RECEP: Selection-based Reuse for Distributed Complex Event Processing

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

Kirak Hong, Umakishore Ramachandran
College of Computing
Georgia Institute of Technology,
Atlanta, GA, USA
hokira@gatech.edu,rama@cc.gatech.edu

ABSTRACT

An appealing use case of complex event processing (CEP) systems is for mobile users to react in real-time to events in their environment, e.g., to the occurrence of a dangerous situation such as an accident. Maintaining mobile CEP systems is highly resource intensive since in many cases events need to be detected in a consumer-centric manner to ensure low latency event detection and high quality of results. In this paper we propose the RECEP system to increase the scalability of mobile CEP systems. In the presence of mobile users with partially overlapping interest, the RECEP system offers methods to efficiently reuse computations and this way reduces the resource requirements of mobile CEP. Since reuse of computations happens with respect to well defined quality metrics, RECEP can be easily tailored to specific mobile applications and maximize the resource savings for their desired quality in terms of precision and recall of the processed events from the user's environment.

Categories and Subject Descriptors

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

Keywords

mobility; complex event processing; query optimization

1. INTRODUCTION

With the ubiquitous deployment of sensors and mobile devices, highly popular applications are emerging that depend on live information collected by these devices. For example, a smart navigation system [18] uses location and speed information of other vehicles on its consumer's path to provide real-time traffic information. Similarly, a friend finder application can process video streams of camera networks to find friends close to a consumer. The common goal of these applications, denoted *situation awareness applications*, is to automatically process live streams from relevant devices and provide meaningful *situational information* to consumers.

Complex Event Processing (CEP) systems [22, 16] offer tremendous support for large-scale situation awareness applications by detecting meaningful events—often referred to as complex event—from low level sensor or event streams. Consumers can register a continuous query with the system that describes the situational information of interest. The CEP system will notify the consumers in return about any detected event that matches the consumer's query. The logic to detect events is provided by the CEP system as computing modules, called operators, which are executed on distributed computing resources to process live streams.

By continuously adapting the placement of operators with respect to a consumer's location, previous work [25] has shown that low latency detection of events can be achieved. Especially, arising computing paradigms like *fog computing* [5] or *cloudlets* [29] allow the operators to be placed flexibly in a mobile infrastructure that utilizes computing resources at the edge and in the core of the network. This way CEP systems can establish low latency paths between mobile consumers and producers. However, supporting a large number of consumers in a dynamic environment is highly challenging. For each mobile consumer an individual set of operators needs to be maintained and adapted to ensure low latency paths. This requires a significant amount of resources. Besides imposing a significant cost for a large-scale deployment, resources that can be utilized at the edge of the network will certainly be constraint in their number. Therefore, a primary concern for the development of mobile CEP systems is to increase the resource-efficiency in performing consumer-centric CEP.

To address this problem, we propose the Reuse-Aware CEP (RECEP) system, a scalable CEP system that reduces computational and communication load by reusing computations and streams between operators. In particular, our system exploits two inherent characteristics of situation awareness 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 an average of 70 cars per hour pass the same mile on an interstate [1] and, according to recent surveys, up to 80% [2] of consumers can have the same interest in information from a connected car. When there are more consumers in a certain area, there will also be more overlaps between the consumers, since their

DEBS'14, May 26–29, 2014, MUMBAI, India.

http://dx.doi.org/10.1145/2611286.2611297.

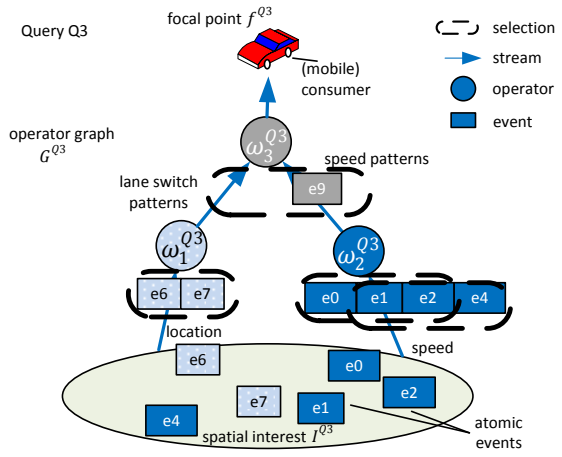


Figure 1: Example of operator graph: The most recent, temporally ordered, atomic events e_0 through e_7 are streamed and processed by G^{Q3}.

locations are similar. Another important characteristic is that most situation awareness applications do not require a perfectly accurate set of input data to generate meaningful situational information, since the location of the consumer is often uncertain [21]. 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. 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.

In this paper, we make the following contributions. First of all, we provide system mechanisms for fine-grained reuse of computations and streams between operators to support scalable processing of consumer queries. Secondly, we present methods to control the quality of the approximate results and the associated overhead of the reuse mechanism. Lastly, we provide evaluation results that show the benefits of the proposed reuse approach in terms of system scalability and resource utilization.

The rest of the paper is structured as follows: We present the CEP and system model in Section 2. The problem is discussed in Section 3. Our fine-grained reuse mechanism is detailed in Section 4. In Section 5 we then evaluate our system before we present the related work in Section 6. We conclude in Section 7 with a short summary and outlook.

2. CEP AND SYSTEM MODEL

In this section we introduce the necessary premises for our work, namely i) the query and operator graph model, ii) assumptions about the operator execution, and iii) assumptions we make for the execution of mobile CEP operators.

2.1 Operator graph and Query model

To describe situational information we use an *operator graph* model [18, 17] which comprises *operators* (vertices) that perform the processing of events and the *event streams* established between pairs of operators (edges). For instance in Figure 1, three operators ω_1, ω_2, and ω_3 are connected to an operator graph. In our work the operator graph is adapted to the changing interest of each individual consumer, i.e., the operator graph always receives *atomic event streams* (i.e., sensor data) from sensors close to the location of a consumer [18]. Therefore, the RECEP system maintains for each query of a consumer a distinct operator graph. Each query Q comprises the operator graph G, an *interest* in a spatial region I, and the *focal point* of a consumer f. The spatial region I defines the region from which atomic events are selected as input to the operator graph. Updates to the focal point f, e.g., due to the mobility of the consumer, trigger updates to the spatial interest I. To match atomic events against the spatial interest and reason about temporal causalities, they are stamped with a type ($type$), location (l), and timestamp (ts). The query in the example of Figure 1 selects changes in speed and locations of cars as atomic events with respect to its spatial interest I^{Q3} and uses them in operator graph G^{Q3} to detect obstacles or the occurrence of an accident around the consumer. When the focal point f^{Q3}, given by the mobile consumer's location, changes, the processing of an old spatial region is stopped, the spatial interest is updated, and the operator graph is restarted on the updated spatial interest.

2.2 Operator Execution

The execution of an operator is characterized by performing a sequence of correlation steps. In each correlation step the operator takes as input a *selection* of events from its incoming event stream and applies its operator specific correlation function—which implements the operator logic—to produce a set of outgoing events. The selection on the incoming stream is regularly updated by means of a *selection policy*, i.e., windows that shift on the incoming streams and determine sub-streams. In previous work [18] we already discussed that such a model is highly expressive in terms of temporal correlations. Yet note that this particular window model needs to be extended to efficiently support unbounded operations like Kleene closure. For example, ω_2^{Q3} in Figure 1 has specified a selection policy in form of a sliding window of size 3 to select three subsequent speed events $\{e_0, e_1, e_2\}$ on its incoming stream. In the next correlation step the operator will apply its correlation function to the selection $\{e_1, e_2, e_4\}$. To support selection policies like counting windows or time-based windows [8] we assume the input streams and outgoing streams are temporally ordered by means of a timestamping mechanism. Any event that has been produced with respect to a selection s of events, carries the largest timestamp comprised in s. Furthermore, each event is annotated with a sequence number to resolve ambiguities that can stem from two events carrying the same timestamp. At times, we also need to refer to the start and end of a selection s, where $t_s(s)$ is the smallest and $t_e(s)$ is the largest timestamp comprised in a selection. Any event comprised in a selection s of an operator is caused by and therefore *depends on* the processing of atomic events in the operators sub-graphs. We denote this set of atomic events as $a(s)$.

Observe, although the model of executing selections performs 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.

2.3 Infrastructure

The RECEP system is deployed on a distributed infrastructure consisting of a set of Virtual Machines (VMs).The virtual machines in RECEP are primarily used to execute one or multiple operators. This allows the CEP system to

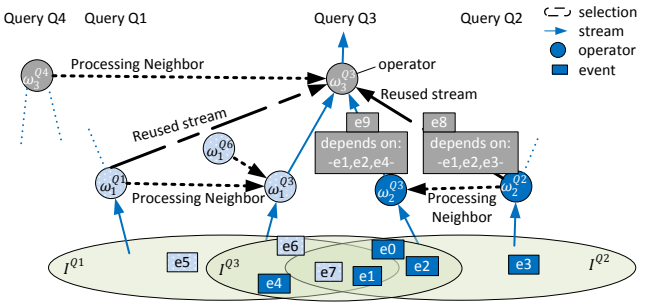


Figure 2: Example for reuse-aware operator graph: ω_2^{Q2} processes on behalf of its processing neighbor ω_2^{Q3}. The event e_8 is delivered to ω_3^{Q3} instead of e_9.

dynamically adapt the number of required computing resources to the current workload without the need for the system's provider to maintain a large-scale infrastructure. The virtual machines are requested on demand and follow the Infrastructure as a Service model. Similar to cloudlets and the fog model, virtual machines can be requested on the path between a consumer and a producer to ensure low latency, in-network processing of events. We assume the capabilities of each virtual machine is constraint in terms of compute cycles, memory, and bandwidth.

3. PROBLEM DESCRIPTION

In this paper 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. Note, that 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 operators and the same dependencies to their predecessors and successors.[1] Since processing is performed with respect to a consumer-centric spatial interest, two mobile consumers can only share the result of an operator's correlation step if the spatial interest of their queries is overlapping. In particular, the result of an operator can only be reused by another operator if both operators intend to perform a processing step with respect to matching selections, i.e., both selections comprise the same set of events. Hence, the execution of the RECEP system can be characterized by a reuse-aware operator graph (see Figure 1) in which some operators perform correlation steps on behalf of other operators. For an operator ω multiple operators—named the *processing neighbors*—may exist on whose behalf ω can process and stream produced events.

Due to the dynamically changing partial overlap in the spatial interests of mobile consumers it becomes highly challenging to find matching selections between pairs of operators without significantly interrupting the execution of the CEP system and this way increase the time until events can be detected. As part of this paper, we approach this problem by providing mechanisms to relax the quality at which the CEP system detects events. 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. 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 set of atomic events $a(s)$ and $a(s')$ on which s and s' depend. This allows us to apply two commonly used metrics, called *precision* and *recall*: The precision defines how noisy the input of a selection is. The recall defines how relevant the input of a selection is.

$$precision(s, s') = \frac{a(s) \bigcap a(s')}{a(s')} \quad (1)$$

$$recall(s, s') = \frac{a(s) \bigcap a(s')}{a(s)} \quad (2)$$

Since an input of an operator depends on multiple predecessors in the reuse-aware operator graph and each event comprised in a selection was detected reusing results of a distinct set of processing neighbors, we also need to measure the *disparity* in spatial regions on which events in a selection depend on. In particular, let I_u denote the set of all spatial interests utilized by operators of G in detecting events of a selection. Then we measure disparity by determining for each atomic event $e \in a(s)$ how many $r \in I_u$ match the corresponding location l over all possible combinations of atomic events and spatial interests, i.e.,

$$disparity(s) = \frac{\sum_{e \in a(s)} |\{r | (r \in I_u) \wedge (l(e) \in r)\}|}{|a(s)||I_u|} \quad (3)$$

A good disparity of 1 means that all spatial interests are comprising all relevant atomic events while in the example of Figure 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}, and the spatial interests of $Q1$ and $Q2$ hardly overlap.

Our goal is therefore to maximize the reuse 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, recall, and disparity (denoted *Quality of Result (QoR)*) should each be preserved above an individual threshold T_p^Q, T_r^Q, T_d^Q (short QoR_{th}^Q) for each query Q in the system.

4. REUSE-AWARE PROCESSING

Before describing our approach to reuse-aware event processing, we briefly introduce and summarize the main components that make up the RECEP system.

Upon registering a query Q for a mobile consumer, Q is directed to a component called RECEP controller to bootstrap the deployment of the query. The RECEP controller is in charge of finding an initial placement for the operators of

[1]Consider that many cars are equipped with navigational systems that may display live-information about traffic jams or blocked roads. Corresponding situational information can be detected using an operator to calculate the average speed in an area in addition to other, disjoint operators.

G on the set of VMs. To this end the controller can reserve new VMs or remove VMs similar to [14].

Each VM of the RECEP system runs a reuse-aware execution environment which controls the streaming and the processing of the operators. Moreover, the execution environment also provides mechanisms to automatically adapt the placement of an operator and provides mechanisms for the live migration of operators from one VM to another. Besides ensuring low latency, the placement mechanism of the RECEP environment will ensure that operators of the same type and similar location will be hosted at the same VM. In this work we build on already proposed migration and placement mechanisms [25], so we omit the details.

However, what distinguishes RECEP from traditional distributed or mobile CEP systems is the presence of a component we call *selection manager* (see Figure 3). Each operator receives streams from and forwards streams to the selection manager. In return, the selection manager i) determines independently processable selections on the incoming stream (*selection phase*), ii) decides which selections need to be executed by which operator (*processing phase*), and iii) decides to which targets the results of a correlation step needs to be streamed to (*streaming phase*).

In the remainder of this section we present the details behind the concepts and implementation of the selection manager. In Section 4.1, we first detail the concepts behind the phases performed by the selection manager where a group of operators is already assigned to the selection manager. To increase the scalability of our approach, we illustrate in Section 4.2 how to perform a general grouping of operators. Since the mechanisms for managing selections crucially depend on the mechanisms for monitoring the quality of selections, we will detail the quality monitoring in Section 4.3.

4.1 Selection-based Processing

Recall, the main objective of the selection manager is to minimize the number of selections that need to be processed by the operators connected to the selection manager. Therefore, the selection manager analyzes the incoming streams and determines whether in a subsequent correlation step the selection of one operator can cover also some selections processed by other operators. In this case, the result of a correlation step with respect to such a selection—we call a *covering selection*—can be reused by all operators whose selections are covered. Each operator ω is also assigned a set of processing neighbors, i.e., a preselected set of operators connected to the selection manager for whom ω can provide covering selections. For now, we will detail the phases behind 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.

4.1.1 Selection Phase

In the selection phase incoming streams of all connected operators are analyzed 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 stream. Furthermore, the selection manager identifies with respect to the selection

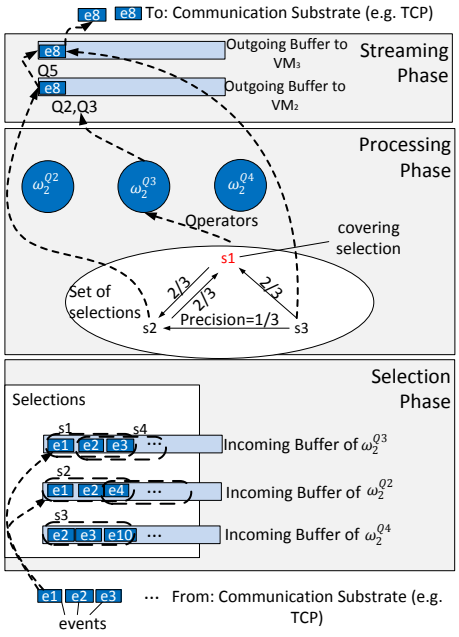


Figure 3: Since comprising the same events, s_1 through s_3 can cover each other. In the processing phase s_1 is selected as covering selection and ω_2^{Q3} processes on behalf of the other operators.

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. 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. Initially, every selection will be marked as a covering selection 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 will be used by the selection manager to identify covering selections.

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 QoR following the definition of precision and recall. How to provide such information efficiently is discussed in Section 4.3.1. For now, assume that each event is annotated with the complete provenance information [10], i.e., the provenance information comprises ids of all atomic events on which the event depends.

4.1.2 Processing Phase

The processing phase is periodically initiated and performed with respect to all selections for which processing is disabled. The goal of the processing phase is 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 prob-

```
1: function find_covering_selection(Set of Operator Ω)
2:   W ← ∅ //Set of covering selections
3:   F ← ∅ //Family of subsets
4:   for all s_ω ∈ S_Ω do //iterate over selections of oper. in Ω
5:     C ← {s_ω} //each selection covers itself
6:     for all ω' ∈ neighbor(ω) do
7:       for all s' ∈ S_ω' do
8:         if precision(s_ω', s_ω) > T_p^ω'
             ∧ recall(s_ω', s_ω) > T_r^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
```

Figure 4: Maximum QoR Coverage Heuristic

lem [7], 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 by X and a family of subsets of X denoted by F, the minimum number of subsets in F covering all elements in X. In our setting, an element of 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 quality.

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 [15], which is known to give a logarithmic ratio bound with respect to the maximum size of a subset. As alternative, we propose a heuristic to reduce the computational cost that stems from the generation of subsets. Intuitively, the heuristic prioritizes covering selections that stem from operators with many processing neighbors, since those selections potentially cover many selections.

Maximum QoR Coverage Heuristic (MQC). The MQC algorithm, which is detailed in Figure 4, takes as input a set of operators Ω. All operators in Ω are of the same type and are grouped ahead by the selection manager. *MQC* returns as a result W, comprising the set of covering selections as well as the selections covered by each covering selection. To this end *MQC* 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 *MQC* builds w.r.t. s_ω a subset as follows: *MQC* 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 quality threshold (Line 9). The subset is then added to F (Line 13).

After *MQC* 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 covering selection is removed (Line 20).

```
1: function find_covering_selection(Ω))
2:   P ← PriorityList(Ω) //Operators are sorted by priorities
3:   W ← ∅ //Set of covering selections
4:   while P ≠ ∅ do //cover all selections of all operators
5:     ω ← Extract-Max(P) //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, s_ω') ∈ W with ω' ∈ neighbor(ω) do
9:         if precision(s_ω, s_ω') > T_p^ω
             ∧ recall(s_ω, s_ω') > T_r^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
```

Figure 5: Maximum Neighbor Heuristic

Maximum Neighbor Heuristic (MNH). The maximum neighbor heuristic presented in Figure 5 also takes a set of operators Ω as input and returns the set of covering selections W. Each operator in Ω is assigned a priority level which is given by the number of processing neighbors. *MNH* then 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 P 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 (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 *MQC* 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 and recall of a pair of selections. The complexity of MQC is driven by i) generating the subsets (Line 4) and ii) applying Johnson's heuristic (Line 16). Generating all subsets has an expected cost of $O(|S_\Omega| n_\Omega n_s n_q)$ since the quality of each selection is compared against the quality of $n_\Omega n_s$ elements. Furthermore, Johnson's greedy heuristic will impose a cost of $O(|W||S_\Omega| + |S_\Omega| n_s n_\Omega)$. In each round—from beginning until termination—one element is inserted to W. Extracting the maximal element costs $O(\log |S_\Omega|)$ while updating the priorities of $|F| = |S_\Omega|$ elements can be achieved amortized in $O(|S_\Omega|)$ 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 $O(|S_\Omega| n_\Omega n_s n_q + |W||S_\Omega|)$. In contrast, MNH iterates over all selections $|S_\Omega|$ and makes in each iteration at most $|W| n_q$ comparisons. Hence, the time complexity for MNH is $O(|W||S_\Omega| n_q)$.

The performance for MNH is therefore expected to yield performance gains when the generation of subsets is the dominating factor. 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 quality 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 MQC, the complexity analysis shows that the computational overhead of the selection manager requires to carefully increase the scalability with appropriate mechanisms. In that context we ask:

- How many pairs of operators should compare their 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.

4.1.3 Streaming Phase

When an operator ω produces events as a result of processing a covering selection s_ω, the streaming phase is initiated. The produced events 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 VM. 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 phase is initiated. Note that because preceding operators can reuse different asynchronously processed selections, events might not arrive in a desired order. Each selection is therefore assigned a sequence number. 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.

4.2 Grouped Selection Processing

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.

4.2.1 Distributed Execution of a Selection Manager

Operators of the same type that process on behalf of consumers with overlapping interests have to be connected to the same selection manager to find covering selections. Therefore, 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.

However, note that the selection manager and its connected operators do not need to be hosted by the same VMs. Therefore, it is easy to integrate in RECEP placement algorithms to optimize latency and bandwidth usage (see [25]). Moreover, the number of operators connected to a selection manager influences the throughput of the selection manger 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 their consumer's focal point 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 RECEP controller if at least the focal point of one query is comprised within this area. Hence, 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 RECEP controller is a distributed component that scalably keeps track of the location of mobile consumers similar to location services [33]. 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. All selection managers are initially hosted by the same VM. Their placement can then be dynamically adapted with the previously mentioned operator placement mechanism by treating selection managers in the same way as operators. For example, for rectangular shaped interests Quad-Trees [24] can be used to efficiently update the deployment of the selection managers and find selection mangers to which an operator can connect to. This connection is dynamically adapted with location updates provided by the consumers.

4.2.2 Dynamic Processing Neighbor Selection

After a new group is established and operators are connected to selection managers, RECEP needs to 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 MQC in finding high quality covering selections. Besides, efficiently updating the

```
1: Distance d //maximal allowed distance between neighbors
2: number k_p //allowed number of neighbors
3: N //set of operators managed by the selection manager
4: upon init_reference_graph(Operator graph G)
5:   N(G) ← range_query(f(G), d)
6:   H(G) ← g ∈ N(G) with max Σ_{g'∈N} |I(g) ⋂ I(g')|)
7:   ∀ω ∈ G: trigger neighbor_selection(ω, H(G))
8: end
9: upon neighbor_selection(Operator ω, Op. graph H(G))
10:  Ω' ← k_range_nearest_neighbor(f(H(G)),d,k_p)
11:  for all ω ∈ Ω' do
12:      neighbor(ω') ← ω
13:  end for
14: end
```

Figure 6: Basic Coordinated Neighbor Selection

set of processing neighbors, we address how to achieve a low disparity which results from reusing selections that comprise events that result from reusing selections. For example, we aim to avoid the situation illustrated in Figure 2, where ω_2^{Q2} and ω_1^{Q1} are spatially dispersed and therefore a high disparity at ω_3^{Q3} is imposed.

Further observe, even though operators are grouped according to type and spatial region, the number of processing neighbors can differ for each operator. Since consumers can choose distinct quality thresholds, the relationship is in general not even symmetric.

Coordinated Neighbor Selection. The key idea behind the solution illustrated in Figure 6 is to consistently restrict the number of processing neighbors over all levels of the operator graph G. This is achieved by restricting the set of atomic events on which the set of operators that process on behalf of another operator depend. In particular: i) we reduce the maximum distance between the focal points of two operators that can process on behalf of the same operator and ii) we restrict the number of operators that select the same operator as processing neighbor.

The system determines in a first step at the level of the root operator of an operator graph G, a reference graph $H(G)$. Note that operators are always informed about the current location of a consumer [18]. The focal point of $H(G)$ is then used as a reference by all operators in G for choosing the operators to whom they are processing neighbors. To allow the reference graph to be chosen independently for each operator graph, we introduce a metric which enforces each operator to select a reference graph with similar focal point and this way reduce the disparity based on the dependency to atomic events. Let $N(G)$ be the set of operator graphs with a spatial overlap to the spatial interest imposed by G and with focal points that are at most a distance d away from the focal point of G (Line 5). Then the reference operator graph $H(G)$ is the operator graph in $N(G)$ which achieves the highest pairwise spatial overlap over all operator graphs N managed by the root's selection manager (Line 6). The focal point of $H(G)$ is then forwarded to all selection manager which are connected to an operator $\omega \in G$ (Line 7). Once the selection manager knows $H(G)$, it will determine the set of operators to whom ω is a processing neighbor by performing a k nearest range query centered in the focal point of $H(G)$. The query returns a maximum of k_p operators whose focal points are comprised within a radius of size d around the focal point of $H(G)$. (Line 10).

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 $H(G)$ and the selection of processing neighbors is performed in regular time intervals. A low time 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. An adaptive solution to determine such a time interval can be derived from safe time approaches [9], yet the details are left out for brevity. 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.

Note that the presented algorithm only intends to reduce the disparity for each operator graph. However, reusing selections from other operators still imposes the potential that the introduced disparity is exceeding a quality threshold.

4.3 Optimizations

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 QoR.

4.3.1 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, e.g., one instruction *speed* $<$ 20. 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.

Processing of Batches of Selections. Selections can be batched on a temporal basis or for a number of n_b subsequent selections. In the first case a selection batch b is assigned a time window $(t_s(b), t_e(b))$ and comprises all selections s where $t_s(s) \geq t_s(b)$ and $t_e(s) \leq t_e(b)$. In the latter case, 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., $t_s(b) = t_s(s_1)$ and $t_e(b) = t_s(s_n)$. Furthermore, there are two special selection batches: $n_b = 1$ denotes the individual reuse of selections as discussed until now, $n_b = \infty$ denotes the case where all selections are shared with the same processing neighbor during the whole run-time. 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 MQC 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 QoR-aware reuse. The system either cannot cover another batch if at least one selection of the batch does not yield an acceptable quality, or the consumer accepts that at least some selections in the batch have a bad quality and only the average QoR over all selections yields a better quality than QoR_{th}. How important it is to ensure QoR depends

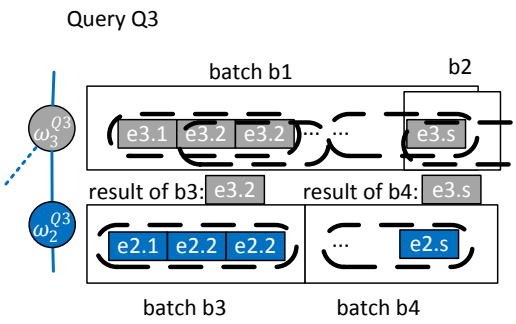


Figure 7: Batched reuse-aware processing

on the application, yet, only the latter approach can be efficiently implemented.

Recursive Operator Stop. Observe, so far an operator ω with a covered batch b_ω ceases to perform processing with respect to b_ω. However, preceding operators of ω may still perform processing steps and produce events which are subsequently discarded by ω. For example, if the batch b_1 in Figure 7 is covered by another batch, all events produced by b_3 are discarded—even if b_3 reuses another batch.

Therefore, RECEP offers mechanisms to stop the execution of an entire sub-graph of an operator graph. The protocol that implements such a stopping is invoked after the MNH or MQC determined a set of covering batches. For all batches that reuse the results of covering batches a stop message is sent to the selection managers that are connected to corresponding preceding operators, i.e., operators that provide the input streams to these batches. This message contains a time-interval $(t_s(b_i), t_e(b_i))$ that indicates which selections can be stopped with respect to a batch b_i and an id of the corresponding preceding operators. Selection managers buffer all received intervals for each operator and prevent corresponding batches that only comprise selections s with a start and end within these time-intervals, e.g., where $t_s(s) \in (t_e(b_i), t_s(b_i))$, from being processed or becoming a covering batch, iff they are not already selected as covering batch. Such a stop can be propagated even further down the operator graph, since now the predecessors of the predecessor can stop processing for all stopped batches. Note that $t_e(b)$ of a batch b is not fixed if b is defined for a number of selections and thus has to be estimated based on the inter-arrival time of the most recently arrived events.

4.3.2 Scalable Run-Time QoR 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 and predict the quality if not all events of a selection or batch are present at an operators incoming buffer.

Determining the QoR metrics. Both heuristics of Section 4.1.1 calculate the precision and recall for a pair of selections or batches based on estimates about atomic events they depend on, e.g., by annotating the ids of atomic events to processed events similar to [10]. The better these estimates are, the better the calculated quality. However, since calculating the quality according to the metrics given in Section 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 quality:

Spatial Interest Quality: The QoR can be determined by intersecting the spatial interests of queries that correspond to a pair of selections or batches. To this end, operators keep track of the current spatial interest of their corresponding query. 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 Quality: The number of shared incoming events with processing neighbors, i.e., their predecessors reused the processing of the same batch, can be used as indicator for the overlap in atomic events. 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 phase. The system determines, for each pair of selections s, s', the number m_o of events that are produced by a common predecessor and the number of events in those selections $m_s, m_{s'}$. Precision and recall thus evaluates to $\frac{m_o}{m_s}$ and $\frac{m_o}{m_{s'}}$. This method disregards events that stem from already reused selections and therefore already have a degraded quality. Hence, we annotate the calculated QoR for each selection that reuses the result to each outgoing event. The system can then weight m_o by the sum of all annotated QoRs of events from a common predecessor comprised in the selection. Since quality 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 Quality: 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 timespan, e.g., during the start and end of a selection. 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. 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. Congruent cells are joined by summing up the corresponding numbers of atomic events. We further denote these grids as reuse dependency. Moreover, events comprised in a selection can already depend on reused selections, which 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. 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 QoR according to the metrics given in Section 3. Cells with the same spatial coordinates comprise the number of events in the overlap. 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 QoR against overhead for maintaining estimates with the grid size.

Prediction. Due to the asynchronous processing, not all events that are comprised in a selection or batch might be present in the incoming buffers when calculating the QoR. Moreover, it can be computationally less intensive to predict the QoR from a sample of annotated estimates, instead of calculating precision and recall over all annotated estimates. In these cases we have to predict the QoR at run-time.

Pessimistic and optimistic approaches assume that no or all events that arrive in the future lie in interest overlaps of queries corresponding to selections. This means, if some events already arrived for a selection s and based on that information the system determines that m_x atomic events lie in the interest overlap with another selection s' which typically comprises $m_{s'}$ atomic events, then the recall would evaluate to $\frac{m_x}{m_{s'}}$ in the pessimistic case.

Event pattern prediction. Assuming that the past event pattern of selections indicates the future event pattern, the most recent received estimates about atomic events are buffered. The estimation of future atomic events comprised in a pair of selections can then be determined based on the buffered estimates using a bins and balls model. In order to estimate a QoR metric for a selection s, e.g., a precision $> \frac{k}{m_{s'}}$, where $m_{s'}$ is the number of atomic events typically comprised in the potential covering selection s', the system has to draw k atomic events from the interest overlap out of $m_{s'}$ draws. The corresponding probability to draw k or more atomic events can then be calculated using a hypergeometric distribution. Let K be the atomic events in the interest overlap of the buffered estimates and M all events in the buffered estimates of the processing neighbor, then the QoR can be predicted to $\frac{k_e}{m_{s'}}$, according to the expectation value of $k_e = n * \frac{K}{M}$. This means if the estimates for m_y atomic events are still missing for the determination, we assume that $m_y * \frac{K}{M}$ will arrive that lie in the overlap of these atomic events.

5. EVALUATION

We evaluated our reuse methods with two sets of experiments, each with a different operator graph. An approach without reuse, in particular the MCEP [18], was used as *baseline approach* in both cases. The first set of experiments (*mobility setup*) was using realistic movement patterns of cars generated by SUMO [4], the simulation environment Omnetpp [28], and an operator graph that resembled the one in Figure 1 that detects accidents (for details see [18]) to show the benefits of our approach. In the second set of experiments (*basic setup*) we determined the overhead of finding covering selections for different complex operators written in C++ without mobility.

In the mobility setup, approximately 500 cars were simulated on a street map of a German town (around 8 km^2 in size with corresponding speed limits). The cars emitted events when accelerating or decelerating beyond a threshold or changing lanes (approximately two event every second). A number of cars (*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. Each simulation was performed approximately 5 to 10 times using a different set of movement trajectories as input with each new run.

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 QoR was always determined according to the formulas given in Section 3 by annotating each event with the set of atomic events it depends on. The actual measured QoR is presented as the average precision and recall over all detected situational information (*avg. QoR*). Most results are presented as relative results, in particular relative to the baseline, MCEP, without reuse.

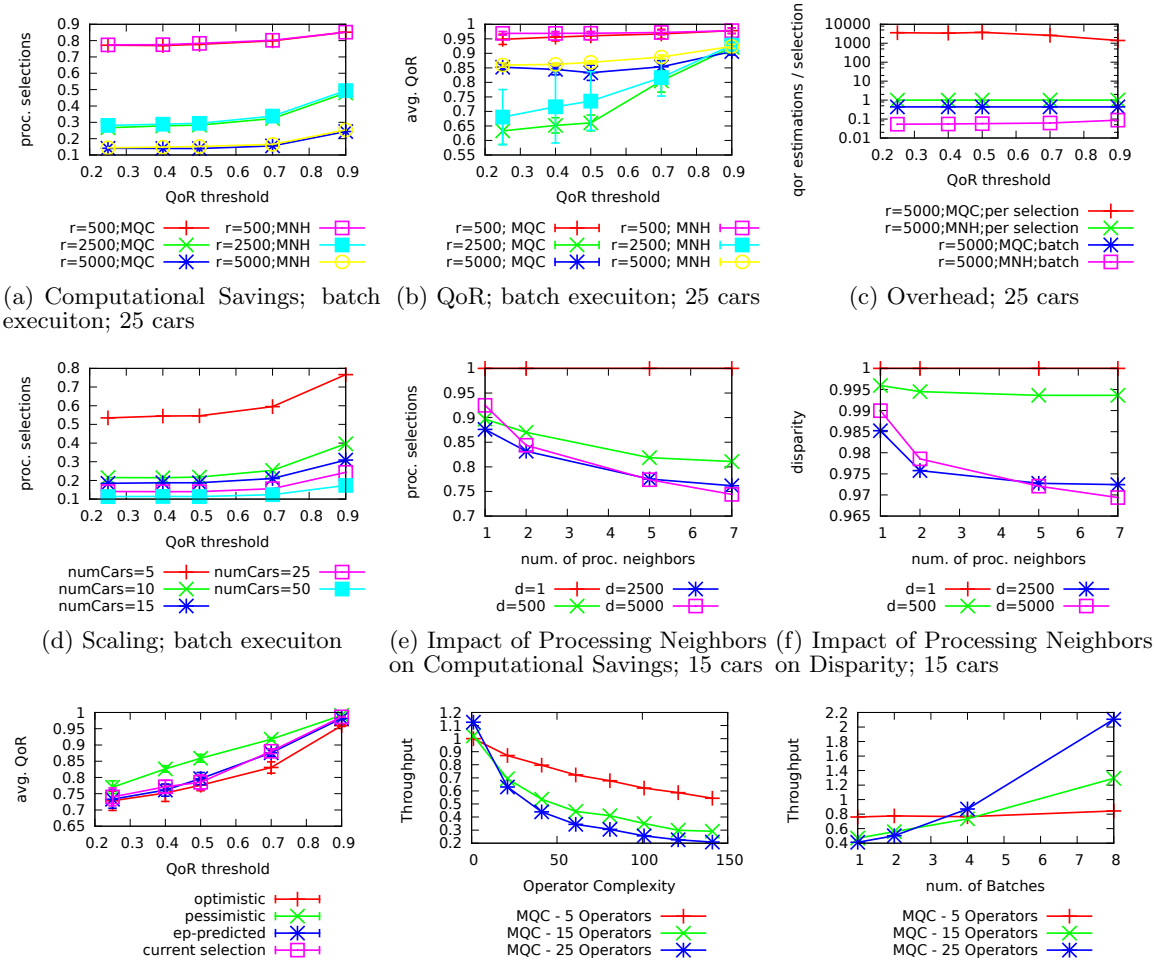
5.1 Scalability of the Set Cover Heuristics

With the mobility setup we tested how much computational savings we can achieve with MQC and MNH, how much these approaches affect the QoR, and how significant the overhead is.

We varied several control parameters. At first, the quality threshold QoR_{th}. Secondly, the side length (in m) of the spatial interest (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 8(a) presents the computational savings when reusing batches of selections. The x-axis depicts the QoR_{th} 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 QoR_{th} threshold the system is able to reuse by far more selections since more selections can cover each other. However, it can only reuse when 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 8(b) presents the effect of the reuse on the actually measured average QoRs. Since all operators of an operator graph cooperatively keep the threshold, the average quality 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 8(c) on the y-axis as the number of QoR 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.

We also varied the number of cars (*numCars*) that queried for traffic. The higher *numCars*, the more operators are deployed and more queries overlap on the very same or similar



(a) Computational Savings; batch executon; 25 cars

(b) QoR; batch executon; 25 cars

(c) Overhead; 25 cars

(d) Scaling; batch executon

(e) Impact of Processing Neighbors on Computational Savings; 15 cars

(f) Impact of Processing Neighbors on Disparity; 15 cars

(g) Impact of QoR Prediction; 5 cars

(h) Throughput ratio over operator complexity

(i) Throughput ratio over batch sizes

Figure 8: Evaluation of the RECEP System

spatial interest. We used the MQC for selection batches, while fixing the spatial interest to 5000 m.

The results in Figure 8(d) depict the relative computational savings in comparison to the baseline approach (y-axis) for different quality 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.

5.2 Impact of Processing Neighbors

Since increasing the number of processing neighbors (k_p) increases the potential to find operators to reuse with and the distance d between processing neighbors affects the disparity (see Section 4.2.2), 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 QoR threshold to 0.5, the spatial interest to 5000 m and reused individual selections with the MNH heuristic among 15 cars.

Figure 8(e) 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 8(f) the disparity drops with the number of neighbors and their relative distance to each other.

5.3 Impact of Quality Prediction Methods

To study the impact of the different quality prediction methods (see Section 4.3.2) 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 5000 m. Figure 8(g) shows how the different quality predictions affect the actually measured quality (y-axis) for different quality thresholds (x-axis). The optimistic approach reused many selections with actually bad qualities, since it always overestimated the quality 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 quality over the events in *current selection* settled between both approaches.

5.4 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 10000 selections and sliding windows that each comprised 30 events. We used the event pattern prediction method to reduce the overhead for estimating the QoR

and a naive implementation of the MQC. The results depicted in Figure 8(h) 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 8(i) depicts how the throughput ratio changed with the number of batches. More batches require more comparisons and estimations of the QoR, hence our system preforms especially well when the number of batches that have to be reused per operator is small.

6. 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 [11, 9, 31]. 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.

CEP and data base systems [30, 6, 26] typically allow 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 QoR 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 [3, 13, 20, 19] are mostly tailored towards specialized operators and not tailored towards a general solution as presented with this paper. 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 quality.

Filter operations [23, 27, 32] 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 [29] 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. Our previous work [12] used a set cover heuristic to select a minimal number of spatial interests for a single query, in contrast to this work, where we select a minimal number of reusable selections for multiple queries.

7. CONCLUSION

In this paper we presented a method for sharing computations between stateful operators in distributed CEP systems. In the context of a situation-awareness 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.

Beyond the methods proposed as part of this paper, we plan to study as part of future work how to further reduce resources needed in meeting consumer-centric quality requirements. In this direction, we intend to investigate the potential of a tighter coupling between methods for placing operators and the proposed methods for reuse. Furthermore, to ease the deployment of RECEP-based applications, we will research how to self-configure many application-specific RECEP parameters like the batch-size or the number of processing neighbors that at current stage need to defined by the programmer.

8. REFERENCES

[1] www.rita.dot.gov/bts/sites/rita.dot.gov.bts/files /publications/national_transportation_statistics/ html/table_01_36.html. online, 2011. [online; accessed 2014-04-10].

[2] http://www.capgemini.com/resources/cars-online-1213. online, 2013. [online; accessed 2014-04-10].

[3] A. Assefa and F. Getahun. Multi-query Optimization for Semantic News Feed Query. In *Proc. of Int. Conf. on Management of Emergent Digital EcoSystems*, MEDES '12, pages 150–157. ACM, 2012.

[4] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz. SUMO - Simulation of Urban MObility: An Overview. In *Proc. of 3rd Int. Conference on Advances in System Simulation (SIMUL)*, pages 63–68, Oct. 2011.

[5] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog Computing and Its Role in the Internet of Things. In *Proc. of 1st MCC workshop on Mobile Cloud Computing*, pages 13–16. ACM, 2012.

[6] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *Proc. of 2000 ACM SIGMOD Int. Conf. on Management of Data*, SIGMOD '00, pages 379–390. ACM, 2000.

[7] T. H. Cormen, C. Stein, R. L. Rivest, and C. E.

Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[8] G. Cugola and A. Margara. TESLA: A Formally Defined Event Specification Language. In *Proc. of 4th ACM Int. Conf. on Distributed Event-Based Systems*, DEBS '10, pages 50–61. ACM, 2010.

[9] B. Gedik and L. Liu. MobiEyes: A Distributed Location Monitoring Service Using Moving Location Queries. *IEEE Transactions on Mobile Computing*, 5:1384–1402, 2006.

[10] B. Glavic, K. Sheykh Esmaili, P. M. Fischer, and N. Tatbul. Ariadne: Managing Fine-grained Provenance on Data Streams. In *Proc. of 7th ACM Int. Conf. on Distributed Event-based Systems*, DEBS '13, pages 39–50. ACM, 2013.

[11] A. M. Hendawi and M. F. Mokbel. Panda: A Predictive Spatio-Temporal Query Processor. In *Proc. of 20th Int. Conf. on Advances in Geographic Information Systems*, SIGSPATIAL '12, pages 13–22. ACM, 2012.

[12] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe. Opportunistic Spatio-temporal Event Processing for Mobile Situation Awareness. In *Proc. of 7th ACM Int. Conf. on Distributed Event-Based Systems*, DEBS '13, pages 195–206. ACM, 2013.

[13] M. Hong, M. Riedewald, C. Koch, J. Gehrke, and A. Demers. Rule-based Multi-query Optimization. In *Proc. of 12th Int. Conf. on Extending Database Technology: Advances in Database Technology*, EDBT '09, pages 120–131. ACM, 2009.

[14] A. Ishii and T. Suzumura. Elastic Stream Computing with Clouds. In *Proc. of 2011 IEEE Int. Conf. on Cloud Computing*, CLOUD, pages 195–202, 2011.

[15] D. S. Johnson. Approximation Algorithms for Combinatorial Problems. In *Proc. of 5th annual ACM Symp. on Theory of Computing*, STOC '73, pages 38–49. ACM, 1973.

[16] G. G. Koch, B. Koldehofe, and K. Rothermel. Cordies: Expressive event correlation in distributed systems. In *Proc. of the 4th ACM Int. Conf. on Distributed Event-Based Systems*, DEBS '10, pages 26–37. ACM, 2010.

[17] B. Koldehofe, R. Mayer, U. Ramachandran, K. Rothermel, and M. Völz. Rollback-Recovery without Checkpoints in Distributed Event Processing Systems. In *Proc. of 7th ACM Int. Conf. on Distributed Event-Based Systems*, DEBS '13, pages 27–38. ACM, 2013.

[18] B. Koldehofe, B. Ottenwälder, K. Rothermel, and U. Ramachandran. Moving Range Queries in Distributed Complex Event Processing. In *Proc. of 6th ACM Int. Conf. on Distributed Event-Based Systems*, DEBS '12, pages 201–212. ACM, 2012.

[19] S. Krishnamurthy, M. J. Franklin, J. M. Hellerstein, and G. Jacobson. The Case for Precision Sharing. In *Proc. of 30th Int. Conf. on Very Large Data Bases*, VLDB '04, pages 972–984. VLDB Endowment, 2004.

[20] S. Krishnamurthy, C. Wu, and M. Franklin. On-the-fly Sharing for Streamed Aggregation. In *Proc. of 2006 ACM SIGMOD Int. Conf. on Management of Data*, SIGMOD '06, pages 623–634. ACM, 2006.

[21] R. Lange, H. Weinschrott, L. Geiger, A. Blessing, F. Dürr, K. Rothermel, and H. Schütze. On a Generic Uncertainty Model for Position Information. In *QuaCon*, LNCS 5786, pages 76–87. Springer, 2009.

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

[23] S. Madden, M. Shah, J. M. Hellerstein, and V. Raman. Continuously Adaptive Continuous Queries over Streams. In *Proc. of 2002 ACM SIGMOD Int. Conf. on Management of Data*, SIGMOD '02, pages 49–60. ACM, 2002.

[24] A. Meka and A. Singh. DIST: A Distributed Spatiotemporal Index Structure for Sensor Networks. In *Proc. of 14th ACM Int. Conf. on Information and Knowledge Management*, CIKM '05, pages 139–146. ACM, 2005.

[25] B. Ottenwälder, B. Koldehofe, K. Rothermel, and U. Ramachandran. MigCEP: Operator Migration for Mobility Driven Distributed Complex Event Processing. In *Proc. of 7th ACM Int. Conf. on Distributed Event-Based Systems*, DEBS '13, pages 183–194. ACM, 2013.

[26] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and Extensible Algorithms for Multi Query Optimization. In *Proc. of 2000 ACM SIGMOD Int. Conf. on Management of Data*, SIGMOD '00, pages 249–260. ACM, 2000.

[27] M. A. Tariq, B. Koldehofe, G. G. Koch, I. Khan, and K. Rothermel. Meeting subscriber-defined QoS constraints in publish/subscribe systems. *Concurrency and Computation: Practice and Experience*, 23(17):2140–2153, 2011.

[28] A. Varga and R. Hornig. An Overview of the OMNeT++ Simulation Environment. In *Proc. of the 1st Int. Conf. on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, pages 1–10. ICST, 2008.

[29] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt. Leveraging Cloudlets for Immersive Collaborative Applications. *Pervasive Computing, IEEE*, 12(4):30–38, 2013.

[30] S. Xiang, H. B. Lim, and K.-L. Tan. Impact of Multi-query Optimization in Sensor Networks. In *Proc. of 3rd Workshop on Data Management for Sensor Networks: In Conjunction with VLDB 2006*, DMSN '06, pages 7–12. ACM, 2006.

[31] X. Xiong, M. Mokbel, and W. Aref. SEA-CNN: Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-temporal Databases. In *Proc. of 21st Int. Conf. on Data Engineering*, ICDE 2005, pages 643–654, 2005.

[32] Z. Xu and H.-A. Jacobsen. Expressive Location-Based Continuous Query Evaluation with Binary Decision Diagrams. In *Proc. of 2009 IEEE Int. Conf. on Data Engineering*, ICDE '09, pages 1155–1158. IEEE, 2009.

[33] J. Zhang, G. Zhang, and L. Liu. GeoGrid: A Scalable Location Service Network. In *Proc of 27th Int. Conf. on Distributed Computing Systems*, ICDCS '07, page 60. IEEE, June 2007.