MuSE Graphs for Flexible Distribution of Event Stream Processing in Networks

Technical Report

Samira Akili
Humboldt-Universität zu Berlin
Berlin, Germany
akilsami@hu-berlin.de

Matthias Weidlich
Humboldt-Universität zu Berlin
Berlin, Germany
matthias.weidlich@hu-berlin.de

ABSTRACT

Complex event processing (CEP) enables reactive and predictive applications through the continuous evaluation of queries over streams of event data. In a network of event sources, efficient query evaluation is achieved through distribution: Queries are split into operators (query decomposition), which are then assigned to some of the nodes (operator placement). Yet, existing solutions show two fundamental issues: They limit the decomposition to the operator hierarchy of a query, ignoring possible rewritings of it, and place each operator at exactly one node in the network. That neglects potential optimizations related to the composition of patterns through multiple queries as results are always gathered at a single sink node.

In this paper, we propose a new evaluation model for CEP, coined Multi-Sink Evaluation (MuSE) graphs. It incorporates arbitrary projections of queries for distribution and assigns them to potentially many nodes. We prove correctness of query evaluation with MuSE graphs and provide a cost model to assess its efficiency. Since the construction of cost-optimal MuSE graphs is intractable, we present an approximation algorithm and several pruning strategies. Our evaluation results show that MuSE graphs reduce network transmission costs by up to three orders of magnitude over baseline strategies. An implementation in a state-of-the-art framework for distributed computing highlights significant latency and throughput improvements.

CCS CONCEPTS

• **Information systems → Data streams**.

KEYWORDS

complex event processing, operator placement, query distribution

1 INTRODUCTION

Complex event processing (CEP) is a computational paradigm that is based on the continuous evaluation of queries over event streams [14]. A query correlates events of particular types through operators and a time window. CEP has been employed in domains such as transportation [5], electric grids [12], or finance [23].

Recently, applications that employ CEP over a network of event sources have gained importance, e.g., in the context of the internet-of-things [13, 22]. Following the model of traditional CEP, events of all sources are gathered at a single location, so that queries can be evaluated over a unified event stream. While such a centralized model has advantages in terms of observability and traceability, its scalability is inherently limited: *All* events need to be sent over the network, even if only a small fraction of them is required to detect the queried patterns. Moreover, conducting query evaluation, which shows exponential runtime in the number of processed events [26], at a single location introduces a potential performance bottleneck.

Approaches to distributed CEP, in turn, leverage that event sources can often also serve as event processors, so that the network structure can be exploited to reduce data transmission costs [9, 13]. Then, a query workload is modularized by splitting the queries in a set of operators, which are assigned to network nodes for evaluation. Any realization of this idea faces the following questions:

Q1 *How to decompose a query into operators?* Here, the challenge is to ensure correctness once the patterns detected by the query are composed from those of its individual operators.

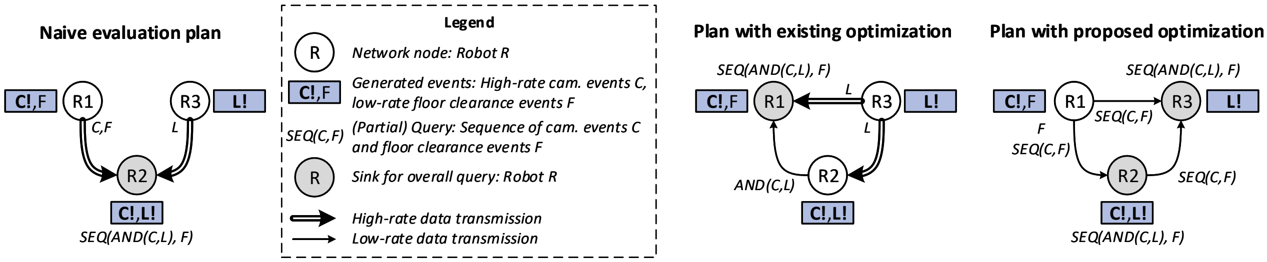
Q2 *How to place operators at network nodes?* The challenge is to place operators such that the transmission costs between nodes are minimized, see also [18].

Q3 *How to compute query decompositions and operator placements efficiently?* Given the combinatorial explosion of candidate solutions, it is challenging to develop tractable methods.

Existing solutions for distributed CEP largely ignore Q1 and consider solely operators that are given directly as part of the operator hierarchy defined by a query [13]. Also, approaches for Q2 are limited as they consistently follow a centralised approach that collects all query results at *one* designated sink node [22]. Even under these assumptions, operator placement was shown to be NP-hard [25], so that heuristic approximations are commonly employed [18].

The above design choices realized in existing work imply that query evaluation is inherently hierarchical and limited by the syntactical structure of any query. This has major drawbacks. First, it prevents us from exploiting that some nodes produce only the events to detect a *partial* pattern for an operator of a given query. Second, while distributed operator placement reduces data transmission costs, a single sink is still a potential performance bottleneck, similar to a centralized model. Hence, opportunities for efficient query evaluation are neglected, especially in non-hierarchical systems (e.g., networks of mobile clients or autonomous agents [9]), where patterns are handled in a decentralized manner. A single sink also undermines the compositionality of CEP, where the result of one query serves as input for another one. Here, only the first level of queries in a composition would benefit from distributed evaluation.

We illustrate the above issues for a case of autonomous transport robots that serve machines in a factory and communicate via WiFi [16]. Some of them are equipped for obstacle detection, e.g., with cameras (C) and lidar sensors (L), while others also report on a certain floor segment being clear to be used (F). Fig. 1 shows a setting with three robots R1-R3, emitting events of different types (e.g., R2 emits solely camera and lidar events) and rates (camera



(a) (b) (c)

Figure 1: Setting of three transport robots R1-R3 emitting events of camera (C) and lidar (L) sensors with high rates, and, rarely, floor clearance events (F): (a) Data transmission when evaluating the query *SEQ(AND(C,L),F)* naively; (b) with existing optimization strategies; (c) using the ideas proposed in this paper.

and lidar sensors have high rates, floor clearance is emitted rarely). Consider a query *SEQ(AND(C,L),F)* checking for an obstacle detected by both types of sensors, followed by a floor clearance event. A naive query evaluation plan, Fig. 1a, gathers all atomic events at one robot, which yields high data transmission costs. Existing optimization strategies decompose the query and exploit the operator *AND(C,L)* in the placement, which, due to its selectivity, reduces data transmission, see Fig. 1b. Yet, the exchange of events of highly frequent types is only avoided entirely once decomposition is not limited to the operator hierarchy and once query evaluation involves multiple sinks. Fig. 1c shows how the use of arbitrary query projections (i.e., *SEQ(C,F)*) and multiple sinks (i.e., R2 and R3) enables a more comprehensive reduction of data transmission costs.

In this paper, we propose a new evaluation model for distributed CEP that enables flexibility in the aforementioned dimensions. Given a query workload, we rely on arbitrary projections of queries as operators for query evaluation (Q1), while our placements achieve low data transmission costs by potentially defining multiple sinks (Q2). We also show how plans in this model are computed efficiently (Q3).

We summarize our contributions, and the paper structure following preliminaries (§2) and a problem statement (§3), as follows:

- *MuSE Graphs* (§4): We present Multi-Sink Evaluation (MuSE) graphs as an evaluation model for distributed CEP. A MuSE graph assigns query projections to network nodes and defines the exchange of partial patterns within the network.
- *Analysis of MuSE Graphs* (§5): We formally characterize the correctness of a MuSE graph and introduce a notion of cost-based optimality to reason on its evaluation efficiency.
- *Computation of MuSE Graphs* (§6): We show how to compute an optimal MuSE graph. Since this construction is NP-hard, we present pruning and approximation strategies.

We evaluate our techniques in simulation experiments and in a case study using real-world data and a contemporary framework for distributed computing (§7). Our results indicate reductions in transmission costs of up to three orders of magnitude compared to baseline strategies. Finally, we discuss related work (§8) and conclude (§9).

2 PRELIMINARIES

2.1 Event-sourced network

Let $\mathcal{E} = \{E_1, \ldots, E_n\}$ be the universe of *event types*. Following common streaming models [3, 14], each type defines a schema for a set of events of similar semantics and structure. An *event* is an instantiation of an event type, which is further assigned a unique identifier and an occurrence timestamp. We write $e.type \in \mathcal{E}$ for the type of an event e and $e.time \in \mathbb{N}$ for its timestamp.

An *event-sourced network* $\Gamma = (N, f, r)$ is defined by a set of nodes N, a function $f : N \to 2^{\mathcal{E}}$ that assigns event types to nodes, and a function $r : \mathcal{E} \to \mathbb{R}$ that assigns rates to event types. As such, function f models a network of heterogeneous nodes that differ in their ability to emit events. For an event e, we write $e.origin \in N$ for the node that generated e. Moreover, function r captures heterogeneity related to event sourcing (e.g., sensor types) in terms of the event generation rates, i.e., the number of events per time unit. We consider networks in which all nodes may directly exchange events, so that the network can be thought of as a complete graph.

Example 1. *Consider again the example of transport robots [16]. Fig. 1 shows an event-sourced network with $N = \{R1, R2, R3\}$ and, e.g., $f(R2) = \{C, L\}$. Also, we already discussed that $r(C) \gg r(F)$ and $r(L) \gg r(F)$ – camera and lidar sensors have high rates.*

A node $n \in N$ generates a *local trace* $t(n) = \langle e_1, e_2, \ldots \rangle$, an infinite sequence of events, ordered by their timestamps. Interleaving local traces of all nodes yields a *global trace* of the network, which is assumed to be totally ordered. The global trace is a conceptual notion. It is never materialised, but required to define an unambiguous semantics of queries over events generated in a network. It may be seen as the equivalent to collecting all events at a single node and ordering them by a distinguished timestamp (e.g., valid time, occurrence time, or arrival time [6]), resolving ties deterministically.

2.2 Query language

Syntax. We adopt a common query language for CEP [4, 14]. A *query* is composed of *operators*, a set of *predicates* to define conditions based on the events' payload data, and a *time window*. *Primitive operators* detect events of a specific type, while *composite operators* detect event patterns based on the results of their child operators. We consider the following composite operators:

- *AND*, detects a pattern if a trace contains patterns as defined by all its child operators, regardless of their ordering;
- *SEQ*, detects a pattern if a trace contains patterns as defined by all its child operators in the order specified by the operator;
- *OR*, detects a pattern if a trace contains at least one of the patterns as defined by its child operators;
- *NSEQ*, an operator with three children that detects a pattern if a trace contains the patterns of the first child operator,

followed by those of the last child operator, if the pattern of the middle child operator does not occur in between.

A query is represented as an ordered tree annotated with predicates and a time window. Formally, a query $q = (O, \lambda, P)$ contains a set of operators O, a partial function $\lambda : O \nrightarrow O^k$, $k \in \mathbb{N}$ and $k > 1$, that assigns a sequence of child operators to an operator, and a set of Boolean predicates P. The predicates are defined over constants and variables for the (transitively) contained primitive operators. Splitting up complex predicates, we assume predicates to be defined over at most two primitive operators and to be independent of each other. As a short-hand, we write $o <_\lambda o'$ for the parent-child relation, i.e., $\lambda(o) = \langle o_1, \ldots, o_k \rangle$ and $o_i = o'$ for some $1 \leq i \leq k$.

A query q has a time window, τ_q, which sets a bound for the max timestamp difference within a pattern. This window is of minor importance for distribution, so that we do not include it in our model.

The primitive operators, $O_p \subseteq O$, are those without children, i.e., for all $o \in O_p$ it holds that $o \notin \mathrm{dom}(\lambda)$. The remaining ones are composite operators, $O_c = O \setminus O_p$. In the context of multiple queries, we write O_p^q (O_c^q) to denote the primitive (composite) operators of a query q. Semantics of a primitive operator $o \in O_p$ is given by an event type, denoted by $o.sem \in \mathcal{E}$. A composite operator $o \in O_c$ carries an operator type, $o.sem \in \{AND, SEQ, OR, NSEQ\}$.

A query $q = (O, \lambda, P)$ is valid, if (O, λ) defines a tree of operators with a single root, denoted $root(q)$, and if no two directly nested operators have the same type, for all $o, o' \in O_c$ with $o <_\lambda o'$ it holds that $o.sem \neq o'.sem$. Table 1 summarizes our notation.

Semantics. Let $q = (O, \lambda, P)$ be a query and $t = \langle e_1, e_2, \ldots \rangle$ be a global trace of an event-sourced network. Then, evaluating q over t yields a set of *matches*, M^q (M for short), defined as a set of sequences of events. For such event sequences m and m', we denote their concatenation by $m.m'$ and the set of all possible interleavings by $\{m, m'\}_\simeq$. The set of matches M is then defined recursively.

For a primitive operator $o \in O_p$, M contains sequences $\langle e \rangle$ for all events e of t of the respective type, $e.type = o.sem$, that satisfy P.

Next, let $o \in O_c$ be a composite operator. For the operators that do not include negation, the set of matches M is derived from the sets of matches $M_1, \ldots, M_k$ of the child operators $\lambda(o) = \langle o_1, \ldots, o_k \rangle$. For a conjunction, $o.sem = AND$, the matches are derived from all interleavings, i.e., $M = \{m \in \{m_1, \ldots, m_k\}_\simeq \mid \forall\ 1 \leq i \leq k : m_i \in M_i\}$. For a sequence, $o.sem = SEQ$, the matches are concatenated, $M = \{m_1. \ \ldots\ .m_k \mid \forall\ 1 \leq i \leq k : m_i \in M_i\}$. For a disjunction, $o.sem = OR$, the matches are unified, $M = M_1 \cup \ldots \cup M_k$.

The semantics of the *NSEQ* operator is based on the sets of matches M_1, M_2, M_3 of its three child operators and the global trace $t = \langle e_1, e_2, \ldots \rangle$. The matches are defined through the concatenation of matches of the first and third child, if none of the matches of the second child is observed in between. With $\#_t$ as an auxiliary function that maps events to their index in t, the matches of *NSEQ* are $M = \{m_1.m_3 \mid m_1 = \langle e_1^1, \ldots, e_n^1 \rangle \in M_1 \wedge m_3 = \langle e_1^3, \ldots, e_m^3 \rangle \in M_3 \wedge \forall\ m_2 = \langle e_1^2, \ldots, e_l^2 \rangle \in M_2 : \#_t(e_1^2) < \#_t(e_n^1)\ \vee\ \#_t(e_l^2) > \#_t(e_1^3)\}$.

In any case, predicates of the operators and the time window need to be respected: M contains only matches $m = \langle e_1, \ldots, e_n \rangle$ that satisfy all predicates in P and where $e_n.time - e_1.time \leq \tau_q$. We overload notation and write $e \in m$ if event e is part of the event sequence that defines the match m. Moreover, by $\sigma(a)$, we denote the selectivity of predicate $a \in P$, i.e., the ratio of events satisfying it. Then, $\sigma(q) = \Pi_{a \in P} \sigma(a)$ is the selectivity of the query.

Table 1: Overview of notations for networks and queries.

Notation	Explanation
$\mathcal{E}$	Universe of event types
e	Event, with type $e.type \in \mathcal{E}$ and timestamp $e.time \in \mathbb{N}$
$\Gamma = (N, f, r)$	Event-sourced network: nodes N, event types $f : N \to 2^{\mathcal{E}}$, rates $r : \mathcal{E} \to \mathbb{R}$
$q = (O, \lambda, P)$	Query: operators O, children $\lambda : O \nrightarrow O^k$, predicates P
$O_p^q, O_c^q \subseteq O$	Primitive and composite operators of query q
$root(q) \in O$	Root operator of query q
$o.sem$	Operator semantics: Event type for primitive operator, one of $\{AND, SEQ, OR, NSEQ\}$ for composite operator
M, M^q	Matches of a query (q) in an event-sourced network
$\sigma(a), \sigma(q)$	Selectivity of predicate $a \in P$ and of the whole query q
Q	Query workload: a set of queries

The above semantics corresponds to a *greedy* event selection policy, also known as skip-till-any-match [1]. We focus on this policy as it a very challenging one: The number of matches may grow exponentially in the number of processed events [26].

Query workload. We define a *query workload* as a set of queries $Q = \{q_1, \ldots, q_n\}$, which are free of disjunction operators. Since each query with a composite operator of type *OR* can be split into multiple queries containing solely *SEQ*, *AND*, and *NSEQ* operators, this does not constrain the expressiveness of our model.

The queries of a workload can be represented as a forest. To simplify the discussion, we consider workloads in which queries are related, i.e., they share some of their composite operators. Specifically, for each query $(O, \lambda, P) \in Q$, there exists at least one distinct query $(O', \lambda', P') \in Q$ and it holds that $O_c \cap O_c' \neq \emptyset$. Then, the structure of the queries in the workload can be represented as a directed acyclic graph. Workloads that do not satisfy this condition may be treated independent of each other, though. Moreover, all queries of a workload are assumed to define the same time window. Practically, this is achieved by adopting the largest window among the queries, while matches are potentially filtered based on smaller windows when evaluating the root operators of the individual queries.

3 PROBLEM STATEMENT

Having defined a model for event stream processing, we are ready to capture the problem of efficient distributed query evaluation. Let $\Gamma = (N, f, r)$ be an event-sourced network and $Q = \{q_1, \ldots, q_n\}$ be a query workload to execute in this network. At an abstract level, an *evaluation plan* is a function μ that takes the network and the query workload as input and returns the, generally unbounded, sets of matches $M_1, \ldots, M_n$ for each of the queries.

An evaluation plan derives the matches by evaluating query operators at the nodes of the network and exchanging intermediate results between them. As such, a specific plan μ can be assigned a cost of data transmission, denoted by $c(\mu) \in \mathbb{R}$, induced by the exchange of intermediate results between operators at different nodes. This cost is grounded in the rates with which events are produced in the network. Therefore, it is a time-invariant measure.

A simple plan is the naive evaluation of a query workload, see Fig. 1a. All operators are evaluated at one distinguished node, while all other nodes transmit the required events to this node. Then, the cost of the plan is derived directly as the sum of the rates of these types and the number of nodes (except the distinguished node) that produce the respective events.

EXAMPLE 2. *Consider the plan in Fig. 1a for naive evaluation of the query* SEQ(AND(C,L),F). *If the query is evaluated at R2, as shown, the cost of the plan is $r(F) + r(C) + r(L)$. An evaluation of the query at R3, in turn, would have a higher cost of $r(F) + 2r(C) + r(L)$.*

Using this abstract cost model, we define the problem addressed in this paper, as follows.

PROBLEM 1 (EFFICIENT DISTRIBUTED QUERY EVALUATION). *Given an event-sourced network and a query workload, the problem of* efficient distributed query evaluation *is to construct an evaluation plan μ, such that $c(\mu)$ is minimal.*

4 THE MUSE GRAPH MODEL

To address the problem of efficient, distributed query evaluation, we propose Multi-Sink Evaluation (MuSE) graphs as a novel model to define evaluation plans. We first discuss the notions of event type bindings (§4.1) and query projections (§4.2) as essential building blocks of this model, before introducing the definition of a MuSE graph (§4.3). We then define a cost model for MuSE graphs (§4.4). Notations for MuSE graphs, as well as for the notions needed to reason on their properties in the remainder, are summarized in Table 2.

Table 2: Overview of notations of the MuSE graph model.

Notation	Explanation
$\mathfrak{e}$	Event type binding: bag of tuples (E, n), with event type $E \in \mathcal{E}$ and node $n \in N$. Contains tuple $(o.sem, n)$ for each primitive operator $o \in O_p$ and each node $n \in N$ with $o.sem \in f(n)$.
$\mathfrak{E}(q)$, $\mathfrak{E}(\Gamma, q)$	Set of event type bindings of a query q (in a network Γ)
$\pi(q, \mathcal{E}')$	Projection of q induced by a set of event types $\mathcal{E}' \subseteq \mathcal{E}'$
$\Pi(q)$	Set of all possible projections of q
$G = (V, E, c)$	MuSE graph: vertices V, edges $E \subseteq V \times V$, edge weights $c : E \to \mathbb{R}$
$c(G)$	Sum of all edges weights of MuSE graph G
$\mathfrak{U}(v)$	Event type bindings covered by a vertex v of a MuSE graph
$\hat{r}(p)$	Output rate of a projection p
$\mathfrak{c} = (\mathfrak{P}, \beta)$	Combination graph: projections $\mathfrak{P}$, predecessors $\beta : \mathfrak{P} \nrightarrow \mathfrak{P}^k$
$\mathfrak{C}(Q)$, $\mathfrak{C}(q)$	Set of all combinations of a query workload Q (for a query q)

4.1 Event Type Bindings

In an event-sourced network, several nodes may generate events of the same type. Therefore, the events of a specific type that are part of matches of a query may differ in their origin. We capture the pairs of event types and nodes that may serve as the origin of events of the type by the notion of an event type binding. Recall that queries of a query workload are free of disjunction operators, see §2.2.

DEFINITION 1 (EVENT TYPE BINDING). *Let $\Gamma = (N, f, r)$ be an event-sourced network and $q = (O, \lambda, P)$ be a query of a query workload. An event type binding $\mathfrak{e}$ is a bag of tuples (E, n) of an event type $E \in \mathcal{E}$ and a node $n \in N$ that may contribute to* one *match of the query, i.e., $\mathfrak{e}$ contains a tuple $(o.sem, n)$ for each primitive operator $o \in O_p$ where n is one of the nodes with $o.sem \in f(n)$.*

An event type binding gives the possible origins that may contribute to a single match. Capturing all such possibilities, we use a function $\mathfrak{E}$ that, applied to an event-sourced network and a query, $\mathfrak{E}(\Gamma, q)$, returns the set of all event type bindings of q in Γ. If the network is clear from the context, we write $\mathfrak{E}(q)$ as a short-hand.

EXAMPLE 3. *Continuing with our example, the query is formalized as q_1 in Fig. 2a (top). For the network given as Γ, Fig. 2a (middle) lists all possible event type bindings, e.g., $[(F, 1), (C, 1), (L, 2)]$.*

The number of event type bindings of a query $q = (O, \lambda, P)$ in a network $\Gamma = (N, f, r)$ is bound by $|N|^{|O_p|}$. All sets of $|O_p|$ nodes, i.e., the number of events per query match, may be considered.

4.2 Query Projections

As mentioned, existing approaches exploit solely the operator hierarchy of a query for the placement of operators at network nodes. To lift this limitation, we present query projections as a flexible mechanism to derive operators for partial matches of the query. Intuitively, a query projection is the restriction of a query to a set of event types. Unlike traditional notions of sub-patterns, matches of projections are not necessarily contiguous sub-sequences of query matches.

DEFINITION 2 (QUERY PROJECTION). *Let $q = (O, \lambda, P)$ be a query and $\mathcal{E}' \subseteq \mathcal{E}$ be event types. The projection of q induced by $\mathcal{E}'$ is a query $p = (O', \lambda', P')$ such that $O'_p = \{o \in O_p \mid o.sem \in \mathcal{E}'\}$ and for all $o \in O'$, $\lambda'(o)$ is the projection of $\lambda(o)$ to O' and $P' \subseteq P$ is the subset of predicates over constants and variables for operators in O'_p. We also write $p = \pi(q, \mathcal{E}')$ for the respective projection.*

The set of all possible projections of a query q is denoted by $\Pi(q)$. Its size is bound by $2^{|O_p|}$.

EXAMPLE 4. *Fig. 2a (bottom) shows projections for query q_1 for the sets of event types $\{C, F\}$ (p_1), $\{L, F\}$ (p_2), and $\{C, L\}$ (p_3).*

A query projection is defined based on a subset of the primitive operators of the original query and comprises the complete subset of predicates that have been defined in relation to these operators. Hence, the projection of any match of the original query q to the events of a set of types $\mathcal{E}'$ will be a match of the respective query projection $p = \pi(q, \mathcal{E}')$. In the same vein, the selectivity of p is linked to the selectivity of q. Since $P' \subseteq P$, $\sigma(p)$ corresponds to the product of the selectivities of the shared predicates.

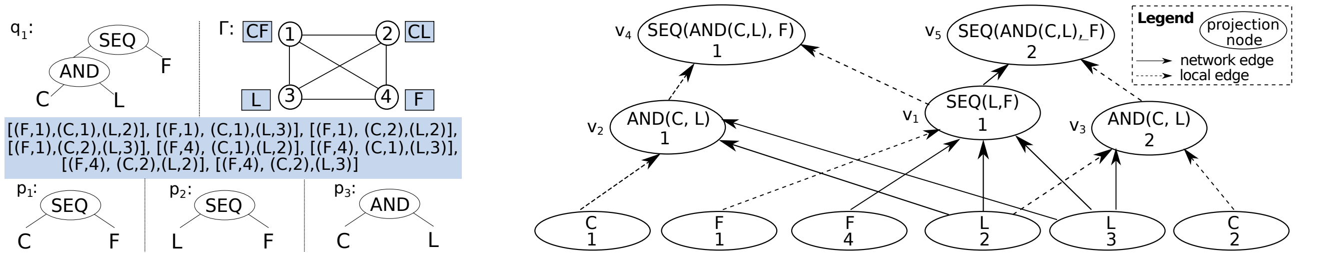
Given a query q and a set of event types $\mathcal{E}'$, the generation of the respective projection follows a simple algorithm. In a first step, all leaf operators of q that do not relate to types in $\mathcal{E}'$ are removed. If this results in a (non-leaf) operator o having no child, o is removed entirely. If o has a single child o', o is removed and o' becomes a child of the parent of o (or the new root, if o was the root).

EXAMPLE 5. *Taking up the projections of Fig. 2a (bottom), p_1 and p_2 are derived by deleting the primitive operators for either L or C, which leads to the deletion of their parent operator AND. Projection p_3 is generated by deleting the primitive operator for F, which also removes the old root operator SEQ.*

Finally, since query projections are defined as queries, the notion of an event type binding from §4.1 is also applicable for projections. Here, we note that event type bindings of a projection are sub-bags of the bindings of the original query.

4.3 Multi-Sink Evaluation Graphs

We are now ready to introduce Multi-Sink Evaluation (MuSE) graphs for distributed query evaluation. A MuSE graph specifies for a given event-sourced network and a query workload Q:



(a) **(b)**

Figure 2: (a) Query q_1 to be evaluated in network Γ (top), with its event type bindings (middle), and query projections p_1 - p_3 (bottom). (b) MuSE graph for the example. Dashed edges have a weight of zero as the respective operators are hosted by the same node.

i. which projections of the queries in Q are used for evaluation;
ii. where the operators of these projections are placed; and
iii. how matches of projections are exchanged between nodes.

DEFINITION 3 (MUSE GRAPH). *Let $\Gamma = (N, f, r)$ be an event-sourced network and Q be a query workload. A MuSE graph is a weighted directed acyclic graph $G = (V, E, c)$ with the vertices being pairs of a query projection and a node, $V \subseteq \bigcup_{q \in Q} \Pi(q) \times N$, and edges $E \subseteq V \times V$ being assigned a weight, $c : E \to \mathbb{R}^+$.*

Next, we explain the intuition of the model, while its formal properties are explored later. Let p, p' be projections of a query $q \in Q$ and $n, n' \in N$ be network nodes. A vertex $(p, n) \in V$ of a MuSE graph denotes that matches of projection p are generated at node n. That is, the root operator $root(p)$ is evaluated at node n. Yet, child operators of $root(p)$, as part of the operator tree of projection p, may be evaluated at other nodes. As such, a vertex in the MuSE graph corresponds to a traditional operator placement for $root(p)$.

Node n is called *host* of $root(p)$ and, thus, it is also called host of p. Again, note that n being a host of projection p does not imply that the child operators of $root(p)$ are evaluated at n. A node hosting the root operator $root(q)$ of the original query q is called a *sink*. A directed edge $e = ((p, n), (p', n')) \in E$ describes that matches of p generated at n are sent to n' where they are used to generate matches of p'. The weight of such an edge, $c(e)$, models the rate with which matches of projection p hosted at n are sent to node n'. A vertex that has no incoming edges or outgoing edges represents a node that hosts a primitive operator (no incoming edges) or a root operator (no outgoing edges) of some query of the workload.

The above model incorporates arbitrary projections of a query for the placement of operators in an event-sourced network. However, there is another important difference compared to existing approaches for distributed CEP, in which an evaluation plan is fully characterized by an assignment of query operators to nodes. A MuSE graph, through its edges, specifies explicitly how matches of a projection are constructed based on matches generated at other nodes. In particular, they may be derived from matches of a *single* projection that are partitioned, as they are generated at *different* nodes.

The above observation leads to the set of event type bindings of a projection that are *covered* by a vertex in a MuSE graph: Matches of the respective event type bindings are generated by the vertex.

DEFINITION 4 (VERTEX COVER). *Let $G = (V, E, c)$ be a MuSE graph, $v = (p, n) \in V$ be a vertex, and $V' \subseteq V$ be the vertices that do not have an incoming edge and for which there exists a path to v in G. Then, the cover $\mathfrak{A}(v) \subseteq \mathfrak{E}(p)$ of v is the subset of the event type bindings of projection p, such that for each tuple (E, n) in the event type binding, there is a vertex $(\pi(p, \{E\}), n) \in V'$.*

EXAMPLE 6. *[Fig. 2](#) shows a MuSE graph for query q_1. It uses projections $p_3 = AND(C, L)$ and $p_2 = SEQ(L, F)$, the projection that corresponds to q_1, and those including solely primitive operators. Vertex v_1 defines that p_2 is hosted by node 1, while vertices v_2 and v_3 specify that nodes 1 and 2 host p_3. Nodes 1 and 2, which both host the projection corresponding to query q_1, are sinks and use matches of p_1 and p_2 to generate matches of q_1. Event type bindings of p_3 are partitioned in terms of their coverage by vertices. Vertex v_2 covers $\{[(C, 1), (L, 2)], [(C, 1), (L, 3)]\}$, whereas vertex v_3 covers $\{[(C, 2), (L, 2)], [(C, 2), (L, 3)]\}$.*

PROPERTY 1. *Let w be a predecessor vertex of v in a MuSE graph. For each event type binding covered by w, i.e. $\mathfrak{e}_w \in \mathfrak{A}(w)$, there exists at least one event type binding covered by v, i.e. $\mathfrak{e}_v \in \mathfrak{A}(v)$, such that $\mathfrak{e}_\mathfrak{w} \subseteq \mathfrak{e}_\mathfrak{v}$.*

PROPERTY 2. *For each sub-bag $\mathfrak{e}_\mathfrak{o} \subseteq \mathfrak{e}_\mathfrak{p}$ of an event type binding $\mathfrak{e}_\mathfrak{p} \in \mathfrak{A}(v)$ contained in the cover of v which is an event type binding of a projection o i.e. $\mathfrak{e}_\mathfrak{o} \in \mathfrak{E}(o)$ such that there exist predecessor vertices $V_o \subseteq V$ of v which generate matches of o, there exists at least one vertex $v_o \in V_o$ which covers $\mathfrak{e}_\mathfrak{o}$.*

EXAMPLE 7. *In [Fig. 2](#),the cover of each vertex is contained in the cover of each predecessor vertex. Moreover, v_4 has predecessors which generate the (non-primitive) projections p_2 and p_3. Each event type binding covered by v_4, contains sub-bags which are event type bindings of p_2 and p_3 and covered by predecessors of v_4, e.g. the event type binding $[(F, 1), (C, 1), (L, 2)]$ which is covered by v_4 contains the sub-bag $[(C, 1), (L, 2)]$ which is an event type binding of p_3 and covered by v_2 and the sub-bag $[(F, 1)(L, 2)]$ which is an event type binding of p_2 and covered by v_1.*

If v contains an event type binding $\mathfrak{e} \in \mathfrak{A}(v)$ in its cover, then v can generate *all* occurring matches of $\mathfrak{e}$. From Property 2 it follows that this is only possible if for all projections p_w and p_u hosted by predecessor vertices of v which share a primitive operator $\tilde{e}$, i.e. $\tilde{e} \in O_p^{p_u} \cap O_p^{p_w}$ the following holds: There exist predecessor vertices $w = (p_w, i)$ and $u = (p_u, j)$ of v such that each (primitive) vertex $v_{\tilde{e}} = (\tilde{e}, k) \in V$ (i.e. $v_{\tilde{e}}$ has no incoming edge) for which there exists a path to w there exists also a path to u and vice versa

EXAMPLE 8. *Let $v = (p, n)$ and u, w predecessor vertices of v which generate matches of the projections p_u and p_w accordingly..*

Let $root(p) = SEQ(A, B, C)$, $root(p_u) = SEQ(A, B)$ and $root(p_w) = SEQ(B, C)$, i.e. p_u and p_w share the primitive operator $\tilde{e}$ with $\tilde{e}.sem = B$. Moreover, let $(A, 2)$ and $(B, 1)$ be the (primitive) predecessor vertices of u and $(B, 2)$,$(C, 3)$ be the (primitive) predecessors vertices of w. The cover of v contains $\{[(A, 2)(B, 1)(C, 3)], [(A, 2)(B, 2)(C, 3)]\}$. The vertex v receives matches of the event type binding $[(A, 2)(B, 1)]$ from vertex u and matches of the event type binding $[(B, 2), (C, 3)]$ from vertex w. The matches of w contain only events of C generated at node 3 which occurred after a B event was generated at node 2. However, there might be matches of C generated at node 3 before a B was generated at node 2 but after a B was generated at node 1 - those C matches are not input to v. Thus, based on the matches received by w and u, v is not capable of generating all matches of the event type binding $[(A, 2)(B, 1)(C, 3)]$ (which is by definition contained in its cover). However, if $(B, 1)$ is also a predecessor of w, then w additionally covers $[(B, 1), (C, 3)]$. As such, it is possible to generate all matches of the event type binding $[(A, 2)(B, 1)(C, 3)]$ at v. To also generate all matches of the event type binding $\mathfrak{e} = [(A, 2)(B, 2)(C, 3)] \in \mathfrak{A}(v)$, u must additionally cover $[(A, 2)(B, 2)]$.

4.4 Cost Model

Next, we clarify the assignment of edge weights in a MuSE graph. Recall that the weight of an edge $e = ((p, n), (p', n'))$, $c(e)$, models the rate with which matches are exchanged. Hence, if $n = n'$, the projections p and p' are hosted by the same node and the weight shall be zero. Based thereon, the edges are partitioned into *local edges* with a weight of zero, and *network edges* with a non-zero weight, as determined by the following cost model.

We first capture the *output rate* of a query $q = (O, \lambda, P)$ of some workload and, hence, also of a projection, in a network $\Gamma = (N, f, r)$. The output rate $\hat{r}$ is defined recursively, along the operator hierarchy. For each primitive operator $o \in O_p$, the rate is derived from the rate with which events of the respective type are generated $r(o.sem)$, i.e., $\hat{r}(o) = r(o.sem)$. For a composite operator $o \in O_c$ with $\lambda(o) = \langle o_1, \ldots, o_k \rangle$, the output rate is computed from the output rates of child operators, $\hat{r}(o_1), \ldots, \hat{r}(o_k)$ and the operator semantics $o.sem$:

- If $o.sem = SEQ$, then $\hat{r}(o) = \prod_{1 \le i \le k} \hat{r}(o_i)$.
- If $o.sem = AND$, then $\hat{r}(o) = k \cdot \prod_{1 \le i \le k} \hat{r}(o_i)$.
- If $o.sem = NSEQ$ (by definition, $k = 3$), then $\hat{r}(o) = \hat{r}(o_1) \cdot \hat{r}(o_3)$, i.e., the bound ignores the negated child operator.

Recall that a query (projection) of a workload does not comprise operators of type OR, as those are split into separate queries. The output rate of a query (and, hence, a projection) is defined as $\hat{r}(q) = \sigma(q) \cdot \hat{r}(root(q))$, i.e., its selectivity times the rate of its root operator.

Using this cost model for output rates, we assign the edge weights of a MuSE graph $G = (V, E, c)$. To capture reuse of matches, $V_{v,n} \subseteq V$ represents operator placements at node n, which all have the same predecessor v in G, i.e., $V_{v,n} = \{v' = (p', n') \in V \mid n = n' \wedge (v, v') \in E\}$. In that case, matches of the operator of the preceding vertex in the MuSE graph are sent only once to the respective node.

Let $e = (v, v')$ be a network edge with $v = (p, n)$, $v' = (p', n')$, and $n \neq n'$. The weight $c(e)$ is given as the product of the output rate of projection p and the number of event type bindings covered by v divided by the number of vertices that denote operator placements sharing matches, i.e., $c(e) = \hat{r}(p) \cdot |\mathfrak{A}(v)|/|V_{v,n'}|$.

Aggregating the costs represented by the edge weights in a MuSE graph $G = (V, E, c)$, we derive the overall *network cost* of G, defined as $c(G) = \sum_{e \in E} c(e)$. It corresponds to the cost of the specific evaluation plan that is represented by the MuSE graph—the total rate with which matches are exchanged in the network.

Example 9. *Consider vertex v_1 in Fig. 2, which assigns projection $p_2 = SEQ(L, F)$ to node 1. Neglecting the selectivity, its output rate is derived from the rates of the children, i.e., $\hat{r}(C) \cdot \hat{r}(L)$. The respective matches stem from four projections, each including a single primitive operator, evaluated at nodes 1, 2, 3, and 4. Based thereon, the matches of p_2 at node 1 are constructed using four event type bindings, so that the weight of the edge (v_1, v_5) is $\hat{r}(C) \cdot \hat{r}(L) \cdot 4$.*

Our model enables the definition of an evaluation plan for a given event-sourced network and query workload, as defined in abstract terms in §3, as a MuSE graph. Moreover, the network cost of a MuSE graph denotes the cost of the evaluation plan, as it is the basis for the problem of efficient distributed query evaluation (Problem 1).

5 PROPERTIES OF MUSE GRAPHS

Having discussed the intuition of MuSE graphs for query evaluation, we turn to a formal investigation of their properties. We first introduce the auxiliary notion of a combination of query projections (§5.1). It is required to establish correctness of query evaluation with MuSE graphs (§5.2). Subsequently, we turn to the use of MuSE graphs for efficient distributed query evaluation as specified in Problem 1 and introduce the notion of a cost-optimal MuSE graph (§5.3).

5.1 Combinations of Projections

By incorporating arbitrary query projections, a MuSE graph enables a flexible definition of evaluation plans. In particular, there exist *multiple* possibilities for deriving the matches of a projection, i.e., of its root operator, from the matches of other projections. We capture one such possibility as a *combination* of projections and represent it as a graph that specifies the projections' hierarchy.

Definition 5 (Combination). *Let Q be a query workload. A combination of Q is a directed acyclic graph $\mathfrak{c} = (\mathfrak{V}, \beta)$ with the vertices being projections of the queries, $\mathfrak{V} \subseteq \bigcup_{q \in Q} \Pi(q)$, and $\beta : \mathfrak{V} \nrightarrow \mathfrak{V}^k$, $k \in \mathbb{N}$ and $k > 1$, that assigns to a projection, a set of predecessor projections in $\mathfrak{c}$.*

We write $\mathfrak{C}(Q)$ for the set of all combinations of a workload Q. For a combination to be useful for query evaluation, we define its *correctness*. In essence, it requires that for each projection p that includes more than a primitive operator, the projections that are predecessors in the combination suffice to generate matches of p (recall that $M_{\simeq}$ are all interleavings of a set of matches M, see §2.2).

Definition 6. *Let Q be a query workload evaluated in some event-sourced network. A combination $(\mathfrak{V}, \beta) \in \mathfrak{C}(Q)$ is correct, if for each query $q \in Q$ and each match $m \in M^q$, there exists a set of matches $M^{\beta} = \bigcup_{p \in \beta(q)} M^p$, such that $m \in M^{\beta}_{\simeq}$.*

Example 10. *Fig. 3 shows the combination used to generate matches of q_1 in Fig. 2. It contains projections $AND(C, L)$, and $SEQ(L, F)$, in addition to the projection corresponding to the query*

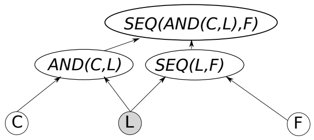


Figure 3: Combination for the query $SEQ(AND(C, L), F)$.

and those comprising a primitive operator. This combination is correct. It further highlights that the matches of a single projection, the marked one, may be incorporated in several other projections.

5.2 Correctness of MuSE Graphs

Using the notion of a combination, we characterize MuSE Graphs that are correct, i.e., that denote an evaluation plan that detects all matches of all queries of the workload, and only those. We separate the discussion into workloads that contain solely conjunctive queries built of composite operators of types *AND* and *SEQ*, and those that also include queries with negation, i.e., a *NSEQ* operator. Again, recall that queries of a workload cannot contain *OR* operators.

Conjunctive queries. We first characterize that a MuSE graph does generate only matches of queries of the workload at sinks using the property of well-formedness. Intuitively, a MuSE graph is well-formed, if it incorporates events of all types from all nodes and its structure is such that the matches generated at each sink can actually be generated based on the matches that are exchanged between the nodes. Formally, this property is captured as follows:

DEFINITION 7. *Let $G = (V, E, c)$ be a MuSE graph for a query workload Q. G is well-formed, if it holds for each $v = (p, n) \in V$:*

(i) for each query and each pair of a primitive operator and a node generating the respective event type, the graph contains a vertex, i.e., for all $q = (O, \lambda, P) \in Q$, $o \in O_p$, and $n \in N$, we have $o.sem \in f(n) \Rightarrow (o, n) \in V$; and

(ii) with $V' \subseteq V$ as the vertices for which there exists a path to v in G, including v, the combination $\mathfrak{c} = (\mathfrak{V}, \beta)$ that captures the structure of the projections in V', $\mathfrak{V} = \bigcup_{(p,n)\in V'} \{p\}$ and $\beta(p) = \{p' \in \mathfrak{V} \mid \exists\, v_1 = (p_1, n_1), v_2 = (p_2, n_2) \in V' : (v_1, v_2) \in E \wedge p_1 = p' \wedge p_2 = p\}$ for all $p \in \mathfrak{V}$, is correct, i.e., it holds that $c \in \mathfrak{C}(Q)$.

From the above definition, it follows directly that matches at sinks are indeed matches of one of the queries of the workload.

PROPERTY 3. *Each match generated at a sink in a well-formed MuSE graph for a query workload Q is a match of a query $q \in Q$.*

Next, we turn to completeness of query evaluation: All matches of a query are indeed constructed by a MuSE graph. However, for a single query, the matches may be partitioned over multiple sinks, which is characterized by the following definition.

DEFINITION 8. *Let $G = (V, E, c)$ be a MuSE graph for a query workload Q. G is complete, if for each query $q \in Q$, there exist a set of vertices $V' \subseteq V$ that jointly cover all event type bindings of q, i.e., $\bigcup_{v\in V'} \mathfrak{A}(v) = \mathfrak{E}(q)$.*

Based on the definition of event type bindings and their coverage by a vertex, completeness directly implies the following property.

PROPERTY 4. *A complete MuSE graph for the evaluation of a workload Q generates all matches of each query $q \in Q$.*

Algorithm 1: Optimal MuSE Graph Generation

input : A query $q = (O, \lambda, P)$; an event sourced network $\Gamma = (N, f, r)$.
output : An optimal MuSE graph G_{opt}.

1 $\Pi(q) \leftarrow \bigcup_{\mathcal{E}\in\mathcal{P}(O_p)} \pi(q, \mathcal{E})$ `// Set of all projections`
2 $\mathfrak{C}(\{q\}) \leftarrow$ *allCombinations*$(\Pi(q))$ `// Set of all combinations`
3 $C \leftarrow \emptyset$ `// Set of placed projection per combination`
4 **for** $\mathfrak{c} = (\mathfrak{V}, \beta) \in \mathfrak{C}(\{q\})$ **do** `// For each combination`
5   **if** $\mathfrak{c}$ *is correct according to Def. 6* **then**
    `// Determine all non-primitive projections`
6     $\mathfrak{V}' \leftarrow \{p = (O', \lambda', P') \in \mathfrak{V} \mid |O'| > 1\}$
    `// Derive all placements for these projections`
7   **for** $V' \in \left\{\bigcup_{p\in\mathfrak{V}'} \{(p, n_p)\} \mid n_p \in N\right\}$ **do** $C \leftarrow C \cup \{V'\}$
8 $\mathcal{G} \leftarrow \emptyset$
`// Construct MuSE graph from an assignment of all event type bindings to placed projections of some combination`
9 **for** $A \in \left\{\bigcup_{\mathfrak{c}\in\mathfrak{C}(q,\Gamma)} \{(\mathfrak{c}, V_\mathfrak{c})\} \mid V_\mathfrak{c} \in C\right\}$ **do** $\mathcal{G} \leftarrow \mathcal{G} \cup$ *constructMuSE*(A)
10 $G_{opt} \leftarrow \text{argmin}_{G\in\mathcal{G}}\, c(G)$ `// Select optimal MuSE graph`
11 **return** G_{opt}

We define a MuSE graph G for the evaluation of a workload Q to be correct, if it is both, well-formed and complete.

EXAMPLE 11. *The MuSE graph in Fig. 2 is correct: Each vertex includes a projection p of q_1, which induces a correct combination for p. Vertices v_4 and v_5 jointly cover all event type bindings of q_1.*

Queries including negation. The above characterization of correctness for MuSE graphs is lifted to query workloads that include queries with negation by constraining the set of projections. Intuitively, a projection that includes the event types used in the negated child operator of a *NSEQ* operator also needs to include the event types of its preceding and succeeding children. This way, the context for the verification of the absence of matches of the negated child operator is defined unambiguously in a projection. Formally, this is captured as a projection being negation-closed.

DEFINITION 9. *Let Q be a query workload and $q = (O, \lambda, P) \in Q$ be a query and $o \in O_c$ an operator with $o.sem = NSEQ$ and $\lambda(o) = \langle o_1, o_2, o_3 \rangle$. Let $p = (O', \lambda', P') \in \Pi(q)$ be a projection of q. Then, p is negation-closed, if $o_2 \in O'$ implies that $o_1, o_3 \in O'$.*

For a workload that includes queries with negation, a MuSE graph must be constructed solely based on projections that are negation-closed for all queries. Then, the above characterizations of well-formedness and completeness are directly applicable and ensure correctness of query evaluation.

5.3 Optimality of MuSE Graphs

A MuSE graph shall be constructed such that the overall rate with which events are exchanged in the network is minimal. If that is the case, the MuSE graph is said to be optimal.

DEFINITION 10. *From all correct MuSE graphs for a query workload Q, a graph G is optimal, if $c(G)$ is minimal.*

For a workload of a single query q, Alg. 1 computes an optimal MuSE graph. First, the set of projections $\Pi(q)$ for q is enumerated, based on all subsets of event types line 1). Second, the set of all combinations is constructed line 2). Here, function *allCombinations* considers each subset of $\Pi(q)$ as the set of projections $\mathfrak{V}$ for a combination; the predecessor function β is derived from the inclusion relation between the projections in $\mathfrak{V}$ (not shown). For each combination

$\mathfrak{c} = (\mathfrak{B}, \beta)$, we verify correctness (line 5). For correct combinations, we determine the non-primitive projections line 6) and construct all possible assignments of them to network nodes (line 7).

Next, we enumerate possible assignments of all event type bindings $\mathfrak{E}(q, \Gamma)$ of the query q in the network Γ to the placed projections of some combination. Each such assignment yields a correct MuSE graph (line 9). Per assignment, the MuSE graph is constructed as follows. For each event type binding, we generate a DAG: The vertices of this DAG are given by the placed projections and all tuples of the event type binding; its edges are derived from the respective combination. The union of all DAGs obtained for an assignment (one DAG per event type binding), yields the structure of a MuSE graph, which is then completed by computing the edge weights. From the set of resulting MuSE graphs $\mathcal{G}$, we return an optimal one (line 10).

We observe that the runtime complexity of Alg. 1 is given by

$$O\left(\left(2^{2^{|O_P|}} \cdot |N|^{2^{|O_P|}}\right)^{N^{|O_P|}}\right).$$

From that it follows, that even for small problem instances, i.e., a few small queries and a small network, there is a hyperexponential solution space. Hence, the construction of an optimal MuSE graph quickly becomes intractable, as underpinned by the following result.

THEOREM 1. *Given a query workload and an event-sourced network, the construction of an optimal MuSE graph is NP-hard.*

PROOF. Traditional operator placement (each operator is assigned to a single node) is an instance of the problem of assigning the modules of a task graph to the nodes of a distributed processor system [25]. If the task graph does not have a tree-structure and the network consists of more than four nodes, this problem is NP-complete [7]. For the construction of an optimal MuSE graph, the projections of correct combinations have to be placed. Combinations are DAGs and, in addition, to place each projection, not only a single node, but a set of nodes is considered. Hence, the construction of an optimal MuSE graph is at least as hard as the computation of a single-sink operator placement and, thereby, NP-hard. □

5.4 Minimality

A *correct* MuSE graph $G = (V, E, c)$ is *minimal*, if no network edge can be removed from E without violating its correctness.

LEMMA 1. *An optimal MuSE graph is minimal.*

PROOF. Let $G = (V, E, c)$ be a correct and optimal MuSE graph that is not minimal. Then, there exists a minimal MuSE graph $G' = (V', E', c')$, such that $E' \subset E$. Hence, in terms of network costs of G, it holds that $c(G) = c(G') + \sum_{e \in E \setminus E'} c(e)$. Since weights of network edges in a MuSE graph are greater than 0, it follows that $c(G') < (G)$ which contradicts the assumption that G is optimal. □

5.5 Equivalence of MuSE graphs

To define the equivalence of MuSE graphs, we first introduce the *collapsed normal form* of a MuSE graph.

DEFINITION 11. *Let $G = (V, E)$ be a correct MuSE graph for a query evaluated in some network and let V_p denote vertices in G which do not have an incoming edge. The graph G is in it collapsed normal form, if there exists no vertex $v = (p, n) \in V$ which has a predecessor vertex $w = (o, m) \in V \setminus V_p$ such that $n = m$.*

In a MuSE graph in its collapsed normal form, each vertex which does not represent the placement of a primitive operator, has at least one out-going network edge. The collapsed normal form of a MuSE graph is obtained by iteratively removing, for all nodes $v = (p, n)$, all predecessors $w = (o, m)$ for which $n = m$ and which do not have outgoing network edges. Incoming edges of removed predecessors are redirected to v. The described transformation does not change $\mathfrak{A}(v)$.

PROPERTY 5. *Two muse graphs are equivalent if they have the same collapsed normal form.*

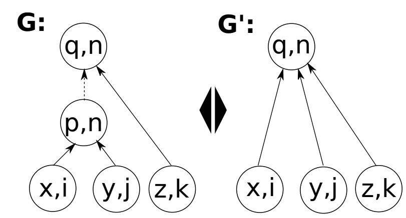


Figure 4: Collapsed normal form of a MuSE graph.

EXAMPLE 12. *Fig. 3shows two equivalent MuSE graphs. G' shows G in its collapsed normal form. G' can be obtained from G by removing the vertex (p, n) and redirecting the incoming edges of (p, n) to its successor vertex (q, n).*

6 CONSTRUCTION OF MUSE GRAPHS

To facilitate efficient construction of MuSE graphs, we present principles to prune the space of MuSE graphs for a single query that does not contain multiple primitive operators that reference the same event type (§6.1). Then, we present *aMuSE* and *aMuSE**, efficient algorithms to construct MuSE graphs for a single query (§6.2) as well as their extension for a multi-query setting.

6.1 Pruning Principles for MuSE Graphs

6.1.1 Beneficial Projections. Using a projection within a MuSE graph of a query shall reduce the rate with which events are exchanged, compared to a centralized evaluation of the query for which *all* primitive events need to be exchanged. For a MuSE graph $G = (V, E, c)$, containing a projection, i.e. having a vertex $v = (p, n) \in V$, will only be beneficial, if the rate of an outgoing edge of v is smaller than the sum of rates associated with v's incoming edges. If this is not the case, then sending matches of predecessors of the vertex v in G directly to all its successors would yield lower costs.

DEFINITION 12. *Let $G = (V, E, c)$ be a MuSE graph for a query q which contains a vertex $v = (p, n) \in V$. Moreover, let $\mathfrak{c} = (\mathfrak{B}, \beta) \in \mathfrak{C}(p)$ be the combination used to generate matches of p at n. Let $Pre(v, e) \subseteq V$ denote predecessors of v in G, that generate the predecessor projections $e \in \beta(p)$ of p. Having the vertex v in G is* beneficial*, if*

$$|\mathfrak{A}(v)| \cdot \hat{r}(p) \leq \sum_{e \in \beta(p)} \left(\hat{r}(e) \cdot \sum_{w \in Pre(v,e)} |\mathfrak{A}(w)| \right). \quad (1)$$

THEOREM 2. *An optimal MuSE graph for a single query q contains only beneficial vertices.*

PROOF. Let $G = (V, E, c)$ be an optimal MuSE Graph in its collapsed normal form which contains a vertex $v = (p, n) \in V$ such that v is not beneficial, i.e. Eq. 1 does not hold for v. As G is in its collapsed normal form, v has at least one outgoing network edge. Let $Succ(v)$ denote the set of successor vertices of v in G that are connected to v by a network edge. And let E_{succ} be the respective edges connecting v to the vertices in $Succ(v)$. The sum over the edge-weights of E_{succ} is given by

$$c(E_{succ}) = |Succ(v)| \cdot |\mathfrak{A}(v)| \cdot \hat{r}(p).$$

Let $G' = (V', E')$ be a transformed version of G which does not contain v, i.e. $V' = V \setminus v$. As such incoming and outgoing edges of v are not contained in E'. Moreover, let E' contain an edge between each predecessor in $Pre(v, e)$ of v (that hosts a predecessor projection e of p) and successor of v in $Succ(v)$. Let $E'_{dif} = \{(x, y) \mid x \in Pre(v, e) \wedge y \in Succ(v)\}$ denote the set of those edges. The sum over the edge-weights of E'_{dif} is given by

$$c(E'_{dif}) = |Succ(v)| \cdot \sum_{e \in \beta(p)} \left(\hat{r}(e) \cdot \sum_{w \in Pre(v,e)} |\mathfrak{A}(w)| \right)$$

.

The edge set of G' is given by $E' \subset E \cup E'_{dif} \setminus E_{succ}$. Thus, for the costs of G' it holds that $c(G') \leq c(G) - c(E_{succ}) + c(E'_{dif})$. As Eq. 1 does not hold for v, $c(E_{succ}) > c(E'_{dif})$. As a consequence, $c(G') < c(G)$ which contradicts G being optimal. □

EXAMPLE 13. *In Fig. 2 the vertex v_1 covers 4 event type bindings of the projection p_2, i.e. $\mathfrak{A}(v_1) = \{[(L,2),(F,1)], [(L,2),(F,4)], [(L,3),(F,1)], [(L,3),(F,4)]\}$. Thus, v_1 is a beneficial vertex if it holds that $4 \cdot \hat{r}(p_2) \leq 2 \cdot \hat{r}(L) + 2 \cdot \hat{r}(F)$*

To select projections that are suitable for the construction of MuSE graphs, we prune those projections can not yield beneficial vertices.

THEOREM 3. *Let p be a projection $p = (O, \lambda, P) \in \Pi(q)$ of a query q and $G = (V, E, c)$ a MuSE graph which contains a vertex $v = (p, n) \in V$. If for all possible combinations for p, i.e. for all $\mathfrak{c} = (\mathfrak{B}, \beta) \in \mathfrak{C}(p)$, it holds that*

$$\hat{r}(p) > \sum_{e \in \beta(p)} \hat{r}(e), \quad (2)$$

then G is is not optimal.

PROOF. Let $G = (V, E, c)$ be an optimal MuSE Graph which contains $v = (p, n)$ such that for p the inequation Eq. 2 holds for all possible combinations of p. Let $\mathfrak{c} = (\mathfrak{B}, \beta) \in \mathfrak{C}(p)$ be the combination used to generate matches of p at the vertex v. Let $\{x_{e_1}, \ldots, x_{e_n}\}$ denote the total number of event type bindings of each predecessor projection $\{e_1, \ldots, e_n\} = \beta(p)$ that is covered by the predecessors of v. As G is optimal, G contains only beneficial vertices (cf. Theorem 2). As such, it holds for v that

$$|\mathfrak{A}(v)| \cdot \hat{r}(p) \leq \sum_{e_i \in \beta(p)} x_{e_i} \cdot \hat{r}(e_i). \quad (3)$$

By dividing both sides of inequation by $|\mathfrak{A}(v)|$, we get

$$\hat{r}(p) \leq \sum_{i \in \{1,\ldots,n\}} \frac{x_{e_i}}{|\mathfrak{A}(v)|} \cdot \hat{r}(e_i). \quad (4)$$

Because of Property 1 it holds that $|\mathfrak{A}(v)| \geq x_{e_i}, \forall e_i \in \beta(p)$. As such $\frac{x_i}{|\mathfrak{A}(v)|} \leq 1$. Thus,

$$\hat{r}(p) \leq \sum_{e_i \in \beta(p)} \hat{r}(e_i). \quad (5)$$

which contradicts our assumption that Eq. 2 holds for p. Hence, G is not optimal. □

DEFINITION 13. *If for a projection p there exists at least one combination $\mathfrak{c} = (\mathfrak{B}, \beta) \in \mathfrak{C}(p)$, such that $\hat{r}(p) \leq \sum_{e \in \beta(p)} \hat{r}(e)$, p is a beneficial projection.*

To exploit this result in the construction of a MuSE graph, we do not explore all combinations of a projection p. Rather, we assess whether it is beneficial by considering solely the *primitive combination* $\mathfrak{c} = (\mathfrak{B}, \beta)$, for which the predecessor projections are given by p's primitive operators, i.e. $\beta(p) = O_p^p$. The reason being that a predecessor projection o in a combination of another projection p should reduce network costs compared to the case, in which events of o's primitive operators are directly sent to nodes hosting p. Put differently, we use the sum of the rates of the primitive operators of a projection as an upper bound for the costs of a suitable combination.

In the remainder, we prune the projections for which the inequation of Def. 13 does not hold for their respective primitive combination and consider the remaining projections as beneficial.

6.1.2 Restrictions on Combinations. To limit the search space, we restrict ourselves to MuSE graphs, in which each event type binding of the query is generated with the *same* combination. The idea here is that this restriction fosters sharing of partial matches. As an example, consider the query q having the primitive operators $O_p^q = \{x, y, z\}$. Let $\mathfrak{e}_1 = [(x,1)(y,2)(z,3)]$ and $\mathfrak{e}_2 = [(x,1)(y,2)(z,4)]$ be event type bindings of q evaluated in some network Γ, i.e. $\mathfrak{e}_1, \mathfrak{e}_2 \in \mathfrak{E}(\Gamma, q)$. Let $\mathfrak{c} = (\mathfrak{B}, \beta)$ denote a combination for the evaluation of q, and let $p \in \mathfrak{B}$ be a projection in c with $O_p^p = \{x, y\}$. Moreover, let v be a vertex at which matches of p are generated such that $\mathfrak{A}(v)$ contains $[(x,1)(y,2)]$. The matches generated at v can then be used to generate matches of both, $\mathfrak{e}_1$ and $\mathfrak{e}_2$, as $[(x,1)(y,2)] \subset \mathfrak{e}_1, \mathfrak{e}_2$. However, if matches of $\mathfrak{e}_2$ are generated with a *different* combination c' which does not contain p, the matches generated by v may only be used for generating matches of $\mathfrak{e}_1$. Not reusing the output of v potentially requires to send more events over the network than necessary. Yet, in some cases, it may be beneficial to allow the generation of event type bindings with different combinations. For example, matches of an event type binding $\mathfrak{e}_3 = [(x,1)(y,1)(z,1)]$ of q may be derived with a combination using solely primitive projections, since a placement of q at node 1 causes zero network costs. The construction of MuSE graphs which comprise different combinations for different event type bindings yields optimization potential and is subject to future work.

We denote the class of MuSE graphs in which matches of all event type bindings in $\mathfrak{E}(q)$ are generated with the same combinations as G^{uni}. A MuSE graph $G \in G^{uni}$ has exactly one underlying combination $c \in \mathfrak{C}(q)$ having the combinations for each of the projections used in G as sub-graphs. In the remainder, we only consider MuSE graphs in G^{uni}.

EXAMPLE 14. *The MuSE graph in Fig. 2 is in G^{uni}. The matches of all event type bindings of the projection p_2 are generated with the same combination (which comprises solely p_2's primitive projections); analogously for p_3. Also, the matches of all event type bindings of q_1 are generated with the same combination in which p_2 and p_3 are predecessors of $root(q_1)$.*

Analogously to the collapsed normal form, MuSE graphs in G^{uni} have an *unfolded normal form*.

DEFINITION 14. *Let $G = (V, E, c) \in G^{uni}$ be a correct MuSE graph having $\mathfrak{c} = (\mathfrak{P}, \beta)$ as underlying combination. Let V_p be the vertices of G which have no incoming edge. G is in its unfolded normal form, if there exists no vertex $v = (p, n) \in V \setminus V_p$ which has a predecessor vertex $w = (o, m)$ such that o is not a direct predecessor projection of p with respect to the combination $\mathfrak{c}$, i.e. $o \notin beta(p)$.*

EXAMPLE 15. *In Fig. 4, G' is a MuSE graph in G^{uni}. Let $\mathfrak{c} = (\mathfrak{P}, \beta)$ be G''s underlying combination such that the predecessors projections of q according to $\mathfrak{c}$ are given by $\beta(q) = \{p, z\}$ and the predecessor projections of p are given by $\beta(p) = \{x, y\}$. G shows G' in its unfolded normal form.*

A graph $G \in G^{uni}$ in its unfolded normal form contains at least one vertex for each of the projections in its underlying combination. Moreover, if G is correct, then for each projection p in its underlying combination, all event type bindings $\mathfrak{E}(p)$ are covered by vertices in G.

THEOREM 4. *Let $G = (V, E, c) \in G^{uni}$ be a MuSE graph for the evaluation of a query q and let $\mathfrak{c} = (\mathfrak{P}, \beta) \in \mathfrak{C}(q)$ be the underlying combination of G. If G is correct, then for each projection $p \in \mathfrak{P}$ it holds that matches of all event type bindings $\mathfrak{E}(p)$ of p are generated at the vertices of G.*

PROOF. Let $G = (V, E, c) \in G^{uni}$ be a correct MuSE graph in its unfolded normal form for q evaluated in $\Gamma = (N, f, r)$ having $\mathfrak{c} = (\mathfrak{P}, \beta) \in \mathfrak{C}(q)$ as its underlying combination. Let $\mathfrak{e} \in \mathfrak{E}(p)$ be an event type binding of a projection p which is *not* generated at a vertex in G. As G is correct, matches of all event type binding $\mathfrak{E}(q)$ are generated at the sinks of G. For each projection p of a query q, it holds that for each event type binding $\mathfrak{e}_\mathfrak{p} \in \mathfrak{E}(p)$ of p there exists an event type binding $\mathfrak{e}_\mathfrak{q} \in \mathfrak{E}(q)$ of q, such that $\mathfrak{e}_\mathfrak{p} \subset \mathfrak{e}_\mathfrak{q}$. As such there exists a sink vertex $s \in V$ which generates matches of event type bindings of q that contain the event type binding $\mathfrak{e} \in \mathfrak{E}(p)$ as sub-bag. As $G \in G^{uni}$, each event type binding of q is generated with the same combination. Thus, the sink s has at least one predecessor vertex which generates matches of p. Because of Property 2, e must be covered by at least one predecessor of s which generates matches of p. This contradicts our assumption that matches of e are not generated by a vertex in G. □

Redundant Combinations. We further restrict the combinations considered for the construction of a MuSE graph using the notion of a *redundant* combination:

DEFINITION 15. *A combination $\mathfrak{c} = (\mathfrak{P}, \beta) \in \mathfrak{C}(q)$ is* redundant *if there exists a projection $p \in \mathfrak{P}$ such that p has a predecessor projection $\tilde{e} \in \beta(p)$ for which $O_p^{\tilde{e}} \subseteq \bigcup_{e \in \beta(p) \setminus \tilde{e}} O_p^e$.*

THEOREM 5. *The underlying combination of an optimal MuSE graph $G \in G_{uni}$ for the evaluation of a query q is not redundant.*

PROOF. Let $G = (V, E, c) \in G_{uni}$ be an optimal MuSE graph in its collapsed normal form such that its underlying combination $\mathfrak{c} = (\mathfrak{P}, \beta)$ is redundant. Let $v = (p, n)$ be a vertex at which matches of p are generated such that p has an predecessor projection $\tilde{e}$ for which $O_p^{\tilde{e}} \subseteq \bigcup_{e \in \beta(p) \setminus \tilde{e}} O_p^e$. As such, for each primitive operator in $o \in O_p^{\tilde{e}}$ there exists at least one other predecessor projection $e \in \beta(p) \setminus \tilde{e}$ such that e and $\tilde{e}$ share the primitive operator o. Thus, for each shared primitive o there exist at least two predecessor vertices w, u of v such that w and u have the same primitive predecessor vertex which describes the generation of matches of the event type $o.sem$ at the same node (cf. Property 2). Hence, removing each vertex $u = (\tilde{e}, m)$ (which generate generate $\tilde{e}$) from the predecessors of v does not change the cover $\mathfrak{A}(v)$ and thus, does not violate G's correctness. As G is in its collapsed normal form, (u, v) is a network edge. Hence, G is not minimal and thus, not optimal (cf. Lemma 1). □

EXAMPLE 16. *The underlying combination of the MuSE graph in Fig. 2 is not redundant. For the illustrated combinations of p_2, p_3 and q it holds that no predecessor projection can be removed without violating the correctness of the MuSE graph.*

For each non-redundant combination $\mathfrak{c} = (\mathfrak{P}, \beta)$ of a projection p the number of predecessor projections, i.e. $|\beta(p)|$ is limited by $|O_p^p|$. In the remainder, we prune redundant combinations from the search space.

6.1.3 Single-Sink and Multi-Sink Placements.

For a MuSE graph $G = (V, E, c) \in G_{uni}$ matches of all event type bindings of each projection must be generated (cf. Theorem 4). As such, it must be guaranteed, that the nodes which host a projection p together cover $\mathfrak{E}(p)$. We denote the subset of nodes $V_p \in V$ hosting a projection p as *placement* of p. A *single-sink placement* $v = (p, n) \in V$ assigns the generation of matches of all event type bindings of the projection p to one single node. As a consequence, for v to be a correct single-sink placement it must hold that $\mathfrak{A}(v) = \mathfrak{E}(p)$, i.e. the placement v covers all event type bindings of p. For a *multi-sink placement* V_p of p the vertices in V_p jointly cover $\mathfrak{E}(\mathfrak{p})$. Thus, for V_p to be a correct placement of p, it must hold that $\cup_{v_p \in V_p} \mathfrak{A}(v_p) = \mathfrak{E}(p)$.

The *placement costs* for a placement V_p are given by the sum of the edge weights of incoming edges of the vertices in V_p. As such, placing p at nodes that generate predecessor projections of p favors low placement costs as in this case some of V_p's incoming edges are local edges which have edge weight 0. We denote the placement costs of V_p by $c_p(V_p)$.

EXAMPLE 17. *In Fig. 2 the vertices $V_{p_3} = \{v_2, v_3\}$ illustrate a multi-sink placement for p_3. Together they cover $\mathfrak{E}(p_3)$. The placement costs of V_{p_3} are given by $c_p(V_{p_3}) = 3 \cdot \hat{r}(L)$. In contrast $V_{p_2} = \{v_1\}$ is a single-sink placement for the projection p_2 and thus*

$\mathfrak{A}(v_1) = \mathfrak{E}(p_2)$. *The placement costs of* V_{p_2} *are given by* $c_p(V_{p_2}) = 2 \cdot \hat{r}(L) + 1 \cdot \hat{r}(F)$.

The placement costs for the possible placements of a projection p depend on the combination $\mathfrak{c} = (\mathfrak{V}, \beta)$ used to generate matches of p and on the placement of the predecessor projections $\beta(p)$ of p. Given, that all predecessor projections of p are already placed, we denote the placement costs for a placement V_p for p as *minimal* if there is not other placement yielding lower placement costs.

Beneficial Multi-Sink Placements. Let $G \in G^{uni}$ be a MuSE graph that contains the projection p and let $c = (\mathfrak{V}, \beta)$ be a non-redundant combination with which matches of p are generated in G. Given that all predecessor projections $e \in \beta(p)$ of p are already placed, we can easily compute a single-sink placement for p having *minimal* placement costs by iterating over all nodes that generate a predecessor projection $e \in \beta(p)$ and comparing the respective costs of incoming edge weights. This is not the case for multi-sink placements due to their exponential search space. To restrict the search space, we introduce properties for a multi-sink placement of a projection p that must hold in order for it to yield lower placement costs than a respective minimal single-sink placement for p.

DEFINITION 16. *Let* $G = (V, E, c) \in G^{uni}$ *be a MuSE graph that contains the projection p and let* $\mathfrak{c} = (\mathfrak{V}, \beta)$ *be the combination with which matches of p are generated in G. Let* v_s *be a minimal single-sink placement for p. A multi-sink placement* V_m *for p is* beneficial *if it yields lower placement costs than* v_s*, i.e.* $p_c(V_m) \leq p_c(v_s)$.

THEOREM 6. *Let* v_s *be a minimal single-sink placement of p and* V_m *be a multi-sink placement of p. If the multi-sink placement* V_m *is beneficial then the following holds for* V_m*:*

(1) the placement V_m *contains two vertices* $v = (p, n), w = (p, m)$ *with* $n \neq m$ *and the nodes* n, m *also host predecessor projections* $e, e' \in \beta(p)$ *of p, i.e.* $x = (e, n), y = (e', m) \in V$ *and neither x nor y have an outgoing edge to another vertex in* V_m

(2) $e = e'$

(3) $min(|\mathfrak{A}(x)|, |\mathfrak{A}(y)|) \cdot \hat{r}(e) > \sum_{\tilde{e} \in \beta(p) \setminus e} \hat{r}(\tilde{e}) \cdot \mathfrak{E}(\tilde{e})$.

PROOF. We prove Theorem 6 by showing that there exists no beneficial multi-sink placement for which (1)-(3) does not hold.

Let V'_m be a beneficial multi-sink placement for which (1) does not hold which means that one of the following three cases apply for V'_m:

case 1: there exists exactly one vertex $v = (p, n) \in V'_m$ such that n also hosts a predecessor projection e of p, i.e. $x = (e, n) \in V$ with x having no outgoing edge to another vertex in V_m.

In this case, there is no other node than n which hosts p *and* does not send locally generated matches of predecessor projections of p over the network. As such the placement costs are given by the sum of the rates of all event type bindings of all predecessor projections of p minus the fraction of event type bindings of predecessor projections that are generated at n. Let $Pre_{local}(v, e) \subset V$ denote predecessors of v that are hosted at n and generate a predecessor projection $e \in \beta(p)$. Then, for the costs for the placement V'_m it holds $p_c(V'_m) \geq \sum_{e \in \beta(p)} \hat{r}(e) \cdot (\mathfrak{E}(e) - \sum_{w \in Pre_{local}(v,e)} \mathfrak{A}(w))$. In contrast, a single sink placement of p at v has exactly $\sum_{e \in \beta(p)} \hat{r}(e) \cdot (\mathfrak{E}(e) - \sum_{w \in Pre_{local}(v,e)} \mathfrak{A}(w))$ as placement costs. As a consequence, V'_m can not yield lower placement costs than the single-sink placement $\{v\}$.

case 2: for each vertex $(p, n) \in V'_m$ it does not hold that n also hosts a predecessor projection $e \in \beta(p)$. In this case, no vertex in V'_m has an incoming *local* edge, i.e. all event type bindings of all predecessor projections of p have to be sent at least once over the network. As such the placement costs of V'_m are given by $p_c(V'_m) \geq \sum_{e \in \beta(p)} \hat{r}(e) \cdot \mathfrak{E}(e)$. As shown for *case 1*, a single-sink placement at one node hosting at least one predecessor projection of p yields lower placement costs.

case 3: there exists at least one vertex $v = (p, n) \in V'_m$ such that the same node hosts at least one predecessor projection $e \in \beta(p)$ of p, i.e. $x = (e, n) \in V$. However x has an outgoing edge to other vertices in V'_m. As in *case 2*, all event type bindings of all predecessor projection are sent at least once over the network. Again, a single sink placement at a node hosting at least one predecessor projection of p yields lower placement costs.

Let V'_m be a beneficial multi-sink placement for which (2) does not hold. As V'_m beneficial (1) holds, which means that there exist two vertices $v = (p, n), w = (p, m) \in V_m$ with $n \neq m$ such that n, m also host predecessor projections of p, i.e. $x = (e, n), y = (e', m) \in V$. Moreover, x, y do not have an outgoing edge to other vertices of V'_m. However, $e \neq e'$. As V'_m is a correct placement, matches of all event type bindings of p are generated at vertices of the placement V'_m. As the vertex x does not have an outgoing edge to another vertex of the placementV_p, matches of all event type bindings of the projection p containing $\mathfrak{A}(x)$ have to generated by the vertex v. Analogously, w must generate matches of all event type bindings of p containing $\mathfrak{A}(y)$. If the predecessor projection e and e' are not of the same type, there exist at least one event type binding $\mathfrak{e} \in \mathfrak{E}(p)$ of p containing as sub-bags event type bindings contained in $\mathfrak{A}(x)$ *and* $\mathfrak{A}(y)$, i.e. $\mathfrak{e}_x, \mathfrak{e}_y \subset \mathfrak{e}$ with $\mathfrak{e}_x \in \mathfrak{A}(x)$ and $\mathfrak{e}_y \in \mathfrak{A}(y)$. Let matches of $\mathfrak{e}$ be generated by a vertex $u = (p, l) \in V'_m$. To this end, both vertices, x and y, must have an outgoing edge to u. As $n \neq m$ this contradicts (1).

Let V'_m be a beneficial multi-sink placement for which (3) does not hold. However, as V'_m is beneficial (1) and (2) hold for V'_m. We can construct a single-sink placement $v_s = (p, n)$ for p having less placement costs than V'_m: Let w.l.o.g $min(|(\mathfrak{A})(x)|, |(\mathfrak{A})(y)|) = |\mathfrak{A}(y)|$, i.e. the vertex $x = (e, n)$ covers the bigger fraction of the event type bindings of e. Moreover, let node n also generate matches of *all* event type bindings of *all* other predecessor projections $\beta(p) \setminus e$ of p. Furthermore, we assume that $\mathfrak{A}(x) \cup \mathfrak{A}(y) = \mathfrak{E}(e)$, i.e. that there is not event type binding of e that is not covered by the vertices y and x. Making this assumption makes the argument a bit clearer although it also holds without the assumption. In order for $v_s = (p, n)$ to be a correct single-sink placement, i.e. v_s generates matches of all event type bindings of p, n only requires matches of the event type bindings of e that are generated at y. As such $v_p(v_s) = \mathfrak{A}(y) \cdot \hat{r}(e)$. In contrast, for V'_m to be a correct multi-sink placement, $v = (p, n)$ and $w = (p, m)$ require matches of all event type bindings of the predecessor projections of p that are not of type e ($\beta(p) \setminus e$). While those are generated at n, they may have to be sent over the network to m. In this case, $v_p(V'_m) = \sum_{\tilde{e} \in \beta(p) \setminus e} \hat{r}(\tilde{e}) \cdot \mathfrak{E}(\tilde{e})$. As (3) does not hold, it follows that $v_p(V'_m) \geq v_p(v_s)$ which contradicts V'_m being beneficial. ◻

Minimal Multi-Sink Placements. Let V_p be a minimal placement for p. If V_p is a multi-sink placement, then V_p has to be beneficial – otherwise there exists a single-sink placement for p having lower placement costs. A beneficial multi-sink placement is minimal if there exists no other multi-sink placement having lower placement costs. In the following, we introduce more properties to restrict beneficial multi-sink placement to those that yield minimal placement costs.

THEOREM 7. *Let V_{Min} be the set of minimal placements for a projection p. Moreover, let V_{Min} contain only multi-sink placements. As such each placement in V_{Min} is beneficial. There exists a placement $V_m \in V_{Min}$ for which the following properties hold:*

(1) for each vertex $v_m = (p,l) \in V_m$ it holds that l hosts at least predecessor projection $\tilde{e} \in \beta(p)$
(2) for each vertex $v_m = (p,l) \in V_m$ it holds that at least one predecessor projection $\tilde{e} \in \beta(p)$ which is generated at l is not sent to another node r such that $(p,r) \in V_m$.

PROOF. Let V'_m be a minimal (multi-sink) placement for p for which (1) does not hold, i.e. there exists a vertex $u = (p,l) \in V'_m$ such that the node l does not host a predecessor projection $\tilde{e} \in \beta(p)$. As such, to generate matches of p, the node l requires matches from at least one event type binding from each predecessor projection $\tilde{e} \in \beta(p)$. As such, the sum of the incoming edge-weights of u are $\geq \sum_{\tilde{e} \in \beta(p)} \hat{r}(\tilde{e})$. As V'_m is beneficial, there exist two vertices $v = (p,n), w = (p,m) \in V'_m$ which generate matches of all event type bindings of p which contain their locally generated matches of the predecessor projection $e \in \beta(p)$. To this end, n and m receive all matches of the event type bindings of the predecessor projection $\beta(p) \setminus e$ (that are not e). Let $V''_m = V'_m \setminus u$ be a transformation of V'_m which generates the event type bindings of p that were generated at vertex u in V'_m instead at vertex v. To this end, v only requires the matches of the event type bindings of the predecessor projections e that are used to generate matches at u in V'_m. As such V''_m reduces the placement costs of V'_m by at least $\sum_{\tilde{e} \in \beta(p) \setminus e} \hat{r}(\tilde{e})$ which contradicts V'_m being minimal.

Let V'_m be a minimal multi-sink placement for p for which (2) does not hold. As such there exists a vertex $u = (p,l) \in V_m$ such that all matches of predecessor projections $\tilde{e} \in \beta(p)$ that are locally generated at l are also input to other vertices $(p,r) \in V_m$ of the placement. Let e again be the predecessor projection of $\beta(p)$ that is local input to the vertices v and w referenced in Theorem 6. If at node l matches of event type bindings of $\tilde{e}$ are generated, they are sent over the network to other vertices of the placement V'_m. Let $V''_m = V'_m \setminus u$ be a transformation of V'_m which generates the event type bindings of p generated at vertex u in V'_m instead at vertex v. To this end, v only requires the matches of the event type bindings of the predecessor projection e that were used to generate matches at u in V'_m. As such, having V''_m instead of V'_m leads to the potentially additional edge $((e,l),v)$ However, if v gets the event type bindings of e that are generated at u, v is capable of generating matches of *all* event type bindings of p that contain the event type bindings of e from u. As such all other outgoing edges from (e,u) to other vertices of V'_m (from which there was at least one) can be removed, without violating the correctness of the placement V''_m. The number of outgoing network edges from (e,l) is with respect to our transformation V''_m less or equal than in V'_m. Those event type bindings of e, that are not generated by l can be sent over the network to v inflicting the same costs as sending them to u. As with removing u from V'_m potential incoming network edges from u are removed as well, it holds that $c_p(V''_p) \leq c_p(V'_p)$. □

COROLLARY 1. *Let V_{Min} be a set of minimal multi-sink placements for p There exists a placement $V_m \in V_{Min}$ for which holds: There exists one predecessor projection $e \in \beta(p)$ such that for all vertices $u = (p,l) \in V_m$ the respective node l also generates matches of event type bindings of e.*

PROOF. From V_m being minimal it follows that for each vertex $u = (p,l) \in V_m$ of the placement that the respective node l also generates a predecessor projection $\tilde{e} \in \beta(p)$ and that there exists at least one predecessor projection generated at each node of the placement which is not sent over the network. From V_m being beneficial, it follows that there exist two vertices $v = (p,n)$ and $w = (p,m)$, such that the respective nodes n and m also generate the *same* predecessor projection $e \in \beta(p)$. Moreover, the matches of e which are generated at n, m are not input to other vertices of the placement. However, to generate all matches of the event type bindings of p with their matches of e the nodes n, m require all matches of all other event type bindings of the predecessor projections of p besides e. As such the only predecessor projection $\tilde{e} \in \beta(p)$ locally generated at a node which hosts p which does not have to be sent over the network in order for the placement to be correct, is e. □

We denote the predecessor projections e as *partitioning input* as the covers of the vertices in the placement V_p partition the generation of the event type bindings of p with respect to their locally generated event type bindings of e.

EXAMPLE 18. *In Fig. 2 the placement $V_{p_3} = \{v_2, v_3\}$ of p_3 has the primitive predecessor projection C as partitioning input. The vertex v_2 covers the event type bindings of p_3 containing $(C,1)$ and v_3 covers the event type bindings of p_3 containing $(C,2)$. The partitioning input of the placement $V_q = \{v_4, v_5\}$ for the query q is given by the predecessor projection $AND(C,L)$.*

If

$$\hat{r}(e_{part}) \geq \sum_{e \in \beta(p) \setminus e_{part}} \hat{r}(e) \cdot |\mathfrak{E}(e)|. \tag{6}$$

a minimal multi-sink placement V_p for p contains a vertex for each node generating matches of event type bindings of e_{part}. Moreover, in this case matches of the partitioning input e_{part} are *never* sent over the network to generate matches of p. We denote minimal multi-sink placements having a partitioning input as *partitioning multi-sink placements*. In the remainder, we only consider partitioning multi-sink placements for which Eq. 6 holds as well as single-sink placements for the construction of MuSE graphs.

6.2 aMuSE and aMuSE*

Based on the pruning principles discussed, we propose *aMuSE*. The algorithm takes a query of workload $q \in Q$ and an event-sourced network Γ as input and returns a MuSE graph. The algorithm proceeds in two phases: (1) an enumeration phase that outlines the space of promising projections and combinations; and (2) a construction phase that explores for each projection several combinations and placements at network nodes.

aMuSE – Enumeration Phase. Alg. 2 outlines the first phase of *aMuSE*. As a first step, the set of beneficial projections Π_{ben} of q is generated by checking the inequation of Def. 13 for the primitive combination of each possible projection of q (lines 1-4, lines 11-13). Then, for each beneficial projection p the (relevant) subset of $\Pi^p_{ben} \subseteq \Pi_{ben}$ is selected which contains only projections that can be part of a correct combination of p (lines 5-6). Combinations are identified by their contained projections (line 7), the predecessor relation β follows from the inclusion relation between the projection (not illustrated). Based on Π^p_{ben} aMuSE enumerates all possible combinations for which the number of contained projections is $|O^p_p|$, pruning those that are not correct or *redundant* (lines 7-9, lines 14-20). The output of this phase is the set of beneficial projections as well as combinations thereof (line 10). The worst-case time complexity of this phase is given by $O(2^{|O^p_p|} \cdot 2^{2^{|O^p_p|}})$. However, by considering only *beneficial* projections and *non-redundant* combinations, the search space is significantly reduced and, as shown in §7, the enumeration is feasible for realistic problem sizes.

aMuSE – Construction Phase. Alg. 3 outlines the construction phase. Using the set of beneficial projections and combinations thereof, aMuSE constructs MuSE graphs for the query q (evaluated in G) in a bottom-up fashion using a dynamic programming approach. To this end, for each projection several MuSE Graphs are constructed. A MuSE graph for a projection p is composed of MuSE Graphs of predecessor projections (with respect to some combination of p), which were constructed in earlier iterations. The MuSE graphs for the primitive projections are given by sets of vertices denoting the generation of the respective event type at each each node capable of generating the event type (line 1). The (non-primitive) projections are sorted based on their number of primitive operators (line 2). Starting with the projections having the fewest of them, for each projection p and combination c of p, aMuSE constructs several MuSE graphs having different placements of the projection p as sink(s) (lines 3-4).

Placement Options. aMuSE considers only local placements for each tuple (p, c), i.e. placements of p at nodes that generate matches of at least one predecessor projection with respect to the combination c, as those yield local edges.

If all predecessor projections of p are given by primitive projections O^p_p of p, then aMuSE constructs at most $|O^p_p|$ MuSE graphs for p, one for each predecessor projection e, having a placement of p at a node or set of nodes generating the respective event type $e.sem$ - we denote e as *placement option* of p. If a predecessor projection e is a non-primitive projection, then, in an earlier iteration, at most $|O^e_p|$ MuSE graphs, i.e. one for each of the placement options O^e_p, were constructed for e (having a placement of e as sink). In order to place p with respect to the non-primitive predecessor e, each placement option of e is considered as a placement option for the placement of p.

Multi-Sink Optimization. If there exists a partitioning multi-sink placement for a projection p and combination c, then MuSE graphs are only constructed for placement options of *one* predecessor projection of p, i.e. the partitioning input of p (lines 5-7). The reason here is, that due to Theorem 7, we know that partitioning multi-sink placements yield lower costs than any single-sink placement for the projection p evaluated with the combination c. The function `getMSP` returns the respective partitioning input *partInput* if a partitioning multi-sink placement exists (lines 18-22). This is done by checking for each predecessor projection according to the combination c if Eq. 6 holds. If there exists no partitioning multi-sink placement for (p, c), then for *each* predecessor projection and each of the respective placement options, MuSE graphs are constructed (lines 11-13). Different predecessor projections within the same combination (or across multiple combination) may yield the same placement option. aMuSe maintains only the MuSE graph having the least cost, among all MuSE graphs constructed for the same projection p and placement option PO (lines 9-10, 15-16).

Choice of Placement. The actual placement for (p, c), i.e. the nodes which should host the projection p given a predecessor projection and placement option, is determined as follows: If (p, c) has a partitioning multi-sink placement, then the nodes for the placement are given by all nodes generating the placement option (line 8). Otherwise, for each predecessor projection e of p and each placement option PO in O^e_p, a single sink placement is chosen from the nodes generating e according to PO (line 14): Let $G[e][PO]$ denote the MuSE graph which was constructed for the projection e having a placement of e at nodes generating the placement option PO as sink(s). As we want to place p at a node generating matches of e, a node n of one sink vertex (e, n) of $G[e][PO]$ is chosen (as shown in function `getSSP`, lines 23-26). If $G[e][PO]$ has a single-sink placement (e, n) of e as sink, then n is the only node generating matches of e (for the considered placement option). Thus, we have to choose n as node for the placement for p. Otherwise, i.e. if $G[e][PO]$ has a multi-sink placement, we have to choose one of the nodes involved in the multi-sink placement, for the placement of p. In `getSSP` an arbitrary node is chosen, however, in our implementation nodes that generate many primitive operators of p are preferred to favor local edges.

Construction of Graphs. Given a projection p, combination c, predecessor projection *predProj*, placement option *placeOption* and set of nodes $Nodes_p$ for the placement, a MuSE graph G is constructed using the MuSE graphs of the predecessor projections generated in earlier iterations as follows (lines 9, 15, `ConstructSubgraph` lines 27-43): Let again, $G[e][PO]$ denote the MuSE graph of a predecessor projection e generated for a placement option PO. The sink vertices of G are constructed by creating a vertex for each tuple of projection p and nodes contained in $Nodes_p$ (lines 29-30). Then, an edge from the sinks of the MuSE graph $G[predProjection][PO]$ to the sinks in G is added to the edge set of E as follows: If G has only one sink $s = (p, n_p)$, i.e. s is a single sink placement of p, then for each sink of $G[predecessorProj][PO]$ an edge to $s = (p, n_p)$ is added to E (line 32). If G has a multi-sink placement of p, then for each pair of sinks of G and $G[predecessorProj][PO]$ which denote placements at the same node an edge is added to E (line 33). In a next step, a sub-graph for each remaining predecessor projection of p according to the combination c is added to G (line 34). For each predecessor projection e, the MuSE Graphs of all placement options $placementOption_e$ are considered for being a sub-graph of G (line 36). The graph G_e (of the predecessor e) that is actually added to G, is chosen such that the cost of the resulting graph - constructed by

Algorithm 2: Projection and Combination Enumeration

input : A query $q = (O, \lambda, P)$; an event sourced network $\Gamma = (N, f, r)$
output : A set of beneficial projections Π_{ben} and set of combinations $\mathfrak{C}$ for the projections in Π_{ben}

$\Pi_{ben}, \mathfrak{C} \leftarrow \emptyset$
for $\mathcal{E} \in \mathcal{P}(O_p^q)$ **do** // For each subset of primitive operators
  if `isBeneficial`$(\pi(q, \mathcal{E}))$ **then** // If beneficial add to Π_{ben}
    $\Pi_{ben} \leftarrow \Pi_{ben} \cup \{\pi(q, \mathcal{E})\}$
for $p \in \Pi_{ben}$ **do** // For each beneficial projection
  $\Pi_{ben}^{p} \leftarrow \Pi_{ben} \cap \Pi(p)$ // Choose for p subset of Π_{ben} that contains projections of p
  for $\mathfrak{c} \in \mathcal{P}(\Pi_{ben}^{p})$ **do** // For each possible combination of p
    if `notRedundantAndCorrect`$(p, \mathfrak{c})$ **then**
      $\mathfrak{C}(p) \leftarrow \mathfrak{C}(p) \cup \mathfrak{c}$
return C, Π_{ben}
function `isBeneficial` $(projection)$
  if $\hat{r}(projection) \leq \sum_{e \in O_p^{projection}} \hat{r}(e)$ **then return** $True$ // Check if *Definition 13* holds for primitive combination of p
  else return $False$
function `notRedundantAndCorrect` $(projection, combination)$
  $O_{prim} \leftarrow \emptyset$
  for $e \in \beta(projection)$ **do**
    if $(O_p^e \not\subseteq O_{prim})$ **then** $O_{prim} \leftarrow O_{prim} \cup O_p^e$ // Exclude redundant combinations
    else return $False$
  if $O_{prim} = O_p^q$ **then return** $True$ // Generate correct combinations
  else return $False$

connecting all sinks of G_e to the sinks of G - is minimal (lines 36-43). The resulting MuSE graph of this step is a correct MuSE graph for the projection p placed at nodes generating the placement option *PO* which also generate predecessor projection *predProjection*.

MuSE Graph for q. With the query being treated as the other projections, in the last iteration MuSE graphs for the query according to each placement option are constructed. The one yielding minimal costs constitutes the output of the algorithm (line 17). The complexity of Alg. 3 is given as $O(|\Pi_{ben}| \cdot |\mathfrak{C}(q)| \cdot |O_p^q|^4)$.

aMuSE*. We propose aMuSE*, a variation of aMuSe that further restricts the set of projections to be used to construct MuSE graphs. aMuSE* only considers a projection p, for which there exists at least one primitive operator in $e \in O_p^p$, such that $\hat{r}(e) \geq \hat{r}(p) \cdot |\mathfrak{C}(p)|$. Moreover, aMuSE* considers for each combination of $\mathfrak{c} = (\mathfrak{B}, \beta) \in \mathfrak{C}(p)$, only predecessor projections $e \in \beta(p)$ for which $\hat{r}(e) \geq \hat{r}(p) \cdot |\mathfrak{C}(p)|$ for local placements which limits the search space for single-sink placements. As a consequence, aMuSE* considers less projections, combinations, and placements thereof. The intuition behind the restriction is that when a projection p is placed at a predecessor e, at least some network costs should be saved by the placement compared to sending all inputs of p directly to a placement of a projection p', which has p as input.

Multi-Query Extension. For the computation of a MuSE graph for a query workload Q, we sequentially apply aMuSE for each query $q \in Q$ of the workload. After a MuSE graph for a query was determined, we make the following updates to support the reuse of projections and event types that are already disseminated in the network when matching another query. Let $G = (V, E)$ be the result of aMuSE for q and a network $\Gamma = (N, f, r)$.

- If $v = (p, n) \in V$, then for another query $q' \in Q$ with $p \in \Pi(q')$, a placement of p at n is assigned a placement cost of zero when computing a MuSE graph for q'.
- For each node $n \in N$, we check which matches of predecessors of projections hosted at n. Those can be reused to match other projections and, hence, affect the computation of placement costs for placing projections at n for another query.

7 EXPERIMENTAL EVALUATION

We evaluated the proposed concepts with the setup detailed in §7.1. Then, §7.2 presents a simulation study that compares the efficiency of query evaluation with MuSE graphs against a centralized baseline strategy and one that uses traditional techniques to find an optimal operator placement. §7.3 presents a case study using an implementation of MuSE graphs on top of a framework for distributed computing.

7.1 Experimental Setup

Datasets. To have a controlled setup for our simulation study, we rely on a synthetic dataset. Our case study uses a real-world dataset.

Synthetic Data: We mostly use a network consisting of 20 nodes generating events of 15 types. For scalability experiments, we consider a larger network with 50 nodes and a universe of 20 event types. The *event node ratio* denotes the share of event types that are, on average, generated at a single node. We set a default value of 0.5 for the average event node ratio, i.e., a node generates, on average, around 50% of the event types. Event generation rates are drawn from a Zipfian distribution with a default *event rate skew* of 1.5. Moreover, event generation follows a Poisson distribution.

Google Cluster Traces: This dataset contains events of a cluster monitoring solution employed at Google, see [24]. We use the first

Algorithm 3: Construction Phase

input : A set of beneficial Projections Π_{ben}; a map of combinations for each projection $\mathfrak{C}$; a query $q = (O, \lambda, P)$; an event sourced network $\Gamma = (N, f, r)$
output : A MuSE Graph G

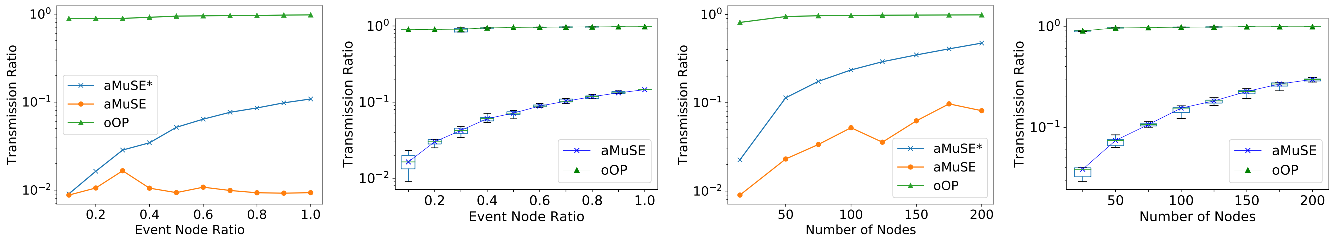
for $primOp \in O_p^q$ **do** $G[primOp][primOp] \leftarrow (V = \{(primOp, n) \mid n \in f(primOP)\}, E = \emptyset)$ `// Construct MuSE graphs for each primitive operator`
$\Pi_{sorted} \leftarrow sort(\Pi_{ben} \cup q)$ `// Sort beneficial projections based on number of primitive operators`
for $p \in \Pi_{sorted}$ **do**
  for $c = (\mathfrak{B}, \beta) \in \mathfrak{C}(p)$ **do**
    $partInput \leftarrow$ `getMSP`(p, c) `// Compute partitioning input if` (p, c) `has partitioning multi-sink placement (MSP)`
    if $partInput \neq \emptyset$ **then**
      for $PO \in O_p^{partInput}$ **do** `// For each placement option` PO `according to` $partInput$
        $Nodes_p \leftarrow \{n \mid n \in N \wedge PO \in f(n)\}$ `//` $Nodes_p$ `for placement are nodes generating` PO
        $G_{curr} \leftarrow$ `ConstructSubgraph` $(p, c, partInput, PO, Nodes_p)$
        $G[p][PO] \leftarrow \text{argmin}(c(G_{curr}), c(G[p][PO]))$ `// Maintain only the one MuSE graph per` PO `which has minimal costs`
    else
      for $e \in \beta(p)$ **do** `// For each predecessor projection` e `according to` c
        for $PO \in O_p^e$ **do** `// For each placement option` PO `according to` e
          $node_p \leftarrow$ `getSSP`(e, PO) `// Get` $node_p$ `for single-sink placement`
          $G_{curr} \leftarrow$ `ConstructSubgraph` $(p, c, e, PO, \{node_p\})$
          $G[p][PO] \leftarrow \text{argmin}(c(G_{curr}), c(G[p][PO]))$
return $\text{argmin}_{PO \in O_p^q}\, c(G[q][PO])$ `// Return MuSE graph for` q `which has minimal costs.`

function `getMSP` (*projection, combination*)
  `/* returns partitioning input if` *combination* `yields partitioning multi sink placement */`
  for $e \in \beta(p)$ **do** `// For each predecessor projection` e `according to combination`
    if $\hat{r}(e) > \sum_{\tilde{e} \in \beta(projection) \setminus e} \hat{r}(\tilde{e}) \cdot \mathfrak{C}(\tilde{e})$ **then** `// Check if rate of` e `is high enough to be a partitioning input`
      return e `// Return partitioning input if found`
  return $\emptyset$

function `getSSP` (*predProjection, placeOption*)
  `/* returns node for single-sink placement */`
  $Sinks \leftarrow G[predProjection][placeOption].Sinks$
  $Nodes \leftarrow \{n \mid (x, n) \in Sinks\}$ `// Extract nodes from sink vertices`
  return $Nodes[0]$ `// Choose one of the nodes for the single-sink placement`

function `ConstructSubgraph` (*projection, combi, predProjection, placeOption,* $Nodes_p$)
  `/* returns MuSE graph` G `for` *projection* `with respect to` *placeOption* `of` *predProjection* `*/`
  $V \leftarrow \emptyset; E \leftarrow \emptyset; G \leftarrow (V, E)$ `// Initialize` G `with empty set of vertices and edges`
  $Sinks \leftarrow \{(projection, n_p) \mid n_p \in Nodes_p\}$ `// Construct sink vertices for` G `using the nodes of the given` $Nodes_p$
  $V \leftarrow V \cup Sinks$ `// Add sinks to set of vertices`
  $G \leftarrow G \cup G[predProjection][placeOption]$ `// Add` $G[predProjection][placeOption]$ `as sub-graph to` G
  if $|Sinks| = 1$ **then** $E \leftarrow E \cup \{(x, y) \mid x \in G[predProjection][placeOption].Sinks \wedge y \in Sinks\}$ `// If` G `has a single-sink placement of` $projection$`, connect sink of` G `to each sink of` $G[predProjection][placeOption]$
  else
    $E \leftarrow E \cup \{(predProjection, m), (projection, n) \mid (predProjection, m) \in G[predProjection][placeOption].Sinks \wedge (projection, n) \in Sinks \wedge n = m\}$
    `//` G `has a multi-sink placement: connect sinks of` G `and` $G[predProjection][placeOption]$ `accordingly`
  for $e \in \beta(projection) \setminus predProjection$ **do** `// For each other predecessor projection according to` $combi$`, choose MuSE graph to add as sub-graph`
    $G_e, G_{curr}, E_e, E_{curr} \leftarrow \emptyset$
    for $placeOption_e \in O_p^e$ **do**
      $G_{curr} \leftarrow G \cup G[e][placeOption_e]$
      $E_{curr} \leftarrow \{(x, y) \mid x \in G[e][placeOption_e].Sinks \wedge y \in G.Sinks\}$ `// Construct set of edges which connects the sinks from` $G[e][placeOption_e]$ `to the sinks of` G
      $G_{curr} \leftarrow G_{curr} \cup (\emptyset, E_{curr})$
      if $G_e == \emptyset$ *or* $c(G_{curr}) == min(c(G_e), c(G_{curr}))$ **then**
        $G_e \leftarrow G[e][placeOption_e]$ `// Keep graph` $G[e][placeOption_e]$ `for which adding` $G[e][placeOption_e]$ `as sub-graph to` G `costs the least`
        $E_e \leftarrow E_{curr}$
    $G \leftarrow G \cup G_e$ `// Add` G_e `as sub-graph to` G
    $E \leftarrow E \cup E_e$ `// Add edges to` G `that connect sinks of` G `and` G_e
return G

12h of the events related to tasks running on the cluster, a dataset of $770k$ events. Each event specifies a type, a timestamp, and a machine identifier. Here, the nine possible event types denote state transitions in the life-cycle of the corresponding task. The dataset comprises the traces of around $12.3k$ machines, which we partitioned randomly into 20 sets. Based thereon, we generated event streams for a network of 20 nodes. The event node ratio is one, i.e., each node is capable



(a) (b) (c) (d)

Figure 5: Network characteristics: Varying the event node ratio and the network size.

of generating events of each type. The rates of event generation have been extracted directly from the dataset per event type.

Query workload. For the simulation experiments, we generated query workloads of five queries, which have six primitive operators on average. For scalability experiments, we used a workload of 15 queries with, on average, eight primitive operators. The queries contain sequence and conjunction operators and differ in their operator hierarchy and nesting depth. To simulate the query predicates, we generate selectivity values for each pair of event types based on a uniform distribution over range $[0.01, 0.2]$. Once an event type is generated by at least two nodes with a rate larger than one, the minimal selectivity of an operator to result in a beneficial projection is given by 0.2.

For the real-world data, we used two queries that describe common monitoring scenarios. The queries are given in Listing 1, adopting a notation similar to the SASE language [1]. For instance, the first query captures that after a task failed, another task of the same job is first evicted and 'killed', before it is rescheduled with updated scheduling constraints. The time window for each query is 30min, and as such reflects the life-time of 85% of jobs in the dataset.

```
Query 1:
PATTERN SEQ(Fail f, Evict e, Kill k, Update1 u)
WHERE f.uID=e.uID ∧ e.uID=k.uID ∧ k.uID=u.uID
WITHIN 30min

Query 2:
PATTERN AND(Finish fi, Fail fa, Kill k, Update1 u)
WHERE fi.jID=fa.jID ∧ fa.jID=k.jID ∧ k.jID = u.jID
WITHIN 30min
```

Listing 1: CEP queries used for cluster monitoring.

MuSE graphs. In our simulation experiments, we construct MuSE graphs using aMuSE or its variant, aMuSE* (§6.2). In our case study with real-world data, the MuSE graph is constructed based on aMuSE. We also implemented a branch-and-bound version of Alg. 1 for the construction of optimal MuSE graphs. Yet, even for small problem instances (i.e., four network nodes, queries with four primitive operators), the computation takes around 24h. Since such small instances do not constitute relevant application scenarios, we forgo including the results of the optimal algorithm in our evaluation.

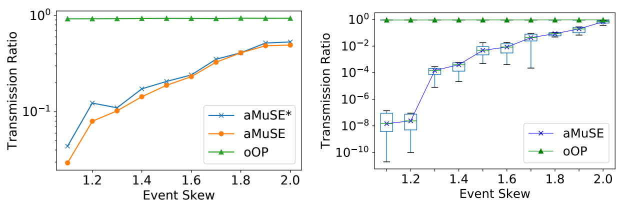
Baselines. We use two baselines for comparison. First, a *centralized* evaluation plan is used, in which all events generated by the network nodes are sent to a central instance outside the network. Second, we incorporate an algorithm for traditional *optimal operator placement (oOP)* for distributed CEP, which exhaustively searches the optimal single-sink placement for a given set of operators.

Metrics. We compare the evaluation plans constructed by aMuSE, aMuSE*, and oOP using the network costs caused by centralized evaluation as a reference point. That is, we compute the *transmission ratio*, the network costs (rates of events sent) induced by the evaluation plan relative to the network costs of centralized evaluation. The rates of events sent over the network, can be seen as a proxy for *throughput* and *latency*: As the number of events to process at a node decreases, also the number of maintained partial matches decreases. Under a negligible network delay, the latency and throughput of query processing depend almost entirely on the number of maintained partial matches [26]. In our case study using a framework for distributed computing, however, we also measure the evaluation efficiency in terms of throughput and latency. Moreover, we report on the *time to construct* a MuSE graph using aMuSE and aMuSE*.

Case study environment. For the case study, a lightweight automata-based query processor was implemented in C#. To achieve a realistic evaluation setup, we implemented the query processor using Ambrosia [15], a framework for resilient distributed computing. The framework provides fault-tolerance for applications that typically run in a Cloud environment. It offers 'virtual resiliency' by encapsulating the application code running at each cluster node in a so-called immortal. The latter can be seen as a wrapper that handles checkpointing of the application's state and materializes all function calls sent and received by the application in a log. As such, applications running in Ambrosia benefit from comprehensive replayability under exactly-once semantics for all function calls.

Using the standard Ambrosia interfaces, each node is implemented by an immortal. A node generates primitive events based on the respective event streams and evaluates a set of assigned query projections. Based on a given MuSE graph, a node retrieves the set of projections to evaluate locally as well as a protocol that dictates the event flow in the network. To provide seamless recovery of a node failure, the state of each node contains, among others, its current input queue and the set of partial matches.

We used an automata-based approach for the evaluation of projections. The input of an automaton for a given projection p can contain arbitrary sub-projections of p. Due to the distribution, results of these sub-projections may arrive in arbitrary order. Hence, we constructed the automatons such that at each state, the result of each sub-projection that is still required to be processed, can actually be evaluated. Constraints on the order of the results of sub-projections are checked at guards assigned to each transition of the automaton.



(a) **(b)**

Figure 6: Network characteristics: Varying event skew.

7.2 Simulation Experiments

Our experiments focus on the sensitivity of the costs of data transmission for varying properties of the network and of the query workload. Significant result variance is indicated by box plots. Finally, we also explore the efficiency of the MuSE graph construction.

We note that in all of the experiments described below, the results obtained for traditional optimal operator placement (oOP) are almost insensitive to parameter changes, and only slightly better than those of a centralized evaluation. This is due to our problem setting being given by a network that has a complete graph as underlying topology. However, traditional strategies for operator placement have been tailored for networks, where data transmission costs are heavily influenced by the distances between the nodes in a network.

Impact of the event node ratio. We first explore changes in the event node ratio, i.e., the share of event types that are generated, on average, per node. Fig. 5a shows that for ratio of 0.2 – a rather heterogeneous network – MuSE graphs lead to large improvements in the evaluation efficiency. They require only around 1% of the costs of centralized evaluation, whereas the improvements obtained with traditional placements with single-sinks, oOP, are negligible.

Increasing the event node ratio, the total rate with which events are generated in the network increases, which causes higher placement costs for both single-sink and multi-sink placements. Yet, even for a ratio of 1.0, evaluation with MuSE graphs keeps the network costs at around 10% (aMuSE*) or lower (aMuSE). This trend is confirmed in Fig. 5b for larger networks with more event types.

Moreover, the difference between the results for aMuSE and aMuSE* is explained as follows. With an increasing event node ratio, the number of possible multi-sink placements decreases as, for a projection to yield a multi-sink placement, a single event type has to have a rate higher then the total rate of all other input types of the projection. While this leads to the transmission ratio of aMuSE* growing almost linearly with the event node ratio, this effect is negligible for the MuSE graphs computed by aMuSE. aMuSE* considers only projections for which one event type used as input has a rate higher than the output rate of the projection. Increasing total rates of the event types also results in increased output rates of projections, so that aMuSE* explores fewer projections than aMuSE.

Impact of the network size. Varying the network size, the total number of generated events and, thus,the total rate per event type increases. As such, the effect of this parameter is the same as for the event node ratio, see Fig. 5c and, for a larger network, Fig. 5d. However, in the previous experiment, the number of nodes that potentially generate events of a certain event type, and thus contributing to its total rate, was bounded by the network size. This is not the case in this experiment. Hence, the difference in the quality of the results obtained with aMuSE and aMuSE* is more pronounced.

Impact of the event skew. Next, we vary the Zipfian parameter to derive the rates for event types. With an event skew of 2.0, rates are nearly equivalent across types. For a value of 1.1, the difference in rates of event types may be up to 1000000×. Query evaluation with MuSE graphs leverages differences in the event rates and hence, benefits from skewed event rates, see Fig. 6a. For an event skew of 1.1, the transmission ratio is up to 1000× lower than with traditional optimal operator placement for single-sinks plan. The scalability experiment with a large network in Fig. 6a confirms the trend.

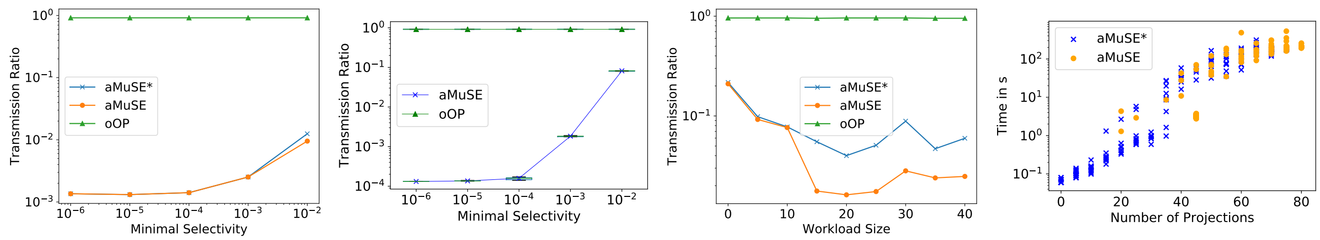
Impact of the query selectivity. Turning to properties of a query workload, we vary the minimal selectivity for each pair of event types. The baseline, oOp, does barely benefit from smaller selectivity values, see Fig. 7a. In contrast and as expected, small selectivity values are beneficial for MuSE graphs and reduces the network costs down to 0.1% of the centralized evaluation. Small selectivity values lower the output rate of projections, thereby increasing the set of beneficial projections. In addition, more multi-sink placement can be constructed: Even event types with medium rates are candidates for the partitioning type of multi-sink placements, when combined with projections having a small output rate. These observations hold independent of the adopted algorithm for MuSE graph construction. The experiment with a larger network, Fig. 7b, shows that the selectivity values do not impose a lot of result variance.

Impact of the workload size. We further explored the workload size as a parameter influencing the efficiency of query evaluation. Fig. 7c illustrates that the quality of the MuSE graphs, for both aMuSE and aMuSE*, is almost independent of the number of queries. In a small workload, the number of event types referenced in the queries and, thus, the total rate of event types to exchange in centralized evaluation tends to be smaller. Hence, the optimization potential to leverage with MuSE graphs also becomes smaller.

Efficiency of MuSE graph construction. Finally, we analyze the efficiency of aMuSE and aMuSE*. Fig. 7d documents for each of the above experiments, the computation time and the number of projections considered for MuSE graph construction. Overall, the runtimes are modest, staying mostly in the range of seconds for aMuSE*, and dozens of seconds for aMuSE. In contrast to aMuSE*, aMuSE does not only consider more projections, but also explores more placement options for each projection. Thus, even for the same number of projections, aMuSE has a higher runtime than aMuSE*.

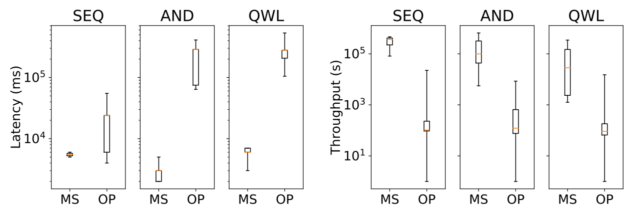
7.3 Case Study

We report the results obtained as part of our case study when running each of the considered queries in isolation (denoted as AND and SEQ), and jointly as part of a workload (QWL). In general, we observe that each node in the network is capable of generating each event type, while the rates per event type extracted from the dataset are roughly equivalent per node. As a result, oOP yields an operator placement, in which all operators are placed at *one* single node. Thus, the network costs of these plans are given by the sum of the total rates of the primitive events contained in a query, reduced by the rate of events generated by the node hosting the query.



(a) **(b)** **(c)** **(d)**

Figure 7: (a)-(c) Query workload characteristics: Varying selectivity and workload size. (d) Efficiency MuSE of graph construction.



(a) **(b)**

Figure 8: Case Study: Latency and Throughput.

In contrast, the plans computed by aMuSE contain projections that exploit the differences between the rates of the event types contained in the queries: For example, an event denoting that a task was finished occurs rather frequent, while update events, which capture that a task was rescheduled with updated resource constraints, are rare. aMuSE constructs MuSE graphs containing projections over such event combinations, together with a multi-sink placement thereof. With each node being capable of generating events of each type, *all* nodes are involved in the respective multi-sink placements, which drastically reduces the transmission ratio: No events of high-rate types are ever sent over the network. As such, in terms of the transmission ratio, aMuSE and oOP confirm the trends from the simulation experiments, as summarized in Table 3.

Table 3: Case Study: Transmission Ratio

	AND	SEQ	QWL
aMuSE	4%	1%	5%
oOP	95%	91%	95%

Moreover, Fig. 8 shows the throughput and latency observed for query evaluation (in terms of *min, 25th, 50th, 75th, max)*. Here, *MS* denotes the results for MuSE graphs, while *OP* is the traditional operator placement. The evaluation with MuSE graphs reduces throughput and latency significantly. The main reason is that multi-sink placements reduce the number of events to be processed by a single node. Hence, the number of partial matches each node must maintain is also reduced, speeding up the processing. Moreover, there is more variance in the *OP* results. This is caused by changes in the query selectivity over time. Since OP plans generate more partial matches, query evaluation is affected from this effect more drastically compared to the evaluation with MuSE graphs.

8 RELATED WORK

Distributed Stream Processing. Operator placement has been investigated for distributed stream processing [17, 18], distributed CEP [10, 11, 13], and sensor networks [8, 21]. While proposed approaches differ in the assumptions imposed on the network and optimization metrics, the operator placement problem is always formulated such that an operator is placed at exactly one node, leading to single-sink placements. As a consequence, these approaches suffer from the aforementioned issues: They assume a hierarchical system structure and lack support for compositional query workloads. In [11, 13] and [10] single-sink operator placements for CEP queries is combined with push-pull communication [2]. Similar to multi-sink placements, push-pull communication leverages skewed event rates, so that an integration in MuSE graphs is a direction for future work.

Pattern Sharing and Query Rewriting. A CEP query may be rewritten before placing its operators, to reduce the number of partial matches exchanged in a network [20]. While this is similar to our ideas, MuSE graphs provide more degrees of freedom, as arbitrary projections and combinations thereof can be considered for distributed query evaluation. (Sub-)Pattern sharing for CEP queries was investigated for centralized evaluation [19], i.e., in a setting where events of *all* types are available for evaluation at a single node. The proposed metrics could guide the selection of projections to use for the construction of a MuSE graph. With the goal to reduce CPU costs, the properties of a useful sub-pattern in [19], do not directly translate to our goal of reducing network costs. However, sharing-based multi-query optimization is orthogonal to our work and may be used to speed up processing on a node level.

9 CONCLUSION

In this paper, we proposed Multi-Sink Evaluation (MuSE) graphs for distributed event stream processing that overcome two fundamental limitations of existing models: Instead of considering only the operator hierarchy for distribution, MuSE graphs exploit arbitrary query projections. Also, MuSE graphs lift the restriction to place each operator solely at a single network node. Addressing the main challenges imposed by distributed CEP, we formally showed that MuSE graphs enable correct and cost-efficient query evaluation in a networked setting, and also proposed algorithms for the efficient construction of MuSE graphs. Through simulation experiments, we showed the benefits of event processing with MuSE graphs, reducing the network costs down to 0.1% of centralized evaluation and traditional operator placement. In a case study we further demonstrated practical feasibility.

REFERENCES

[1] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. 2008. Efficient pattern matching over event streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, Jason Tsong-Li Wang (Ed.). ACM, 147–160. https://doi.org/10.1145/1376616.1376634

[2] Mert Akdere, Uğur Çetintemel, and Nesime Tatbul. 2008. Plan-based complex event detection across distributed sources. *Proceedings of the VLDB Endowment* 1, 1 (2008), 66–77.

[3] Arvind Arasu, Brian Babcock, Shivnath Babu, John Cieslewicz, Mayur Datar, Keith Ito, Rajeev Motwani, Utkarsh Srivastava, and Jennifer Widom. 2016. STREAM: The Stanford Data Stream Management System. In *Data Stream Management - Processing High-Speed Data Streams*, Minos N. Garofalakis, Johannes Gehrke, and Rajeev Rastogi (Eds.). Springer, 317–336. https://doi.org/10.1007/978-3-540-28608-0_16

[4] Alexander Artikis, Alessandro Margara, Martín Ugarte, Stijn Vansummeren, and Matthias Weidlich. 2017. Complex Event Recognition Languages: Tutorial. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017, Barcelona, Spain, June 19-23, 2017*. ACM, 7–10. https://doi.org/10.1145/3093742.3095106

[5] Alexander Artikis, Matthias Weidlich, François Schnitzler, Ioannis Boutsis, Thomas Liebig, Nico Piatkowski, Christian Bockermann, Katharina Morik, Vana Kalogeraki, Jakub Marecek, Avigdor Gal, Shie Mannor, Dimitrios Gunopulos, and Dermot Kinane. 2014. Heterogeneous Stream Processing and Crowdsourcing for Urban Traffic Management. In *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014*, Sihem Amer-Yahia, Vassilis Christophides, Anastasios Kementsietsidis, Minos N. Garofalakis, Stratos Idreos, and Vincent Leroy (Eds.). OpenProceedings.org, 712–723. https://doi.org/10.5441/002/edbt.2014.77

[6] Roger S. Barga, Jonathan Goldstein, Mohamed H. Ali, and Mingsheng Hong. 2007. Consistent Streaming Through Time: A Vision for Event Stream Processing. In *CIDR 2007, Third Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 7-10, 2007, Online Proceedings*. www.cidrdb.org, 363–374. http://cidrdb.org/cidr2007/papers/cidr07p42.pdf

[7] Shahid H. Bokhari. 1981. A Shortest Tree Algorithm for Optimal Assignments Across Space and Time in a Distributed Processor System. *IEEE Trans. Software Eng.* 7, 6 (1981), 583–589. https://doi.org/10.1109/TSE.1981.226469

[8] Georgios Chatzimilioudis, Alfredo Cuzzocrea, Dimitrios Gunopulos, and Nikos Mamoulis. 2013. A novel distributed framework for optimizing query routing trees in wireless sensor networks via optimal operator placement. *J. Comput. System Sci.* 79, 3 (2013), 349–368.

[9] Jianxia Chen, Lakshmish Ramaswamy, David K. Lowenthal, and Shivkumar Kalyanaraman. 2012. Comet: Decentralized Complex Event Detection in Mobile Delay Tolerant Networks. In *13th IEEE International Conference on Mobile Data Management, MDM 2012, Bengaluru, India, July 23-26, 2012*, Karl Aberer, Anupam Joshi, Sougata Mukherjea, Dipanjan Chakraborty, Hua Lu, Nalini Venkatasubramanian, and Salil S. Kanhere (Eds.). IEEE Computer Society, 131–136. https://doi.org/10.1109/MDM.2012.18

[10] Jianxia Chen, Lakshmish Ramaswamy, David K Lowenthal, and Shivkumar Kalyanaraman. 2012. Comet: Decentralized complex event detection in mobile delay tolerant networks. In *2012 IEEE 13th International Conference on Mobile Data Management*. IEEE, 131–136.

[11] Gianpaolo Cugola and Alessandro Margara. 2013. Deployment strategies for distributed complex event processing. *Computing* 95, 2 (2013), 129–156.

[12] Raul Castro Fernandez, Matthias Weidlich, Peter R. Pietzuch, and Avigdor Gal. 2014. Scalable stateful stream processing for smart grids. In *The 8th ACM International Conference on Distributed Event-Based Systems, DEBS '14, Mumbai, India, May 26-29, 2014*, Umesh Bellur and Ravi Kothari (Eds.). ACM, 276–281. https://doi.org/10.1145/2611286.2611326

[13] Ioannis Flouris, Nikos Giatrakos, Antonios Deligiannakis, and Minos N. Garofalakis. 2020. Network-wide complex event processing over geographically distributed data sources. *Inf. Syst.* 88 (2020). https://doi.org/10.1016/j.is.2019.101442

[14] Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos N. Garofalakis. 2020. Complex event recognition in the Big Data era: a survey. *VLDB J.* 29, 1 (2020), 313–352. https://doi.org/10.1007/s00778-019-00557-w

[15] Jonathan Goldstein, Ahmed S. Abdelhamid, Mike Barnett, Sebastian Burckhardt, Badrish Chandramouli, Darren Gehring, Niel Lebeck, Christopher Meiklejohn, Umar Farooq Minhas, Ryan Newton, Rahee Peshawaria, Tal Zaccai, and Irene Zhang. 2020. A.M.B.R.O.S.I.A: Providing Performant Virtual Resiliency for Distributed Applications. *Proc. VLDB Endow.* 13, 5 (2020), 588–601. http://www.vldb.org/pvldb/vol13/p588-goldstein.pdf

[16] InSystems. 2021. proANT Transport Robots . http://www.insystems.de/en/produkte/proant-transport-roboter/.

[17] Matteo Nardelli, Valeria Cardellini, Vincenzo Grassi, and Francesco LO PRESTI. 2019. Efficient Operator Placement for Distributed Data Stream Processing Applications. *IEEE Transactions on Parallel and Distributed Systems* (2019).

[18] Peter R. Pietzuch, Jonathan Ledlie, Jeffrey Shneidman, Mema Roussopoulos, Matt Welsh, and Margo I. Seltzer. 2006. Network-Aware Operator Placement for Stream-Processing Systems. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, Ling Liu, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang (Eds.). IEEE Computer Society, 49. https://doi.org/10.1109/ICDE.2006.105

[19] Medhabi Ray, Chuan Lei, and Elke A. Rundensteiner. 2016. Scalable Pattern Sharing on Event Streams. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 495–510. https://doi.org/10.1145/2882903.2882947

[20] Nicholas Poul Schultz-Møller, Matteo Migliavacca, and Peter R. Pietzuch. 2009. Distributed complex event processing with query rewriting. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, DEBS 2009, Nashville, Tennessee, USA, July 6-9, 2009*, Aniruddha S. Gokhale and Douglas C. Schmidt (Eds.). ACM. https://doi.org/10.1145/1619258.1619264

[21] Utkarsh Srivastava, Kamesh Munagala, and Jennifer Widom. 2005. Operator placement for in-network stream query processing. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART sym posium on Principles of database systems*. ACM, 250–258.

[22] Fabrice Starks, Vera Goebel, Stein Kristiansen, and Thomas Plagemann. 2018. Mobile Distributed Complex Event Processing - Ubi Sumus? Quo Vadimus? In *Mobile Big Data, A Roadmap from Models to Technologies*, Georgios Skourletopoulos, George Mastorakis, Constandinos X. Mavromoustakis, Ciprian Dobre, and Evangelos Pallis (Eds.). Lecture Notes on Data Engineering and Communications Technologies, Vol. 10. Springer, 147–180. https://doi.org/10.1007/978-3-319-67925-9_7

[23] Kia Teymourian, Malte Rohde, and Adrian Paschke. 2012. Knowledge-based processing of complex stock market events. In *15th International Conference on Extending Database Technology, EDBT '12, Berlin, Germany, March 27-30, 2012, Proceedings*, Elke A. Rundensteiner, Volker Markl, Ioana Manolescu, Sihem Amer-Yahia, Felix Naumann, and Ismail Ari (Eds.). ACM, 594–597. https://doi.org/10.1145/2247596.2247674

[24] John Wilkes. 2020. Yet more Google compute cluster trace data. Google research blog. Posted at https://ai.googleblog.com/2020/04/yet-more-google-compute-cluster-trace.html.

[25] Lei Ying, Zhen Liu, Don Towsley, and Cathy H Xia. 2008. Distributed operator placement and data caching in large-scale sensor networks. In *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*. IEEE, 977–985.

[26] Haopeng Zhang, Yanlei Diao, and Neil Immerman. 2014. On complexity and optimization of expensive queries in complex event processing. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu (Eds.). ACM, 217–228. https://doi.org/10.1145/2588555.2593671