MuSE Graphs for Flexible Distribution of Event Stream Processing in Networks

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

ABSTRACT

Complex event processing (CEP) supports reactive applications through the continuous evaluating of queries over streams of event data. In a network of event sources, efficient query evaluation is achieved by distribution: Queries are split into operators (query decomposition), which are then assigned to some of the nodes (operator placement). Yet, existing solutions 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 optimizations based on pattern composition 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 shows that MuSE graphs reduce network transmission costs by up to three orders of magnitude over baseline strategies.

CCS CONCEPTS

• **Information systems** → **Data streams**.

KEYWORDS

complex event processing, operator placement, query distribution

ACM Reference Format:
Samira Akili and Matthias Weidlich. 2021. MuSE Graphs for Flexible Distribution of Event Stream Processing in Networks. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3448016.3457318

1 INTRODUCTION

Complex event processing (CEP) is a computational paradigm that is based on the continuous evaluation of queries over streams of event data [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 [24].

SIGMOD '21, June 20–25, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3457318

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, 23]. 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 [27], 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 [23]. Even under these assumptions, operator placement was shown to be NP-hard [26], 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. 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.

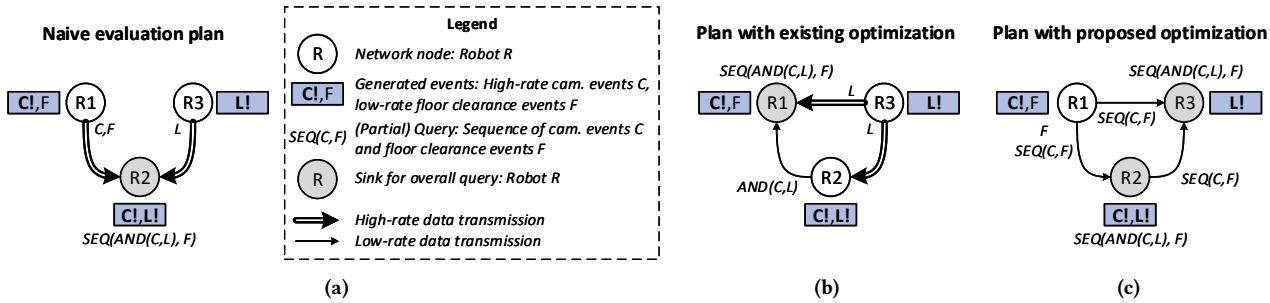


Figure 1: Setting of three transport robots R1-R3 emitting events of camera (C) and lidar (L) sensors with high rates, and, rarely, floor clearance events (F): (a) Data transmission when evaluating the query *SEQ(AND(C,L),F)* naively; (b) with existing optimization strategies; (c) using the ideas proposed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

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 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 framework for distributed computing (§7). Our results shows 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* $\Gamma = (N, f, r)$ comprises a set of nodes N, a function $f : N \rightarrow 2^{\mathcal{E}}$ that assigns event types to nodes, and a function $r : \mathcal{E} \rightarrow \mathbb{R}$ 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 specified order;
- *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 $\lambda : O \twoheadrightarrow O^k$, $k \in \mathbb{N}$ 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 neglect 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 $o \notin \text{dom}(\lambda)$. The remaining ones are composite operators, $O_c = O \backslash O_p$. 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 by concatenating 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 $M = \{m_1.m_3 \mid m_1 = \langle e_1^1, \ldots, e_n^1 \rangle \in M_1 \wedge m_3 = \langle e_1^3, \ldots, e_m^3 \rangle \in M_3 \wedge \forall\, m_2 = \langle e_1^2, \ldots, e_l^2 \rangle \in M_2 : \#_t(e_1^2) < \#_t(e_n^1) \ \vee \ \#_t(e_l^2) > \#_t(e_1^3)\}$.

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 $\lambda : O \twoheadrightarrow O^k$, 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

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.

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 [27].

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 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.

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.

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
$\mathfrak{A}(v)$	Event type bindings covered by a vertex v of a MuSE graph
$\hat{r}(p)$	Output rate of a projection p
$\mathfrak{c} = (\mathfrak{B}, \beta)$	Combination graph: projections $\mathfrak{B}$, predecessors $\beta : \mathfrak{B} \twoheadrightarrow \mathfrak{B}^k$
$\mathfrak{C}(Q)$	Set of all combinations of a query workload Q

4.2 Query Projections

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 $2^{|O_p|}$.

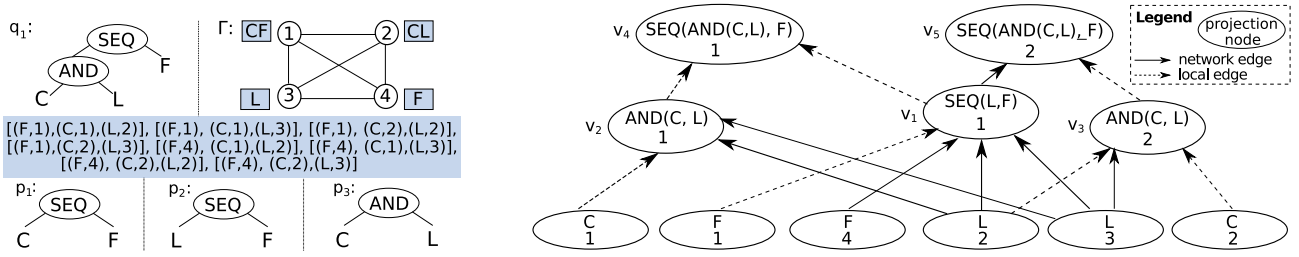
Example 4. *Fig. 2a (bottom) shows projections for query* q_1 *for the sets of event types* $\{C, F\}$ *(*p_1*),* $\{L, F\}$ *(*p_2*), and* $\{C, L\}$ *(*p_3*).*

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.



(a) (b)

Figure 2: (a) Query q_1 to be evaluated in network Γ (top); 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.

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:

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, $c : E \rightarrow \mathbb{R}^+$.*

As a next step, we explain the intuition of the model. 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.

Our 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: 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 $\mathfrak{V}(v) \subseteq \mathfrak{E}(p)$ 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 possible 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 the query 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 $\{[(C, 2), (L, 2)], [(C, 2), (L, 3)]\}$.*

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, of a projection, in a network $\Gamma = (N, f, r)$. The output rate $\hat{r}$ is defined recursively, along the operator hierarchy. For a 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 $\hat{r}(o) = \prod_{1 \leq i \leq k} \hat{r}(o_i)$.
- If $o.sem = AND$, then $\hat{r}(o) = k \cdot \prod_{1 \leq i \leq k} \hat{r}(o_i)$.
- 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 7. *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). Next, 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 $\beta : \mathfrak{V} \twoheadrightarrow \mathfrak{V}^k$, $k \in \mathbb{N}$ 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).

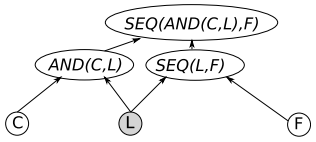


Figure 3: Combination for the query $SEQ(AND(C, L), F)$.

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 8. *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 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 1. *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 2. *A complete MuSE graph for the evaluation of a workload Q generates all matches of each query $q \in Q$.*

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 9. *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 $\mathfrak{c} = (\mathfrak{V}, \beta)$, we verify correctness (line 5). For correct combinations, we determine the non-primitive projections line 6) and construct all possible assignments 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

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** $A \in \left\{\bigcup_{\mathfrak{e} \in \mathfrak{E}(q,\Gamma)} \{(\mathfrak{e}, V_\mathfrak{e})\} \mid V_\mathfrak{e} \in C\right\}$ **do** $\mathcal{G} \leftarrow \mathcal{G} \cup constructMuSE(A)$
10 $G_{opt} \leftarrow \operatorname{argmin}_{G \in \mathcal{G}} c(G)$ // Select optimal MuSE graph
11 **return** G_{opt}

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 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 [26]. 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. □

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). Proofs of the presented pruning principles and further details on the presented algorithms, including a variant for multi-query workloads, can be found in [20].

6.1 Pruning Principles for MuSE Graphs

Beneficial projections. Using a projection in a MuSE graph shall reduce the rate with which events are exchanged, compared to a centralized evaluation of the query for which *all* events need to be exchanged. For a MuSE graph $G = (V, E, c)$, it will only be beneficial to contain a projection, i.e. to have a vertex $v = (p, n) \in V$, 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 the case, we refer to v as a *beneficial* vertex. If this is not the case, sending matches of predecessors of the vertex v directly to all its successors yields lower costs. We proved that an optimal MuSE graph contains only beneficial vertices (see the extended version [20]).

Example 10. *In Fig. 2, vertex v_2 covers two event type bindings of the projection p_3: $\mathfrak{A}(v_2) = \{[(C,1),(L,2)],[(C,1),(L,3)]\}$. Thus, v_2 is a beneficial vertex, if it holds that $2 \cdot \hat{r}(p_3) \leq 2 \cdot \hat{r}(L) + \hat{r}(C)$.*

To select suitable projections for the construction of MuSE graphs, we prune those that do not yield beneficial vertices. This is the case, if for each possible combination of a projection p, the sum of the rates of the predecessor projections is greater than the rate of p. Consequently, we define *beneficial* projections as follows:

Definition 11. *A projection p is beneficial, if there exists a combination $\mathfrak{c} = (\mathfrak{B}, \beta) \in \mathfrak{C}(\{p\})$, such that $\hat{r}(p) \leq \sum_{e \in \beta(p)} \hat{r}(e)$.*

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 all projections that turn out to be non-beneficial based on their respective primitive combination.

Uniform 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. We denote the resulting class of MuSE graphs as G^{uni}. A MuSE graph $G \in G^{uni}$ has exactly one underlying combination $c \in \mathfrak{C}(\{q\})$. In the remainder, we only consider MuSE graphs in G^{uni}.

Example 11. *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 predecessor projections of q_1.*

We showed formally that for a correct MuSE graph $G \in G^{uni}$, for each projection p contained in the respective underlying combination of G, matches for *all* event type bindings $\mathfrak{E}(p)$ are generated at vertices of G (see [20]).

Redundant combinations. We further restrict the combinations considered for the construction of a MuSE graph by identifying those that are redundant, i.e., that contain superfluous projections.

Definition 12. *A combination $\mathfrak{c} = (\mathfrak{B}, \beta) \in \mathfrak{C}(\{q\})$ is* redundant, *if there exists a projection $p \in \mathfrak{B}$ such that p has a predecessor projection $\tilde{e} \in \beta(p)$ for which $O_p^{\tilde{e}} \subseteq \bigcup_{e \in \beta(p) \setminus \{\tilde{e}\}} O_p^e$.*

Example 12. *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.*

From Def. 12 it follows directly that for a non-redundant combination of a projection p, it holds that the number of predecessor projections is limited to $|O_p^p|$. Moreover, we proved that the underlying combination of an optimal MuSE Graph in G^{uni} is not redundant (see [20]). In the remainder, we prune redundant combinations from the search space.

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. Hence, it must be guaranteed that the nodes hosting a projection p, jointly cover $\mathfrak{E}(p)$. We refer to the vertices, at which matches of the projection p are generated, as the *placement* of p. A *single-sink placement* $v_p = (p, n) \in V$ of projection p generates the matches of p at one single node. For it to be a correct, it must hold that $\mathfrak{A}(v_p) = \mathfrak{E}(p)$. A *multi-sink placement* $V_p \subseteq V$ of p assigns the generation of matches of p to a set of nodes. It is correct, if $\bigcup_{v_p \in V_p} \mathfrak{A}(v_p) = \mathfrak{E}(p)$. Next, let the sum of the edge weights of incoming edges of the respective vertices be the placement cost, denoted by $c_p(V_p)$, of a placement V_p. Then, placing p at nodes that generate predecessor projections of p yields low placement costs, since some of the incoming edges are local edges (weight of zero).

Example 13. *In Fig. 2 the vertices $V_{p_3} = \{v_2, v_3\}$ jointly cover $\mathfrak{E}(p_3)$ and thus, illustrate a correct multi-sink placement for p_3 with placement costs $c_p(V_{p_3}) = 3 \cdot \hat{r}(L)$. In contrast, $V_{p_2} = \{v_1\}$ illustrates a correct single-sink placement for the projection p_2 with placement costs $c_p(V_{p_2}) = 2 \cdot \hat{r}(L) + 1 \cdot \hat{r}(F)$.*

Minimal multi-sink placements. The placement costs of placements of a projection p depend on the combination $\mathfrak{c}$ used to generate matches of p, and on the placement of the predecessor projections of p. Once all predecessor projections of p are placed, we can efficiently compute a single-sink placement having minimal placement costs by iterating over all nodes that generate a predecessor projection $e \in \beta(p)$ and comparing the resulting costs. This is not possible for multi-sink placements, due to the exponential search space. We therefore derived properties for a multi-sink placement V_p of a projection p and combination $\mathfrak{c} = (\mathfrak{B}, \beta)$ that have to hold for V_p for it to yield minimal placement costs (see [20]):

(1) $\exists\ e_{part} \in \beta(p)$, s.t. $\hat{r}(e_{part}) \geq \sum_{e \in \beta(p) \setminus \{e_{part}\}} \hat{r}(e) \cdot |\mathfrak{E}(e)|$,
(2) $\forall\ v = (p, n) \in V_p$, it holds that $w = (e_{part}, m) \in V$, s.t. $n = m$,
(3) $|\{((\tilde{e}, m), v_p) \mid \tilde{e} = e_{part} \wedge m \in N \wedge v_p \in V_p\}| = 1$.

Intuitively, there exists a predecessor projection $e_{part} \in \beta(p)$ having a significantly higher rate than the other predecessor projections, and the nodes involved in the placement also generate matches of e_{part}. We refer to the respective projection e_{part} as a *partitioning input*, since the covers of the vertices in the placement V_p partition the event type bindings of p with respect to their locally generated event type bindings of e_{part}. Moreover, the matches of e_{part} generated by a node n are only locally used to generate matches

of projection p and not sent over the network to other vertices of V_p. To this end, each node involved in the placement needs all matches of event type bindings of the other predecessor projections $\beta(p) \setminus \{e_{part}\}$. We refer to minimal multi-sink placements having a partitioning input as *partitioning multi-sink placements*. In the remainder, we prune multi-sink placements for which the properties (1)-(3) do not apply and thus, construct MuSE graphs only from partitioning multi-sink placements and single-sink placements.

Example 14. *In Fig. 2 the placement $V_{p_3} = \{v_2, v_3\}$ of p_3 has the 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 having $AND(C, L)$ as root operator.*

6.2 aMuSE and aMuSE*

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

aMuSE – Enumeration phase. This phase is similar to the first steps of Alg. 1 (line 1-2). First, all beneficial projections Π_{ben} are enumerated. Then, for each projection $p \in \Pi_{ben}$, the set $\mathfrak{C}(\{p\})$ of correct and non-redundant combinations is generated. The output of this phase is the set of beneficial projections and combinations thereof. The worst-case time complexity of this phase is:

$$\mathcal{O}\left(2^{|O_p^p|} \cdot 2^{2^{|O_p^p|}}\right).$$

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. 2 outlines the construction phase. Using the set of beneficial projections and combinations thereof, MuSE graphs are constructed in a bottom-up manner using a dynamic programming approach. For each projection p, we construct MuSE graphs that have different placements of p as sinks and incorporate the generation of matches of p according to different combinations. A MuSE graph for a projection p, given a combination c, is composed of MuSE graphs of predecessor projections according to the combination c, as constructed in earlier iterations.

First, (non-primitive) projections are sorted based on their number of primitive operators, in ascending order (line 2). Then, aMuSE iterates over the sorted projections (line 3) and, for each projection p, considers all its combinations $\mathfrak{c}$ (line 4). Here, aMuSE, in general, considers only placements at nodes, which generate a predecessor projection of p according to c. For each non-primitive predecessor projection e, multiple MuSE graphs were computed in an earlier iteration (stored in $\mathbb{G}$). Each of these graphs for e is considered as sub-graph for the construction of the graphs for p.

If a multi-sink placement for the tuple (p, c) exists (line 5-6), aMuSE explores fewer placements options and thus, constructs fewer MuSE graphs (line 7-10). Otherwise, aMuSE constructs *for each* predecessor projection according to combination c several

Algorithm 2: 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.

1 $\mathbb{G} \leftarrow \emptyset$ // MuSE graphs for pairs of connected projections
2 $\Pi_{sorted} \leftarrow \text{sort}(\Pi_{ben} \cup \{q\})$ // Sort by number of prim. op.
3 **for** $p \in \Pi_{sorted}$ **do** // For each projection
4 **for** $\mathfrak{c} = (\mathfrak{B}, \beta) \in \mathfrak{C}(\{p\})$ **do** // For each combination
// Get partitioning input, if partitioning multi-sink placement (MSP) exists
5 $e_{part} \leftarrow$getMSP$(p, \mathfrak{c})$
6 **if** $e_{part} \neq \emptyset$ **then** // If MSP exists
7 **for** $po \in O_p^{e_{part}}$ **do** // For each prim. op.
8 $V_p \leftarrow \{n \mid n \in N \wedge po \in f(n)\}$
// Construct MuSE graph placing p at V_p as sinks, which contains MuSE graphs for each pred. proj. in $\mathfrak{c}$ as subgraph
9 $G_{curr} \leftarrow \text{constructMuSE}(V_p, \mathbb{G}[e_{part}][po])$
10 $\mathbb{G}[p][po] \leftarrow \text{argmin}_{G \in \{G_{curr}, \mathbb{G}[p][po]\}} c(G)$
11 **else**
12 **for** $e \in \beta(p)$ **do** // For each predecessor proj.
13 **for** $po \in O_p^e$ **do** // For each prim. op.
// Choose single-sink placement
14 $v_p \leftarrow$getSSP(e, po)
15 $G_{curr} \leftarrow$constructMuSE$(\{v_p\}, \mathbb{G}[e][po])$
16 $\mathbb{G}[p][po] \leftarrow \text{argmin}_{G \in \{G_{curr}, \mathbb{G}[p][po]\}} c(G)$
17 $\mathbb{G}' \leftarrow \bigcup_{po \in O_p^q} \{\mathbb{G}[q][po]\}$
18 **return** $\text{argmin}_{G \in \mathbb{G}'} c(G)$

MuSE graphs for projection p (line 12-16). With the query being treated as the other projections, in the last iteration, MuSE graphs for the query according to multiple placement options are constructed. The one with minimal costs constitutes the output of the algorithm (line 17-18). The worst-case time complexity of Alg. 2 is:

$$\mathcal{O}\left(|\Pi_{ben}| \cdot |\mathfrak{C}(\{q\})| \cdot |O_p^q|^4\right).$$

aMuSE*. We propose aMuSE* as a variation of aMuSE that further restricts the set of projections in the construction. aMuSE* only considers a projection p, for which there exists at least one primitive operator in $e \in O_p^p$, such that $\hat{r}(e) \geq \hat{r}(p) \cdot |\mathfrak{E}(p)|$. Moreover, aMuSE* considers for each combination of $\mathfrak{c} = (\mathfrak{B}, \beta) \in \mathfrak{C}(\{p\})$ only inputs $e \in \beta(p)$, for which $\hat{r}(e) \geq \hat{r}(p) \cdot |\mathfrak{E}(\{p\})|$ as placement options, which limits the search space for single-sink placements (enumerated in line 12-16 of Alg. 2). As a consequence, aMuSE* considers less projections, combinations, and placements thereof.

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 for 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.

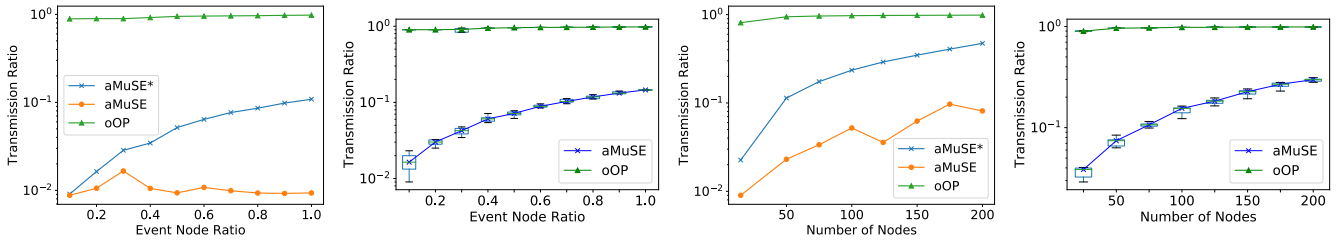


Figure 4: Network characteristics: Varying the event node ratio and the network size.

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 system at Google [25]. We use the first 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 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 with 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 $[0.01, 0.2]$. 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 queries of common monitoring scenarios (listings are available in [20]). For instance, one 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.

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 report the *transmission ratio*, i.e., the network costs (rates of events sent) induced by the evaluation plan relative to the 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 [27]. 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, we implemented a lightweight automata-based query processor in C#. To achieve a realistic evaluation setup, the implementation is based on Ambrosia [15], a framework for resilient distributed computing. Ambrosia 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

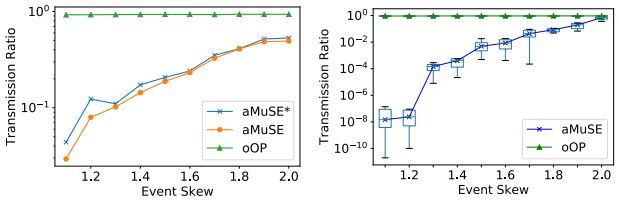


Figure 5: Network characteristics: Varying event skew.

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.

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. 4a 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. 4b for larger networks with more event types.

The difference observed between 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, one event type must 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, the effect is negligible for aMuSE. aMuSE* considers only projections for which one input type 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. 4c and, for a larger network, Fig. 4d. However, in the previous experiment, the number of nodes that potentially generate events of a certain 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. 5a. For an event skew of 1.1, the transmission ratio is up to 1000× lower than with traditional optimal operator placement for single-sink plans. The scalability experiment with a large network in Fig. 5a confirms the trend.

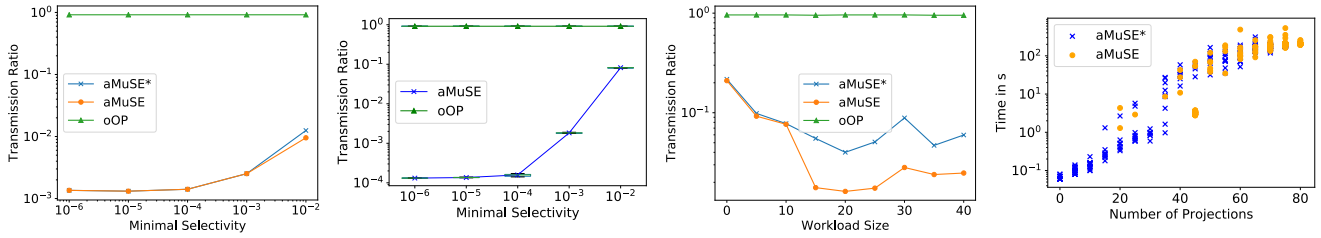
Impact of the query selectivity. Turning to properties of a query workload, we vary the minimal selectivity for each pair of event types. The baseline, oOp, barely benefits from smaller selectivity values, see Fig. 6a. In contrast, small selectivity values are beneficial for MuSE graphs and reduces the 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. Also, 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 for both, aMuSE and aMuSE*. The experiment with a larger network, Fig. 6b, shows that the selectivity values do not impose a lot of result variance.

Impact of the workload size. Fig. 6c illustrates that the quality of the MuSE graphs, for both aMuSE and aMuSE*, is almost independent of the workload size, i.e., 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. 6d shows 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*.

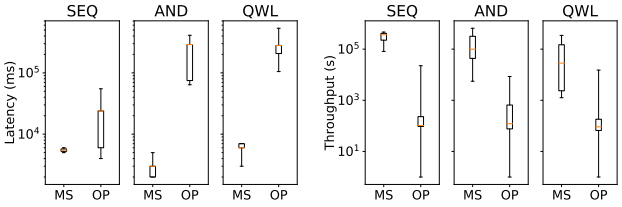
7.3 Case Study

We report the results 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 6: (a)-(c) Workload characteristics: Varying selectivity and workload size. (d) Efficiency of MuSE graph construction.



(a) (b)

Figure 7: 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. 7 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, 22]. 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 [21]. 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] Removed for review. 2021. MuSE Graphs for Flexible Distribution of Event Stream Processing in Networks – Extended Version. https://www.dropbox.com/s/apafhrdx3hmokmr/TechnicalReport_aMuse.pdf.

[21] 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

[22] 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.

[23] 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

[24] 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

[25] 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.

[26] 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.

[27] 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