurate results are acceptable in many application scenarios. Besides introducing the basic algorithms in maximizing the number of selections that can be reused, we proposed a comprehensive

set of run-time mechanisms that ensure the feasibility of our approach in a large-scale and highly dynamic deployment.

6 Conclusion

In this chapter, we provide an overview of the results of this thesis and an outlook on possible research directions in this field.

6.1 Summary

The abundance of sensors in the environment enables many exciting new applications. Mobile situation awareness applications are one such class of applications, which furthermore have a spatio-temporal reference and are event-based as well as latency-sensitive. Event-based applications are typically programmed using a CEP model. CEP systems are widely adopted, e.g., Twitter uses a stream processing engine for discovery, realtime analytics, personalization, search, revenue optimization, and more [Sto14]. Companies like IBM [AGWY11], Oracle [Ora15], or Microsoft [GHAB07] provide CEP solutions. However, it is challenging to implement a CEP system when it includes mobile consumers with an interest in nearby, recent situational information. The work presented in this thesis addresses these challenges with our MCEP middleware system. Consumers register a MCEP query with the system and in return are informed about situational information of interest. The situational information is detected by a so-called operator graph, which processes atomic events that occur in a dynamically changing spatial range.

The mechanisms behind the MCEP system improve the ability to detect and react to events in the presence of a very large number of mobile consumers and producers of mobile situation awareness applications. For instance, it is necessary to update the spatio-temporal range as mobile consumers move through time and space in order to keep the results detected by the operator graph relevant. Our findings have shown how dynamic operator reconfigurations can be performed efficiently while maintaining consistent results by exploiting a window-based execution model. Our evaluations have demonstrated that this way resource savings of up to 99% can be achieved, in the context of a traffic scenario.

Mobile situation awareness applications need to be informed about recent historical situational information as well as current, live situational information to react to all situations of interest. The processing of recent historical events causes a delay until live situational information can be detected, whenever the location of the consumer changes. We have proposed methods to address the problems caused by this delay. The metrics of interest that we improve are the precision and recall with respect to atomic events that represent the input to the MCEP query's operator graph. In particular, we propose a method for the opportunistic computation of historical events, based on a prediction of

the consumer's future locations. We also propose a pipelining method to look several steps into the future which enables us to process large numbers of historical atomic events. An opportunistic computing method that over-provisions operator graphs allows us to compensate for partially inaccurate location prediction results. With these methods we can achieve near zero latency for delivering historical situational information while our system achieves a precision of up to 0.8.

We have also shown how to minimize the resource usage by proposing methods for the opportunistic deployment and migration of operators. In particular, we demonstrated that it pays off to plan migrations ahead of time according to the mobility of users or dependent migrations. Costly migrations of state, in terms of network utilization, can be amortized by selecting suitable targets in a time-graph data structure that models the expected costs. Furthermore, we present how application knowledge improves live-migration systems. We exploit a window-based execution model to infer which state has to be transferred for a better serialization of operators. This way, additional resource savings of up to 40% are possible.

In the context of mobile situation awareness applications, we have also shown the potential of MCEP in decreasing the computational overhead and resources needed for the detection of situational information through reuse. Our method exploits two inherent characteristics of many mobile situation awareness scenarios: overlapping interests of multiple consumers in situational information and the fact that slightly inaccurate results are often acceptable. Besides introducing the basic algorithms for maximizing the number of selections that can be reused, we propose a comprehensive set of run-time mechanisms that ensure the feasibility of our approach in a large-scale and highly dynamic deployment. These methods allow us to reduce the number of actual event correlations of a situation detection by up to 90%.

6.2 Outlook

There are several possible paths to extend the work presented in this thesis in future research. One is to support real-time processing of big data streams, e.g., through parallel processing. Another direction is to off-load more tasks to mobile devices and even integrate them into the processing, e.g., in order to save energy. We briefly discuss the opportunities and challenges of these directions in the remainder of this section.

Real-time Processing of Big Data Streams

At the point in time when applications react to events, they trigger decisions like buying or selling stocks, updating a traffic route, or changing the configuration in power consumption. Therefore, the utility of such decisions is strongly dependent on (i) the time at which events are delivered by the CEP system, and (ii) the view of the application on the set of delivered events at decision time, which should be consistent, i.e., at time of decision all relevant events are present (no false negatives) and no false events (no

false positives) are delivered. For instance, taking a turn without receiving an event about traffic congestions or reacting late to the occurrence of such an event restricts a consumer in its ability in finding optimal traffic routes. Applications can only react fast and consistently to events if the CEP system can ensure the detection of events with high probability within an application-definable latency bound. This is highly challenging since many applications like monitoring the New York Stock Exchange (NYSE) encompass very high event rates. Over the trades and quotes, many events such as changes in their short term or long term trend can be detected with CEP operators to find specific patterns as an anomaly in stock trading or to support algorithmic trading. However, work on measuring the performance of such operators [BDWT13] shows that the sequential design—inherent to most state-of-the-art CEP systems—achieves at best a processing throughput of 10,000 events/s (using commodity hardware of a data center). Therefore, efficient parallel execution of dozens of processing entities is a necessity to speed up CEP systems such that they are capable of meeting application-defined latency bounds.

The critical issue of data parallelization is to efficiently determine suitable ways to partition the event stream into substreams, where the latter need to comply with the stateful processing steps performed by CEP operators and ensure the produced event streams become indistinguishable from a sequential execution. Finding the appropriate partitioning is part of ongoing work at the University of Stuttgart [MKR14, MKR15]. This approach is best described with a split-process-merge architecture, which allows for dynamically adapting the degree of parallelization depending on the event arrival rate and the operator's latency budget. The splitter preprocesses the incoming event streams and determines the points at which a new substream starts and ends. Then, the substreams are assigned and forwarded to operator instances depending on their load.

One problem that needs to be addressed in that context is the task of assigning selections to operators. Streaming subsequent overlapping selections to distinct operator instances will impose redundancy in communication since overlapping events will need to be streamed multiple times.

To be able to configure and adapt parallel operators that are able to meet a given latency budget, we need to model the latency imposed by a specific operator configuration. For example, a latency model is needed to decide which configurations are best suited to achieve a given latency. In addition, also at run-time we rely on the latency model in deciding when a configuration becomes critical with respect to the given latency bound.

Detection on Mobile Devices

Over the last decades, the world suffered from devastating catastrophes, like the nuclear disaster at the Fukushima Nuclear Power Plant [AP11] or the 2010 Haiti earthquake [CNN10]. Event based systems offer the means to support rescue teams and victims in such tragic situations. For example, CEP systems can detect potential dangerous situational information, like aftershocks of an earthquake [LOBC12] by aggregating accelerometer events from many distributed sensors. State-of-the art MCEP systems rely

on an infrastructure support (for example, the one discussed in Chapter 2) to route sensor data from the site of the disaster to a computing resource that processes events and to transport situational information back again.

Although mobile communication infrastructures are widespread nowadays, in times of crises these infrastructures are also prone to severe damage and may only be partially available or not at all. Through advances in wireless communication technologies, mobile devices are able to form ad-hoc networks as partial replacement of the infrastructure. For example, events from distributed sensors can easily be disseminated in such an ad-hoc network [SMK12]. Moreover, modern mobile devices possess sufficient computational capabilities (see Section 1.1.1) to be able to detect situational information locally. This way, mobile CEP systems can easily be deployed in an ad-hoc network.

However, the detection of situational information on mobile devices in an ad-hoc network is even more challenging than a placement in a fixed infrastructure. First, detections must deal with limited resources, since mobile devices have fixed computational capacities and do not have the potential of data-centers to harness nearly limitless resources. Second, energy-efficient detections become crucial to ensure a long service time, since mobile devices are energy-constraint and typically rely on batteries that cannot be recharged at a disaster site. Third, the detection must be robust to dynamic changes in the topology, since mobile devices are constantly moving in and out of the wireless communication range of other mobile devices and this way disrupting the mobile ad-hoc topology. Such topology changes may not only lead to varying communication paths, but also to network partitions, which may lead to a massive message and event loss which needs to be accounted for by the detection.

State-of-the art CEP systems perform load shedding when limited resources run out, i.e., they drop events [GWYL08]. However, mobile devices that are clustered over an ad-hoc network can share computing resources in order to provide fog-like, distributed computing resources. Yet, those resources are highly volatile due to the dynamics of the underlying topology. The interesting questions in that context are therefore: “How can a CEP system efficiently utilize resources of a cluster of mobile phones?”, “How can we minimize the data loss from disappearing volatile sources?”, and “How can we manage the CEP system in the mobile infrastructure with low energy consumption?”.

Moreover, to avoid sending large numbers of events over long paths in an ad-hoc topology and this way waste energy, a placement (see Chapter 4) which minimizes energy costs needs to be employed. In networks where an infrastructure is partially available the question is to decide whether to place an operator in the infrastructure or on a mobile device that represents the consumer. Initial work in that field [Bom14] has shown that an energy-efficient placement depends on the number and size of events, their frequency, and how computationally expensive a detection is. Concrete trigger points to make automated decisions remain open issues as well as mechanisms that decide when and how much state needs to be transferred from the infrastructure to the mobile device if the mobile device moves into an area without access to the infrastructure. The placement in a pure ad-hoc network requires to decide on which mobile devices to place the processing. Since

the topology is highly dynamic, it also remains an issue how to constantly and securely adapt the placement under high churn.

Approximated early results are often sufficient and preferable over concrete late results (see Chapter 5). Which means that a reasonable amount of event loss is manageable for a CEP system. However, the question remains if a CEP system can exploit this by purposefully avoiding to send events in order to save the energy for transmitting them?

List of Figures

List of Tables

List of Algorithms

Bibliography

[AAB+05] 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. The Design of the Borealis Stream Processing Engine. In *Proceedings of the 2nd Biennial Conference on Innovative Data Systems Research*, CIDR'05, pages 277–289, Asilomar, CA, January 2005.

[ABB+03] Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Rajeev Motwani, Itaru Nishizawa, Utkarsh Srivastava, Dilys Thomas, Rohit Varma, and Jennifer Widom. STREAM: The Stanford Stream Data Manager. *IEEE Data Engineering Bulletin*, 26(1):19–26, March 2003.

[ABB+14] Alexander Artikis, Chris Baber, Pedro Bizarro, Carlos Canudas-de Wit, Opher Etzion, Fabiana Fournier, Pauls Goulart, Andrew Howes, John Lygeros, Georgios Paliouras, Assaf Schuster, and Izchak Sharfman. Scalable Proactive Event-Driven Decision-Making. *Technology and Society Magazine, IEEE*, 33(3):35–41, Fall 2014.

[ABW06] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. *The VLDB Journal*, 15(2):121–142, June 2006.

[AC05] Raman Adaikkalavan and Sharma Chakravarthy. Formalization and Detection of Events Using Interval-Based Semantics. In *Proceedings of the 11th International Conference on Management of Data*, COMAD'06, pages 58–69. Computer Society of India, January 2005.

[ACc+03] Daniel J. Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, August 2003.

[AcT08] Mert Akdere, Uğur Çetintemel, and Nesime Tatbul. Plan-based Complex Event Detection across Distributed Sources. *Proceedings of the VLDB Endowment*, 1(1):66–77, August 2008.

[AE04] Asaf Adi and Opher Etzion. Amit - the situation manager. *The VLDB Journal*, 13(2):177–203, May 2004.

[AFFB12] Stefan Appel, Sebastian Frischbier, Tobias Freudenreich, and Alejandro Buchmann. Eventlets: Components for the Integration of Event Streams with SOA. In *Proceedings of the 5th IEEE International Conference on Service-Oriented Computing and Applications*, SOCA '12, pages 1–9, December 2012.

[AFG+10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A View of Cloud Computing. *Communications of the ACM*, 53(4):50–58, April 2010.

[AG12] Andinet Assefa and Fekade Getahun. Multi-Query Optimization for Semantic News Feed Query. In *Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, MEDES '12, pages 150–157, New York, NY, USA, 2012. ACM.

[AGKW14] Alexander Artikis, Avigdor Gal, Vana Kalogeraki, and Matthias Weidlich. Event Recognition Challenges and Techniques: Guest Editors' Introduction. *ACM Transactions on Internet Technology*, 14(1):1:1–1:9, August 2014.

[AGWY11] Henrique Andrade, Bugra Gedik, Kun-Lung Wu, and Philip S. Yu. Processing high data rate streams in System S. *Journal of Parallel and Distributed Computing*, 71(2):145–156, 2011. Data Intensive Computing.

[AH00] Ron Avnur and Joseph M. Hellerstein. Eddies: Continuously Adaptive Query Processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD'00, pages 261–272. ACM, 2000.

[AKF+14] Stefan Appel, Pascal Kleber, Sebastian Frischbier, Tobias Freudenreich, and Alejandro Buchmann. Modeling and Execution of Event Stream Processing in Business Processes. *Information Systems*, 46:140–156, 2014.

[AKTS11] Haidar Al-Khalidi, David Taniar, and Maytham Safar. Approximate Static and Continuous Range Search in Mobile Navigation. In *Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication*, ICUIMC '11, pages 16:1–16:10, New York, NY, USA, 2011. ACM.

[All83] James F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[Ama14] Amazon. http://aws.amazon.com/ec2/, 2014. [online; accessed 2015-09-15].

[AN04] Ahmed M. Ayad and Jeffrey F. Naughton. Static Optimization of Conjunctive Queries with Sliding Windows over Infinite Streams. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 419–430, New York, NY, USA, 2004. ACM.

[AN08] Yanif Ahmad and Suman Nath. COLR-Tree: Communication-Efficient Spatio-Temporal Indexing for a Sensor Data Web Portal. In *Proceedings of the IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 784–793, April 2008.

[AP11] BBC Asia-Pacific. Japan quake: Radiation rises at Fukushima nuclear plant. http://www.bbc.com/news/world-12740843, March 2011. [online; accessed 2015-10-03].

[ARIC12] Imad Afyouni, Cyril Ray, Sergio Ilarri, and Christophe Claramunt. Algorithms for Continuous Location-dependent and Context-aware Queries in Indoor Environments. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '12, pages 329–338, New York, NY, USA, 2012. ACM.

[BBD+02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and Issues in Data Stream Systems. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 1–16, New York, NY, USA, 2002. ACM.

[BBEK11] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. SUMO - Simulation of Urban MObility: An Overview. In *Proceedings of the 3rd International Conference on Advances in System Simulation*, SIMUL '11, pages 63–68, Barcelona, Spain, October 2011.

[BCD03] Brian Babcock, Surajit Chaudhuri, and Gautam Das. Dynamic Sample Selection for Approximate Query Processing. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 539–550, New York, NY, USA, 2003. ACM.

[BDF+07] Michael Branson, Fred Douglis, Brad Fawcett, Zhen Liu, Anton Riabov, and Fan Ye. CLASP: Collaborating, Autonomous Stream Processing Systems. In *Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware*, Middleware '07, pages 348–367, New York, NY, USA, 2007. Springer-Verlag New York, Inc.

[BDWT13] Cagri Balkesen, Nihal Dindar, Matthias Wetter, and Nesime Tatbul. RIP: Run-based Intra-query Parallelism for Scalable Complex Event Processing. In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, pages 3–14, New York, NY, USA, 2013. ACM.

[BGAH07] Roger S. Barga, Jonathan Goldstein, Mohamed Ali, and Mingsheng Hong. Consistent Streaming Through Time: A Vision for Event Stream Processing. In *Proceedings of the 3rd Conference on Innovative Data Systems Research*, CIDR '07, pages 363–374, Asilomar, CA, January 2007.

[BK09] Alejandro Buchmann and Boris Koldehofe. Complex Event Processing. *it-Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik*, 51(5):241–242, 2009.

[BKFS07] Robert Bradford, Evangelos Kotsovinos, Anja Feldmann, and Harald Schiöberg. Live Wide-Area Migration of Virtual Machines Including Local Persistent State. In *Proceedings of the 3rd Int. Conference on Virtual Execution Environments*, VEE '07, pages 169–179. ACM, 2007.

[BKR11] Andreas Benzing, Boris Koldehofe, and Kurt Rothermel. Efficient Support for Multi-Resolution Queries in Global Sensor Networks. In *Proceedings of the 5th International Conference on Communication System Software and Middleware*, COMSWARE '11, pages 11:1–11:12, New York, NY, USA, 2011. ACM.

[Bla12] Max Blau. Atlanta under surveillance. http://clatl.com/atlanta/atlanta-under-surveillance/Content?oid=7121394, December 2012. [online; accessed 2015-09-15].

[BMZA12] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the 1st MCC SIGCOMM Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16. ACM, 2012.

[Bom14] Tycho Bomancz. Platzierung und Migration von CEP-Operatoren in MANet Szenarien. Diplomarbeit, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, April 2014.

[BTJK04] Muhammad Mukarram Bin Tariq, Ravi Jain, and Toshiro Kawahara. Mobility Aware Server Selection for Mobile Streaming Multimedia Content Distribution Networks. In Fred Douglis and BrianD. Davison, editors, *Web Content Caching and Distribution*, pages 1–18. Springer Netherlands, 2004.

[Cal14] California Department of Transportation. Traffic Counts. http://traffic-counts.dot.ca.gov/, 2014. [online; accessed 2015-10-03].

[Cap13] Capgemini. My car, my way. http://www.capgemini.com/resources/cars-online-1213, 2013. [online; accessed 2015-09-15].

[CBD+12] Ben W. Carabelli, Andreas Benzing, Frank Dürr, Boris Koldehofe, Kurt Rothermel, Georg Seyboth, Rainer Blind, Mathias Bürger, and Frank Allgöwer. Exact Convex Formulations of Network-Oriented Optimal Operator Placement. In *Proceedings of the 2012 IEEE 51st Annual Conference on Decision and Control*, CDC '12, pages 3777–3782, December 2012.

[CBL+10] Muhammad A. Cheema, Ljiljana Brankovic, Xuemin Lin, Wenjie Zhang, and Wei Wang. Multi-Guarded Safe Zone: An Effective Technique to Monitor

Moving Circular Range Queries. In *Proceedings of the 2010 IEEE 26th International Conference on Data Engineering*, ICDE '10, pages 189–200, March 2010.

[CBL+11] Muhammad A. Cheema, Ljiljana Brankovic, Xuemin Lin, Wenjie Zhang, and Wei Wang. Continuous Monitoring of Distance-Based Range Queries. *IEEE Transactions on Knowledge and Data Engineering,*, 23(8):1182–1199, August 2011.

[CcC+02] Don Carney, Uğur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Monitoring Streams: A New Class of Data Management Applications. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 215–226. VLDB Endowment, 2002.

[CCD+03] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel Madden, Vijayshankar Raman, Frederick Reiss, and Mehul A. Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *Proceedings of the 1st Biennial Conference on Innovative Data Systems Research*, CIDR '03, pages 24:1–24:12, 2003.

[CDTW00] Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 379–390, New York, NY, USA, 2000. ACM.

[CEB+09] Nazario Cipriani, Mike Eissele, Andreas Brodt, Matthias Grossmann, and Bernhard Mitschang. NexusDS: A Flexible and Extensible Middleware for Distributed Stream Processing. In *Proceedings of the 2009 International Database Engineering & Applications Symposium*, IDEAS '09, pages 152–161, New York, NY, USA, 2009. ACM.

[CEL+08] Andrew T. Campbell, Shane B. Eisenman, Nicholas D. Lane, Emiliano Miluzzo, Ronald A. Peterson, Hong Lu, Xiao Zheng, Mirco Musolesi, Kristóf Fodor, and Gahng-Seop Ahn. The Rise of People-Centric Sensing. *IEEE Internet Computing*, 12(4):12–21, 2008.

[CF02] Sirish Chandrasekaran and Michael J. Franklin. Streaming Queries over Streaming Data. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 203–214. VLDB Endowment, 2002.

[CF04] Sirish Chandrasekaran and Michael Franklin. Remembrance of Streams Past: Overload-sensitive Management of Archived Streams. In *Proceedings of the 30th International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 348–359. VLDB Endowment, 2004.

[CFH+03] Mariano Cilia, Ludger Fiege, C. Haul, Andreas Zeidler, and Alejandro P. Buchmann. Looking into the Past: Enhancing Mobile Publish/Subscribe Middleware. In *Proceedings of the 2nd International Workshop on Distributed Event-Based Systems*, DEBS '03, pages 1–8, New York, NY, USA, 2003. ACM.

[CFMKP13] Raul Castro Fernandez, Matteo Migliavacca, Evangelia Kalyvianaki, and Peter Pietzuch. Integrating Scale out and Fault Tolerance in Stream Processing Using Operator State Management. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 725–736, New York, NY, USA, 2013. ACM.

[CHML14] Hyunseok Chang, Adiseshu Hari, Sarit Mukherjee, and T.V. Lakshman. Bringing the Cloud to the Edge. In *Proceedings of the 2014 IEEE Conference on Computer Communications Workshops*, INFOCOM WKSHPS '14, pages 346–351, April 2014.

[Cis13] Cisco. Continental and Cisco Show the Future of Connected Vehicles. http://newsroom.cisco.com/release/1236059/Continental-and-Cisco-Show-the-Future-of-Connected-Vehicles, 2013. [online; accessed 2015-10-03].

[Cis14] Cisco. Internet Of Everything. http://internetofeverything.cisco.com/, 2014. [online; accessed 2014-03-19].

[CM94] Sharma Chakravarthy and Deepak Mishra. Snoop: An Expressive Event Specification Language For Active Databases. *Data & Knowledge Engineering*, 14(1):1–26, 1994.

[CM10] Gianpaolo Cugola and Alessandro Margara. Tesla: a formally defined event specification language. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, DEBS '10, pages 50–61, New York, NY, USA, 2010. ACM.

[CM12a] Gianpaolo Cugola and Alessandro Margara. Low latency complex event processing on parallel hardware. *Journal of Parallel and Distributed Computing*, 72(2):205–218, 2012.

[CM12b] Gianpaolo Cugola and Alessandro Margara. Processing Flows of Information: From Data Stream to Complex Event Processing. *ACM Computing Surveys*, 44(3):15:1–15:62, June 2012.

[CM13] Gianpaolo Cugola and Alessandro Margara. Deployment Strategies for Distributed Complex Event Processing. *Springer Computing*, 95(2):129–156, 2013.

[CNN10] CNN Library. Haiti Earthquake Fast Facts. http://edition.cnn.com/2013/12/12/world/haiti-earthquake-fast-facts/, 2010. [online; accessed 2015-10-03].

[CSM11] Nazario Cipriani, Oliver Schiller, and Bernhard Mitschang. M-TOP: Multi-target Operator Placement of Query Graphs for Data Streams. In *Proceedings of the 15th Symposium on International Database Engineering & Applications*, IDEAS '11, pages 52–60, New York, NY, USA, 2011. ACM.

[CSRL01] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[DCKM04] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A Decentralized Network Coordinate System. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04, pages 15–26. ACM, 2004.

[DFST11] Nihal Dindar, Peter M. Fischer, Merve Soner, and Nesime Tatbul. 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, pages 243–254, New York, NY, USA, 2011. ACM.

[DHL09] Tau T. Do, Kien A. Hua, and Chow-Sing Lin. ExtRange: Continuous Moving Range Queries in Mobile Peer-to-Peer Networks. In *Proceedings of the 10th International Conference on Mobile Data Management: Systems, Services and Middleware*, MDM '09, pages 317–322, May 2009.

[dMLR07] Cédric du Mouza, Witold Litwin, and Philippe Rigaux. SD-Rtree: A Scalable Distributed Rtree. In *Proceedings of the 23rd International Conference on Data Engineering*, ICDE '07, pages 296–305. IEEE, April 2007.

[DR04] Luping Ding and Elke A. Rundensteiner. Evaluating Window Joins over Punctuated Streams. In *Proceedings of the 13th ACM International Conference on Information and Knowledge Management*, CIKM '04, pages 98–107, New York, NY, USA, 2004. ACM.

[DRH03] Luping Ding, Elke A. Rundensteiner, and George T. Heineman. MJoin: A Metadata-aware Stream Join Operator. In *Proceedings of the 2nd International Workshop on Distributed Event-based Systems*, DEBS '03, pages 1–8, New York, NY, USA, 2003. ACM.

[DS07] Peter J. Desnoyers and Prashant Shenoy. Hyperion: High Volume Stream Archival for Retrospective Querying. In *Proceedings of the 2007 USENIX Annual Technical Conference*, ATC'07, pages 4:1–4:14, Berkeley, CA, USA, 2007. USENIX Association.

[EEF12] Yagil Engel, Opher Etzion, and Zohar Feldman. A Basic Model for Proactive Event-driven Computing. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, pages 107–118. ACM, 2012.

[EFGK03] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[Eis12] Paul A. Eisenstein. In-Vehicle Navigation Sales Will Quadruple by 2019. http://www.thedetroitbureau.com/2012/07/in-vehicle-navigation-sales-will-quadruple-by-2019/, July 2012. [online; accessed 2015-10-03].

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, KDD '96, pages 226–231. AAAI Press, 1996.

[FCR07] Tobias Farrell, Reynold Cheng, and Kurt Rothermel. Energy-Efficient Monitoring of Mobile Objects with Uncertainty-Aware Tolerances. In *Proceedings of the 11th International Database Engineering and Applications Symposium*, pages 129–140, Washington, DC, USA, 2007. IEEE Computer Society.

[FR13] Josef Federman and Max J. Rosenthal. Waze sale signals new growth for Israeli high tech. http://news.yahoo.com/waze-sale-signals-growth-israeli-high-tech-174533585.html, June 2013. [online; accessed 2015-09-28].

[GAW+08] Bugra Gedik, Henrique Andrade, Kun-Lung Wu, Philip S. Yu, and Myungcheol Doo. SPADE: The System S Declarative Stream Processing Engine. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1123–1134, New York, NY, USA, 2008. ACM.

[GD94] Stella Gatziu and Klaus R. Dittrich. Detecting Composite Events in Active Database Systems Using Petri Nets. In *Proceedings of the 4th International Workshop on Research Issues in Data Engineering*, RIDE-ADS '94, pages 2–9, 1994.

[GHAB07] Jonathan Goldstein, Mingsheng Hong, Mohamed Ali, and Roger Barga. Consistency Sensitive Operators in CEDR. Technical Report MSR-TR-2007-158, Microsoft Research, December 2007.

[GL06] Bugra Gedik and Ling Liu. MobiEyes: A Distributed Location Monitoring Service Using Moving Location Queries. *IEEE Transactions on Mobile Computing*, 5:1384–1402, 2006.

[Goo14] Google. https://cloud.google.com/compute/, 2014. [online; accessed 2014-10-21].

[GSEFT13] Boris Glavic, Kyumars Sheykh Esmaili, Peter Michael Fischer, and Nesime Tatbul. Ariadne: Managing Fine-grained Provenance on Data Streams. In

Proceedings of the 7th ACM International Conference on Distributed Event-based Systems, DEBS '13, pages 39–50, New York, NY, USA, 2013. ACM.

[GWYL08] Bugra Gedik, Kun-Lung Wu, Philip S. Yu, and Ling Liu. MobiQual: QoS-aware Load Shedding in Mobile CQ Systems. In *Proceedings of the IEEE 24th International Conference on Data Engineering*, ICDE'08, pages 1121–1130, April 2008.

[GYGC09] Yu Gu, Ge Yu, Na Guo, and Yueguo Chen. Probabilistic Moving Range Query over RFID Spatio-temporal Data Streams. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pages 1413–1416, New York, NY, USA, 2009. ACM.

[Hay08] Brian Hayes. Cloud Computing. *Communications of the ACM*, 51(7):9–11, July 2008.

[HDG09] Michael R. Hines, Umesh Deshpande, and Kartik Gopalan. Post-Copy Live Migration of Virtual Machines. *ACM SIGOPS Operating Systems Review*, 43(3):14–26, July 2009.

[HEG12] Adrian Holzer, Patrick Eugster, and Benoit Garbinato. {ALPS} Adaptive Location-based Publish/Subscribe. *Computer Networks*, 56(12):2949–2962, 2012.

[HFAE03] Moustafa A. Hammad, Michael J. Franklin, Walid G. Aref, and Ahmed K. Elmagarmid. Scheduling for shared window joins over data streams. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, VLDB '03, pages 297–308. VLDB Endowment, 2003.

[Hir12] Martin Hirzel. Partition and Compose: Parallel Complex Event Processing. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, pages 191–200, New York, NY, USA, 2012. ACM.

[HLR+13a] Kirak Hong, David Lillethun, Umakishore Ramachandran, Beate Ottenwälder, and Boris Koldehofe. 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, pages 15–20, 2013.

[HLR+13b] Kirak Hong, David Lillethun, Umakishore Ramachandran, Beate Ottenwälder, and Boris Koldehofe. Opportunistic Spatio-temporal Event Processing for Mobile Situation Awareness. In *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems*, DEBS '13, pages 195–206, New York, NY, USA, 2013. ACM.

[HLSD11] Waldemar Hummer, Philipp Leitner, Benjamin Satzger, and Schahram Dustdar. Dynamic Migration of Processing Elements for Optimized Query Execution in Event-based Systems. In *Proceedings of the 2011th Confederated International Conference on On the move to meaningful internet systems - Volume Part II*, OTM'11, pages 451–468, Berlin, Heidelberg, 2011. Springer-Verlag.

[HM12a] Abdeltawab M. Hendawi and Mohamed F. Mokbel. Panda: A Predictive Spatio-Temporal Query Processor. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '12, pages 13–22, New York, NY, USA, 2012. ACM.

[HM12b] Abdeltawab M. Hendawi and Mohamed F. Mokbel. Predictive Spatio-Temporal Queries: A Comprehensive Survey and Future Directions. In *Proceedings of the 1st ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems*, MobiGIS '12, pages 97–104, New York, NY, USA, 2012. ACM.

[HMLJ09] Songlin Hu, Vinod Muthusamy, Guoli Li, and Hans-Arno Jacobsen. Transactional Mobility in Distributed Content-Based Publish/Subscribe Systems. In *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems*, ICDCS '09, pages 101–110, June 2009.

[Hon14] Kirak Hong. *A distributed framework for situation awareness on camera networks*. PhD thesis, Georgia Institute of Technology, 2014.

[HOR14] Kirak Hong, Beate Ottenwälder, and Umakishore Ramachandran. Scalable Spatio-temporal Analysis on Distributed Camera Networks. In Filip Zavoral, Jason J. Jung, and Costin Badica, editors, *Intelligent Distributed Computing VII*, volume 511 of *Studies in Computational Intelligence*, pages 131–140. Springer International Publishing, 2014.

[HRK+09] Mingsheng Hong, Mirek Riedewald, Christoph Koch, Johannes Gehrke, and Alan Demers. Rule-based Multi-query Optimization. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pages 120–131, New York, NY, USA, 2009. ACM.

[HSS+11] Kirak Hong, Stephen Smaldoney, Junsuk Shin, David Lillethun, Liviu Iftodey, and Umakishore Ramachandran. 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, pages 1–8, August 2011.

[HSS+14] Martin Hirzel, Robert Soulé, Scott Schneider, Buğra Gedik, and Robert Grimm. A Catalog of Stream Processing Optimizations. *ACM Computing Surveys*, 46(4):46:1–46:34, March 2014.

[HV03] Annika Hinze and Agnès Voisard. Location- and Time-Based Information Delivery in Tourism. In Thanasis Hadzilacos, Yannis Manolopoulos, John Roddick, and Yannis Theodoridis, editors, *Advances in Spatial and Temporal Databases*, volume 2750 of *Lecture Notes in Computer Science*, pages 489–507. Springer Berlin Heidelberg, 2003.

[HW08] Mordechai Haklay and Patrick Weber. OpenStreetMap: User-Generated Street Maps. *IEEE Pervasive Computing*, 7(4):12–18, December 2008.

[ibe15] ibeaconinsider. [what is ibeacon? what are ibeacons?]. http://www.ibeacon.com/what-is-ibeacon-a-guide-to-beacons/, 2015. [online; accessed 2015-10-03].

[JJE10] K. R. Jayaram, Chamikara Jayalath, and Patrick Eugster. Parametric Subscriptions for Content-based Publish/Subscribe Networks. In *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware*, Middleware '10, pages 128–147, Berlin, Heidelberg, 2010. Springer-Verlag.

[Joh73] David S. Johnson. Approximation Algorithms for Combinatorial Problems. In *Proceedings of the 5th annual ACM Symposium on Theory of Computing*, STOC '73, pages 38–49, New York, NY, USA, 1973. ACM.

[JSA05] Ruoming Jin, Kaushik Sinha, and Gagan Agrawal. A Framework to Support Multiple Query Optimization for Complex Mining Tasks. In *Proceedings of the 6th International Workshop on Multimedia Data Mining: Mining Integrated Media and Complex Data*, MDM '05, pages 23–32, New York, NY, USA, 2005. ACM.

[KFHJ04] Sailesh Krishnamurthy, Michael J. Franklin, Joseph M. Hellerstein, and Garrett Jacobson. The Case for Precision Sharing. In *Proceedings of the 30th International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 972–984. VLDB Endowment, 2004.

[KKR08] Gerald G. Koch, Boris Koldehofe, and Kurt Rothermel. Higher Confidence in Event Correlation Using Uncertainty Restrictions. In *Proceedings of the 28th International Conference on Distributed Computing Systems Workshops*, ICDCS '08, pages 417–422, June 2008.

[KKR10] Gerald G. Koch, Boris Koldehofe, and Kurt Rothermel. Cordies: Expressive event correlation in distributed systems. In *Proceedings of the 4th ACM International Conference on Distributed Event-Based Systems*, DEBS'10, pages 26–37. ACM, 2010.

[KMR+13] Boris Koldehofe, Ruben Mayer, Umakishore Ramachandran, Kurt Rothermel, and Marco Völz. Rollback-Recovery without Checkpoints in Distributed

Event Processing Systems. In *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems*, DEBS '13, pages 27–38. ACM, 2013.

[KNV03] Jaewoo Kang, Jeffrey F. Naughton, and Stratis D. Viglas. Evaluating Window Joins over Unbounded Streams. In *Proceedings of the 19th International Conference on Data Engineering*, ICDE '03, pages 341–352, March 2003.

[KORR12] Boris Koldehofe, Beate Ottenwälder, Kurt Rothermel, and Umakishore Ramachandran. Moving Range Queries in Distributed Complex Event Processing. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, pages 201–212. ACM, 2012.

[Kre15] Stefan Krempl. Big Data als Allgemeingut: Daten sind Macht. http://heise.de/-2621166, April 2015. [online; accessed 2015-10-03].

[KWF06] Sailesh Krishnamurthy, Chung Wu, and Michael Franklin. On-the-fly sharing for streamed aggregation. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 623–634, New York, NY, USA, 2006. ACM.

[Lan01] Douglas Laney. 3D Data Management: Controlling Data Volume, Velocity, and Variety. Technical report, META Group, February 2001.

[LCT+06] Hua-Gang Li, Songting Chen, Junichi Tatemura, Divyakant Agrawal, K. Selçuk Candan, and Wang-Pin Hsiung. Safety Guarantee of Continuous Join Queries over Punctuated Data Streams. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, VLDB '06, pages 19–30. VLDB Endowment, 2006.

[LDR08] Ralph Lange, Frank Dürr, and Kurt Rothermel. Scalable Processing of Trajectory-Based Queries in Space-Partitioned Moving Objects Databases. In *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '08, pages 270–279, Irvine, CA, USA, November 2008. ACM.

[Lil15] David Lillethun. *ssIoTa: A system software framework for the internet of things*. PhD thesis, Georgia Institute of Technology, 2015.

[LJ05] Guoli Li and Hans-Arno Jacobsen. Composite Subscriptions in Content-based Publish/Subscribe Systems. In *Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware*, Middleware '05, pages 249–269, New York, NY, USA, 2005. Springer-Verlag New York, Inc.

[LLS08] Geetika T. Lakshmanan, Ying Li, and Rob Strom. Placement Strategies for Internet-Scale Data Stream Systems. *IEEE Internet Computing*, 12(6):50–60, November 2008.

[LNR02] Alexander Leonhardi, Christian Nicu, and Kurt Rothermel. A Map-based Dead-reckoning Protocol for Updating Location Information. In *Proceedings of the International Parallel and Distributed Processing Symposium*, IPDPS '02, pages 193–200, 2002.

[LOBC12] Annie Liu, Michael Olson, Julian Bunn, and K. Mani Chandy. Towards a Discipline of Geospatial Distributed Event Based Systems. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, pages 95–106, New York, NY, USA, 2012. ACM.

[LR01a] Alexander Leonhardi and Kurt Rothermel. A Comparison of Protocols for Updating Location Information. *Cluster Computing*, 4(4):355–367, 2001.

[LR01b] Alexander Leonhardi and Kurt Rothermel. Architecture of a Large-scale Location Service. Technical report, Universitaetsbibliothek der Universitaet Stuttgart, Holzgartenstr. 16, 70174 Stuttgart, 2001.

[LRM12] Carlos Lübbe, Anja Reuter, and Bernhard Mitschang. Elastic Load-Balancing in a Distributed Spatial Cache Overlay. In *Proceedings of the IEEE 13th International Conference on Mobile Data Management*, MDM'12, pages 11–20, July 2012.

[LTC14] Yusen Li, Xueyan Tang, and Wentong Cai. On Dynamic Bin Packing for Resource Allocation in the Cloud. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '14, pages 2–11, New York, NY, USA, 2014. ACM.

[Luc01] David C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[Lun13] Ingrid Lunden. Mobile Twitter: 164M+ (75%) Access From Handheld Devices Monthly, 65% Of Ad Sales Come From Mobile. http://techcrunch.com/2013/10/03/mobile-twitter-161m-access-from-handheld-devices-each-month-65-of-ad-revenues-coming-from-mobile/, October 2013. [online; accessed 2014-10-03].

[LWG+09] Ralph Lange, Harald Weinschrott, Lars Geiger, André Blessing, Frank Dürr, Kurt Rothermel, and Hinrich Schütze. On a Generic Uncertainty Model for Position Information. In Kurt Rothermel, Dieter Fritsch, Wolfgang Blochinger, and Frank Dürr, editors, *QuaCon*, volume 5786 of *Lecture Notes in Computer Science*, pages 76–87. Springer, 2009.

[Mac67] James B. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. In L. M. Le Cam and J. Neyman, editors, *Proceedings of*

the 5th Berkeley Symposium on Mathematical Statistics and Probability, volume 1, pages 281–297. University of California Press, 1967.

[MBESG10] Ken Moody, Jean Bacon, David Evans, and Scarlet Schwiderski-Grosche. Implementing a Practical Spatio-Temporal Composite Event Language. In Kai Sachs, Ilia Petrov, and Pablo Guerrero, editors, *From Active Data Management to Event-Based Systems and More*, volume 6462 of *Lecture Notes in Computer Science*, pages 108–123. Springer Berlin Heidelberg, 2010.

[MC08] Anurag S. Maskey and Mitch Cherniack. Replay-Based Approaches to Revision Processing in Stream Query Engines. In *Proceedings of the 2nd International Workshop on Scalable Stream Processing System*, SSPS '08, pages 3–12, New York, NY, USA, 2008. ACM.

[MKR14] Ruben Mayer, Boris Koldehofe, and Kurt Rothermel. Meeting Predictable Buffer Limits in the Parallel Execution of Event Processing Operators. In *Proceedings of the 2014 IEEE International Conference on Big Data*, BigData '14, pages 1–10. IEEE, October 2014.

[MKR15] Ruben Mayer, Boris Koldehofe, and Kurt Rothermel. Predictable Low-Latency Event Detection with Parallel Complex Event Processing. *Internet of Things Journal, IEEE*, 2(4):274–286, 2015.

[MMI+13] Derek G. Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. Naiad: A Timely Dataflow System. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles*, SOSP '13, pages 439–455, New York, NY, USA, 2013. ACM.

[MP13] Christopher Mutschler and Michael Philippsen. Reliable Speculative Processing of Out-of-order Event Streams in Generic Publish/Subscribe Middlewares. In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS'13, pages 147–158, New York, NY, USA, 2013. ACM.

[MPGJ05] Vinod Muthusamy, Milenko Petrovic, Dapeng Gao, and Hans-Arno Jacobsen. Publisher mobility in distributed publish/subscribe systems. In *Proceedings of the Fourth International Workshop on Distributed Event-Based Systems (DEBS) (ICDCSW'05) - Volume 04*, pages 421–427, Washington, DC, USA, 2005. IEEE Computer Society.

[MPVW05] Ulrich Meissen, Stefan Pfennigschmidt, Agnès Voisard, and Tjark Wahnfried. Resolving Knowledge Discrepancies in Situation-aware Systems. *International Journal of Pervasive Computing and Communication*, 1(4):327–336, 2005.

[MS05] Anand Meka and Ambuj Singh. DIST: A Distributed Spatiotemporal Index Structure for Sensor Networks. In *Proceedings of the 14th ACM international*

Conference on Information and Knowledge Management, CIKM '05, pages 139–146, New York, NY, USA, 2005. ACM.

[MSHR02] Samuel Madden, Mehul Shah, Joseph M. Hellerstein, and Vijayshankar Raman. Continuously Adaptive Continuous Queries over Streams. In *Proceedings of the 2002 ACM SIGMOD international Conference on Management of Data*, SIGMOD '02, pages 49–60, New York, NY, USA, 2002. ACM.

[MUH04] Gero Mühl, Andreas Ulbrich, and Klaus Herrman. Disseminating Information to Mobile Clients Using Publish–Subscribe. *Internet Computing, IEEE*, 8(3):46–53, June 2004.

[MW12] Ricky K. K. Ma and Cho-Li Wang. Lightweight Application-Level Task Migration for Mobile Cloud Computing. In *Proceedings of the 26th IEEE International Conference on Advanced Information Networking and Applications*, AINA '12, pages 550–557. IEEE Computer Society, 2012.

[NRB09] Rimma V. Nehme, Elke A. Rundensteiner, and Elisa Bertino. Tagging Stream Data for Rich Real-time Services. *Proceedings of the VLDB Endowment*, 2(1):73–84, August 2009.

[NRNK10] Leonardo Neumeyer, Bruce Robbins, Anish Nair, and Anand Kesari. S4: Distributed Stream Computing Platform. In *Proceedings of the 2010 IEEE International Conference on Data Mining Workshops*, ICDMW '10, pages 170–177, Washington, DC, USA, 2010. IEEE Computer Society.

[OKR+14a] Beate Ottenwälder, Boris Koldehofe, Kurt Rothermel, Kirak Hong, David Lillethun, and Umakishore Ramachandran. MCEP: A Mobility-Aware Complex Event Processing System. *ACM Transactions on Internet Technology*, 14(1):6:1–6:24, August 2014.

[OKR+14b] Beate Ottenwälder, Boris Koldehofe, Kurt Rothermel, Kirak Hong, and Umakishore Ramachandran. RECEP: Selection-based Reuse for Distributed Complex Event Processing. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, DEBS '14, pages 59–70, New York, NY, USA, 2014. ACM.

[OKRR13] Beate Ottenwälder, Boris Koldehofe, Kurt Rothermel, and Umakishore Ramachandran. 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, pages 183–194, New York, NY, USA, 2013. ACM.

[OMK14] Beate Ottenwälder, Ruben Mayer, and Boris Koldehofe. Distributed Complex Event Processing for Mobile Large-scale Video Applications. In *Proceedings*

of Middleware'14: Posters & Demos Session, Middleware Posters and Demos '14, pages 5–6, New York, NY, USA, 2014. ACM.

[Ora15] Oracle. Oracle stream explorer. http://www.oracle.com/technetwork/middleware/co event-processing/overview/complex-event-processing-088095.html, 2015. [online; accessed 2015-10-03].

[Pac09] Hewlett Packard. Shell to use CeNSE for clearer picture of oil and gas reservoirs. http://www.hpl.hp.com/news/2009/oct-dec/cense.html, 2009. [online; accessed 2015-10-03].

[PJT00] Dieter Pfoser, Christian S. Jensen, and Yannis Theodoridis. Novel Approaches in Query Processing for Moving Object Trajectories. In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, pages 395–406, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[PLS+06] Peter Pietzuch, Jonathan Ledlie, Jeffrey Shneidman, Mema Roussopoulos, Matt Welsh, and Margo Seltzer. Network-Aware Operator Placement for Stream-Processing Systems. In *Proceedings of the 22nd International Conference on Data Engineering*, ICDE '06, pages 49–60. IEEE Computer Society, 2006.

[PMS+09] Padmanabhan S. Pillai, Lily B. Mummert, Steven W. Schlosser, Rahul Sukthankar, and Casey J. Helfrich. 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*, NOSSDAV '09, pages 43–48, New York, NY, USA, 2009. ACM.

[PS11] Kostas Patroumpas and Timos Sellis. Maintaining Consistent Results of Continuous Queries under Diverse Window Specifications. *Information Systems*, 36(1):42–61, March 2011.

[PSA+13] Damian Philipp, Jaroslaw Stachowiak, Patrick Alt, Frank Dürr, and Kurt Rothermel. DrOPS: Model-driven optimization for Public Sensing systems. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications*, PerCom'13, pages 185–192, March 2013.

[PSB04] Peter Pietzuch, Brian Shand, and Jean Bacon. Composite Event Detection as a Generic Middleware Extension. *IEEE Network*, 18(1):44–55, February 2004.

[RDR10] Stamatia Rizou, Frank Dürr, and Kurt Rothermel. Solving the Multi-operator Placement Problem in Large-Scale Operator Networks. In *Proceedings of 19th International Conference on Computer Communication Networks*, ICCCN '10, pages 1–6. IEEE Communications Society, Aug. 2010.

[RDR11] Stamatia Rizou, Frank Durr, and Kurt Rothermel. Fulfilling end-to-end latency constraints in large-scale streaming environments. In *Proceedings of the*

IEEE 30th International Performance Computing and Communications Conference, IPCCC '11, pages 1–8, Nov 2011.

[RHI+12] Umakishore Ramachandran, Kirak Hong, Liviu Iftode, Ramesh Jain, Rajnish Kumar, Kurt Rothermel, Junsuk Shin, and Raghupathy Sivakumar. Large-scale situation awareness with camera networks and multimodal sensing. *Proceedings of the IEEE*, 100(4):878–892, April 2012.

[Riz13] Stamatia Rizou. *Concepts and algorithms for efficient distributed processing of data streams*. PhD thesis, Universität Stuttgart, Holzgartenstr. 16, 70174 Stuttgart, 2013.

[RSSB00] Prasan Roy, S. Seshadri, S. Sudarshan, and Siddhesh Bhobe. Efficient and extensible algorithms for multi query optimization. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 249–260, New York, NY, USA, 2000. ACM.

[RSW+07] Frederick Reiss, Kurt Stockinger, Kesheng Wu, Arie Shoshani, and Joseph M. Hellerstein. Enabling Real-Time Querying of Live and Historical Stream Data. In *Proceedings of the 19th International Conference on Scientific and Statistical Database Management*, SSDBM '07, pages 28–38, Washington, DC, USA, 2007. IEEE Computer Society.

[RvdM14] Janessa Rivera and Rob van der Meulen. Gartner Says Annual Smartphone Sales Surpassed Sales of Feature Phones for the First Time in 2013. http://www.gartner.com/newsroom/id/2665715, 2014. [online; accessed 2014-07-15].

[RZRD09] Venkatesh Raghavan, Yali Zhu, Elke A. Rundensteiner, and Daniel Dougherty. Multi-Join Continuous Query Optimization: Covering the Spectrum of Linear, Acyclic, and Cyclic Queries. In Alan P. Sexton, editor, *Dataspace: The Final Frontier*, volume 5588 of *Lecture Notes in Computer Science*, pages 91–106. Springer Berlin Heidelberg, 2009.

[SA11] Nenad Stojanovic and Alexander Artikis. On Complex Event Processing for Real-time Situational Awareness. In *Proceedings of the 5th International Conference on Rule-based Reasoning, Programming, and Applications*, RuleML'2011, pages 114–121, Berlin, Heidelberg, 2011. Springer-Verlag.

[SBCD09] Mahadev Satyanarayanan, Paramvir Bahl, Ramon Caceres, and Nigel Davies. The Case for VM-Based Cloudlets in Mobile Computing. *Pervasive Computing, IEEE*, 8(4):14–23, 2009.

[Sch03] Jochen Schiller. *Mobilkommunikation*. Pearson Studium. Pearson Studium, 2003.

[SESFT11] Kyumars Sheykh Esmaili, Tahmineh Sanamrad, Peter M. Fischer, and Nesime Tatbul. Changing Flights in Mid-air: A Model for Safely Modifying Continuous Queries. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 613–624, New York, NY, USA, 2011. ACM.

[SGM09] Scarlet Schwiderski-Grosche and Ken Moody. The SpaTeC composite event language for spatio-temporal reasoning in mobile systems. In *Proceedings of 3rd ACM International Conference on Distributed Event-Based Systems*, DEBS '09, pages 1–12, New York, NY, USA, 2009. ACM.

[SHCF03] Mehul A. Shah, Joseph M. Hellerstein, Sirish Chandrasekaran, and Michael J. Franklin. Flux: An Adaptive Partitioning Operator for Continuous Query Systems. In *Proceedings of the 19th International Conference on Data Engineering*, ICDE '03, pages 25–36, March 2003.

[SKPR10] Björn Schilling, Boris Koldehofe, Udo Pletat, and Kurt Rothermel. Distributed Heterogeneous Event Processing: Enhancing Scalability and Interoperability of CEP in an Industrial Context. In *Proceedings of the 4th ACM International Conference on Distributed Event-Based Systems*, DEBS '10, pages 150–159, New York, NY, USA, 2010. ACM.

[SKR11] Björn Schilling, Boris Koldehofe, and Kurt Rothermel. Efficient and Distributed Rule Placement in Heavy Constraint-Driven Event Systems. In *Proceedings of the 13th IEEE International Conference on High Performance Computing and Communications*, HPCC '11, pages 355–364, September 2011.

[SKRR13] Björn Schilling, Boris Koldehofe, Kurt Rothermel, and Umakishore Ramachandran. Access Policy Consolidation for Complex Event Processing. In *IEEE Conference on Networked Systems*, NetSys '13, pages 92–101. IEEE, March 2013.

[SM11] Zoe Sebepou and Kostas Magoutis. CEC: Continuous Eventual Checkpointing for Data Stream Processing Operators. In *Proceedings of the 41st International Conference on Dependable Systems Networks*, DSN '11, pages 145–156, June 2011.

[Smi14] Craig Smith. By the numbers: 50 amazing youtube statistics. http://expandedramblings.com/index.php/youtube-statistics/, April 2014. [online; accessed 2014-07-15].

[SMK12] Stephan Schnitzer, Hugo Miranda, and Boris Koldehofe. Content routing algorithms to support Publish/Subscribe in Mobile Ad Hoc Networks. In *Proceedings of the IEEE 37th Conference on Local Computer Networks Workshops*, LCN Workshops '12, pages 1053–1060, October 2012.

[SPTL04] Jimeng Sun, D. Papadias, Yufei Tao, and Bin Liu. 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, pages 202–213, 2004.

[Sto14] Storm. Companies Using Apache Storm. https://storm.incubator.apache.org/documentation/Powered-By.html, 2014. [online; accessed 2015-10-03].

[SXSS14] Nenad Stojanovic, Yongchun Xu, Aleksandar Stojadinovic, and Ljiljana Stojanovic. Using Mobile-based Complex Event Processing to Realize Collaborative Remote Person Monitoring. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, DEBS'14, pages 225–235, New York, NY, USA, 2014. ACM.

[TcZ07] Nesime Tatbul, Uğur Çetintemel, and Stan Zdonik. Staying FIT: Efficient Load Shedding Techniques for Distributed Stream Processing. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 159–170. VLDB Endowment, 2007.

[TDSC07] Goce Trajcevski, Hui Ding, Peter Scheuermann, and Isabel F. Cruz. BORA: Routing and Aggregation for Distributed Processing of Spatio-Temporal Range Queries. In *Proceedings of the 2007 International Conference on Mobile Data Management*, MDM '07, pages 36–43, May 2007.

[TKK+10] Muhammad Adnan Tariq, Gerald G. Koch, Boris Koldehofe, Imran Khan, and Kurt Rothermel. 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*, EuroPar '10, pages 458–470, Berlin, Heidelberg, 2010. Springer-Verlag.

[TKK+11] Muhammad Adnan Tariq, Boris Koldehofe, Gerald G. Koch, Imran Khan, and Kurt Rothermel. Meeting subscriber-defined QoS constraints in publish/subscribe systems. *Concurrency and Computation: Practice and Experience*, 23(17):2140–2153, 2011.

[TMSF03] Peter A. Tucker, David Maier, Tim Sheard, and Leonidas Fegaras. Exploiting Punctuation Semantics in Continuous Data Streams. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):555–568, March 2003.

[TS04] Goce Trajcevski and Peter Scheuermann. Reactive Maintenance of Continuous Queries. *ACM SIGMOBILE Mobile Computing and Communications Review*, 8(3):20–31, July 2004.

[TSBV05] Goce Trajcevski, Peter Scheuermann, Hervé Brönnimann, and Agnès Voisard. Dynamic topological predicates and notifications in moving objects databases.

In *Proceedings of the 6th International Conference on Mobile Data Management*, MDM '05, pages 77–85, New York, NY, USA, 2005. ACM.

[Twi14a] http://twitter.com/. online, 2014. [online; accessed 2014-07-30].

[Twi14b] Twitter. New Tweets per second record, and how! https://blog.twitter.com/2013/new-tweets-per-second-record-and-how, 2014. [online; accessed 2014-07-15].

[UMJ+14] Jacopo Urbani, Alessandro Margara, Ceriel Jacobs, Spyros Voulgaris, and Henri Bal. AJIRA: A Lightweight Distributed Middleware for MapReduce and Stream Processing. In *Proceedings of the IEEE 34th International Conference on Distributed Computing Systems*, ICDCS '14, pages 545–554, June 2014.

[Var01] András Varga. The OMNeT++ Discrete Event Simulation System. In *Proceedings of the European Simulation Multiconference*, ESM '01, pages 1–7, June 2001.

[vdMR14] Rob van der Meulen and Janessa Rivera. Gartner Says Worldwide Traditional PC, Tablet, Ultramobile and Mobile Phone Shipments On Pace to Grow 7.6 Percent in 2014. http://www.gartner.com/newsroom/id/2645115, 2014. [online; accessed 2014-10-21].

[Ver12] http://www.b30-oberschwaben.de/html/vergleiche.html, February 2012.

[Vet12] Marcus Vetter. Ereigniskorrelation auf energiebeschränkten mobilen Endgeräten. Bachelorarbeit: Universität Stuttgart, Institut für Parallele und Verteilte Systeme, Verteilte Systeme, October 2012.

[VH08] András Varga and Rudolf Hornig. 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, pages 1–10, Brussels, Belgium, Belgium, 2008. ICST.

[VKR11] Marco Völz, Boris Koldehofe, and Kurt Rothermel. Supporting Strong Reliability for Distributed Complex Event Processing Systems. In *Proceedings of the IEEE 13th International Conference on High Performance Computing and Communications*, HPCC'11, pages 477–486, September 2011.

[VSDTD13] Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt. Leveraging Cloudlets for Immersive Collaborative Applications. *Pervasive Computing, IEEE*, 12(4):30–38, 2013.

[WDR06] Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-Performance Complex Event Processing over Streams. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 407–418, New York, NY, USA, 2006. ACM.

[WJFR10] Patrick Wendell, Joe Wenjie Jiang, Michael J. Freedman, and Jennifer Rexford. DONAR: Decentralized Server Selection for Cloud Services. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 231–242, New York, NY, USA, 2010. ACM.

[WRE11] Di Wang, Elke A. Rundensteiner, and Richard T. Ellison, III. Active Complex Event Processing over Event Streams. *Proceedings of the VLDB Endowment*, 4(10):634–645, July 2011.

[WSCY99] Ouri Wolfson, A. Prasad Sistla, Sam Chamberlain, and Yelena Yesha. Updating and Querying Databases that Track Mobile Units. *Distributed and Parallel Databases*, 7:257–387, July 1999.

[WZC+12] Zhaoran Wang, Yu Zhang, Xiaotao Chang, Xiang Mi, Yu Wang, Kun Wang, and Huazhong Yang. 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*, DEBS '12, pages 127–138, New York, NY, USA, 2012. ACM.

[XECA07] Xiaopeng Xiong, H.G. Elmongui, Xiaoyong Chai, and W.G. Aref. PLACE*: A Distributed Spatio-temporal Data Stream Management System for Moving Objects. In *Proceedings of the 2007 International Conference on Mobile Data Management*, MDM '07, pages 44–51, May 2007.

[XJ07] Zhengdao Xu and Arno Jacobsen. Adaptive Location Constraint Processing. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 581–592, New York, NY, USA, 2007. ACM.

[XJ09] Zhengdao Xu and Hans-Arno Jacobsen. Expressive Location-Based Continuous Query Evaluation with Binary Decision Diagrams. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ICDE '09, pages 1155–1158, Washington, DC, USA, 2009. IEEE Computer Society.

[XLT06] Shili Xiang, Hock Beng Lim, and Kian-Lee Tan. Impact of Multi-query Optimization in Sensor Networks. In *Proceedings of the 3rd Workshop on Data Management for Sensor Networks: In Conjunction with VLDB 2006*, DMSN '06, pages 7–12, New York, NY, USA, 2006. ACM.

[XMA05] Xiaopeng Xiong, Mohamed F. Mokbel, and Walid G. Aref. SEA-CNN: Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-temporal Databases. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 643–654, 2005.

[XP115] Sony xperia z4. http://www.gsmarena.com/sony_xperia_z4_compact-6957.php, 2015. [online; accessed 2015-08-31].

[YKPS07] Yin Yang, Jürgen Krämer, Dimitris Papadias, and Bernhard Seeger. HybMig: A Hybrid Approach to Dynamic Plan Migration for Continuous Queries. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):398–411, March 2007.

[YLTX08] Lei Ying, Zhen Liu, Don Towsley, and Cathy H. Xia. Distributed Operator Placement and Data Caching in Large-Scale Sensor Networks. In *Proc. of 27th IEEE International Conference on Computer Communications*, INFOCOM'08, pages 977–985, April 2008.

[YRW09] Di Yang, Elke A. Rundensteiner, and Matthew O. Ward. A Shared Execution Strategy for Multiple Pattern Mining Requests over Streaming Data. *Proceedings of the VLDB Endowment*, 2(1):874–885, August 2009.

[ZDI14] Haopeng Zhang, Yanlei Diao, and Neil Immerman. On Complexity and Optimization of Expensive Queries in Complex Event Processing. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 217–228, New York, NY, USA, 2014. ACM.

[ZHBM07] Yang Zhang, Bret Hull, Hari Balakrishnan, and Samuel Madden. ICEDB: Intermittently-Connected Continuous Query Processing. In *Proceedings of the International Conference on Data Engineering*, ICDE '07, pages 166–175, Istanbul, Turkey, April 2007.

[ZJDR10] Rui Zhang, H. V. Jagadish, Bing Tian Dai, and Kotagiri Ramamohanarao. Optimized Algorithms for Predictive Range and KNN Queries on Moving Objects. *Information Systems*, 35(8):911–932, December 2010.

[ZOTW06] Yongluan Zhou, Beng Chin Ooi, Kian-Lee Tan, and Ji Wu. Efficient dynamic operator placement in a locally distributed continuous query system. In *Proceedings of the Confederated International Conference on On the Move to Meaningful Internet Systems: CoopIS, DOA, GADA, and ODBASE - Volume Part I*, ODBASE'06/OTM'06, pages 54–71, Berlin, Heidelberg, 2006. Springer-Verlag.

[ZPG+10] Xiaolan J. Zhang, Sujay Parekh, Bugra Gedik, Henrique Andrade, and Kun-Lung Wu. Workload Characterization for Operator-based Distributed Stream Processing Applications. In *Proceedings of the 4th ACM International Conference on Distributed Event-Based Systems*, DEBS '10, pages 235–247, New York, NY, USA, 2010. ACM.

[ZTH08] Yun Zhai, Ying-Li Tian, and Arun Hampapur. Composite Spatio-Temporal Event Detection in Multi-Camera Surveillance Networks. In *Proceedings of the Workshop on Multi-camera and Multi-modal Sensor Fusion Algorithms and Applications*, M2SFA2 '08, pages 1–12, Marseille France, 2008.

[ZZL07] Jianjun Zhang, Gong Zhang, and Ling Liu. GeoGrid: A Scalable Location Service Network. In *Proceedings of the 27th International Conference on Distributed Computing Systems*, ICDCS '07, pages 60–68, June 2007.