From Distributed Sources to Distributed Sinks: Towards Truly Decentralized Event Stream Processing

Samira Akili
Supervised by Matthias Weidlich
Humboldt-University of Berlin
akilsami@hu-berlin.de

ABSTRACT

Distributed stream processing evaluates queries over data produced by geographically distributed sources. To efficiently handle large amounts of decentralized data, whilst coping with bandwidth restrictions, applications employ in-network processing. To this end, a query is modularized and its operators are assigned to network nodes, especially those that act as data sources. The latter is known as the operator placement problem. Existing solutions to it, however, handle data generated by distributed sources, whereas query results are gathered at one designated node – the sink.

We argue that such single-sink solutions are not applicable for non-hierarchical system, in which multiple nodes need to be informed about query results. Also, having a single sink enforces centralisation in compositional applications, where the result of one query is the input to another query. This PhD project therefore aims to develop algorithms for multi-sink operator placement. We show that the computation of network costs for efficient operator placement, however, requires us to incorporate various aspects of event generation, query processing semantics, and network properties.

1. INTRODUCTION

In domains such as logistics, smart grids, or supply chain management, applications rely on data produced by geographically distributed sources [17]. They exploit this data by evaluating streaming queries in large networks, whose nodes are defined by sensors, middleware components, and information systems. While centralized approaches for distributed stream processing (DSP) typically do not scale due to bandwidth and delay restrictions, decentralized in-network processing takes advantage of data locality to reduce networking costs. To this end, the queries of an application are split into operators, which are evaluated inside the network, possibly directly at the nodes that denote data sources. Hence, at the core of (DSP) lies the modularization of a query workload and the assignment of operators to the nodes of the network. The latter is known in literature as the *operator placement* problem [14].

Proceedings of the VLDB 2019 PhD Workshop, August 26th, 2019. Los Angeles, California.

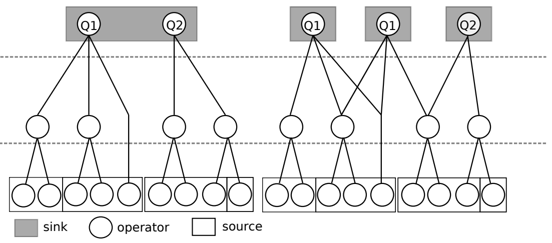


Figure 1: Evaluation of two queries, Q_1 and Q_2, with traditional single-sink operator placement (left) and multi-sink operator placement (right).

While existing approaches for DSP place operators inside the network, the results of an application are gathered at *one* designated node [17], i.e. the *sink*, which is usually located in a data centre. In this work, we argue that a single-sink approach is not suitable for non-hierarchical systems and compositional applications. Non-hierarchical systems, such as autonomous agent networks or car platooning, require a large number of nodes, potentially the whole network, to be informed about query results. At the same time, it is often desirable to build streaming applications by composition: The sink of one query represents a source for another one. By enforcing a single sink, however, existing approaches do not support decentralisation beyond a single query.

This PhD project aims at developing foundations of distributed stream processing with multiple sinks. Unlike traditional operator placement that enforces a single sink node, see Figure 1 (left), we strive for placements with multiple sinks, see Figure 1 (right). This way, we achieve truly decentralised stream processing that caters for distributed information demands in non-hierarchical systems and avoids centralisation in query composition. Considering multiple sinks yields a higher degree of freedom for operator placement. Hence, traditional algorithms to determine placements that minimise network costs can no longer be used.

We illustrate our research setting with a case study provided by [1], which deals with autonomous transport robots serving machines in a factory. The robots are equipped with various sensors and communicate over wifi. The robots differ in the types of their sensors, as they are assigned different kinds of tasks. Sensor signals and network messages constitute continuous streams of events. Based thereon, queries are evaluated to monitor the conduct of transport jobs. These jobs are announced by machines and distributed among the robots that engage in a bidding process.

In the remainder, Section 2 gives a formulation of our research context. We review related work in Section 3, and present our preliminary results in Section 4. In Section 5, we conclude and outline our research agenda.

2. RESEARCH CONTEXT

We first define a query model for stream processing. We then formalize the problem of operator placement with multiple sinks and elaborate on various dimensions of this problem.

2.1 Query Model

To exemplify our approach, we employ a query language model as used in Complex Event Processing (CEP) [5]. Let $U = E_1, \ldots, E_n$ denote the universe of primitive event types. Each event type E consists of a set of attributes $E = (A_1, \ldots, A_n)$, with, w.l.o.g., A_1 being a timestamp. A query defines a composite event pattern through a set of operators, predicates describing correlation between events, and a time window within which the pattern has to be detected. An operator of a query is either defined by a primitive event type, or composed of other operators. Common composite operators are SEQ, AND, OR and NOT [11]. The AND operator matches, if all input event types occur; the SEQ operator requires all inputs in the specified order; the OR operator matches, if at least one of the inputs occurs; and the NOT operator requires the absence of its input and is typically defined in the context of another operator. An example for the latter is the query SEQ(A a, NOT(B b), C c), which matches if no event of type B occurs between events of types A and C. We write A^{n} as a shorthand for SEQ(A a_1, ..., A a_n).

Consider the above case of autonomous transport robots. A monitoring query detects the following situation: An obstacle is reported (event type Ob) three times (a single observation is not trustworthy due to workers passing the area). Then, a request is issued to tow away the obstacle (Tow_Req). A robot capable of towing obstacles acknowledges the request (Tow_Ack). However, within five minutes the obstacle is still observed at the respective position. This situation hints at overloading of the robots capable of towing away obstacles and may be detected with the following query:

```
SEQ((Ob o)^3 , Tow_Req tq , Tow_Ack ta , Obst o')
WHERE o.position = o'.position
AND o'.timestamp − ta.timestamp ≥ 5min
WITHIN 10min
```

2.2 Problem Statement

A query can be represented as operator tree (neglecting the correlation predicates and time window). This is formalized as a tuple (O, λ), where O is a set of operators with $O_p \subset O$ being the operators defined by primitive event types, and $\lambda : O \to 2^O$ is a function that assigns input operators to an operator (i.e. its children in the tree). A workload Q is a set of queries.

Queries are evaluated in an event-sourced network, which is defined as a tuple (G, f) where G is an undirected graph $G = (V, E)$ with V as a set of nodes and E as a set of communication channels, and $f : V \to 2^U$ as a function that defines the types of events generated at a node. Here we only consider the case that G is complete. Each node generates a local trace, an infinite sequence of events of the respective types that are totally ordered by their timestamps.

Given an event-sourced network (G, f) and a query (O, λ), a placement $p : O \to 2^V$ is a function that assigns each operator to a set of nodes. We refer to a node to which an operator o has been assigned as a *host* of o. Placements are directly lifted to a query workload Q, i.e., $p_w : \bigcup_{(O,\lambda)\in Q} O \to 2^V$ assigns all operators of queries of Q to sets of nodes.

A placement dictates which nodes have to exchange events to evaluate queries: In general, a node n receives events from all nodes that host operators that are inputs for at least one operator hosted at n. The quality of a query placement is assessed by the network costs it inflicts. We define the network costs as the total rate with which events are exchanged. Given a placement p_w of a query workload Q, we denote its total cost by $c(p_w) \in \mathbb{R}$. Based thereon, we capture the general problem addressed by our work as follows:

PROBLEM 1. *Let Q be a query workload and (G, f) a event-sourced network. The problem of* multi-sink operator placement *is to determine a placement p_w for Q such that $c(p_w)$ is minimal.*

It is important to note that unlike traditional approaches we consider to assign an operator to a *set* of nodes – leading to placements with *multiple* sinks.

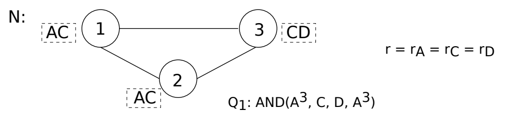
2.3 Problem Dimensions

The problem of *multi-sink operator placement* needs to be addressed in the light of various parameters that arise from an application domain. Below, we provide an overview of this parameter space, along three dimensions: event generation, query processing semantics, and network properties.

2.3.1 Event Generation

Event Rates. To determine an optimal operator placement, knowledge about the rates of event generation is crucial. Intuitively, it is beneficial to host an operator where its input events are generated with high rates. Such knowledge may be available at different granularities, in terms of a *local*, *global*, or *distributed* rate per event type. Under a *local* rate, events of a type are generated with a different rate per node. An example from our case study are events of type BATTERY_LOW generated by a robot, if it needs to charge its battery. The rates at which BATTERY_LOW events are generated depends on the deployed battery and differs for each robot. In contrast, events may be generated with a *global* rate, i.e., the same rate per node. An example are BEACON_MESSAGE events emitted periodically by each robot about its location. Under a *distributed* rate, only the total rate of event generation in the network is known, but not the rates at individual nodes. Consider ACCEPT_JOB_MESSAGE events sent by the robots that win the bidding for transport jobs. While the total rate of these events is known, their distribution among the wining robots is not.

Event Uniqueness. The *uniqueness* of events influences the cost of operator placements: the local event traces of nodes may contain only distinct events or show some overlap (i.e., the *same* event may be generated by more than one node). In our case study, the latter is observed for TEMPERATURE_DROP events: If temperature drops below a threshold, all robots equipped with a thermometer report the situation simultaneously. From an application point of view, there is



	p_1	unrest.	cont.		p_2	unrest.	cont.
①	AND(A^3, C, D, A^3)	-	-	①→③	AND(A^3, C) AND(A^3)	2 x r	2 x r/6
②→①	AND(A^3, C) AND(A^3)	2 x r	2 x r/6	②→③	AND(A^3, C) AND(A^3)	2 x r	2 x r/6
③→①	AND(C,D)	r	r	③	AND(A^3, C, D, A^3)	-	-
		3r	8r/6			4r	4r/6

Figure 2: Depending on the consumption policy different placements yield optimality.

no need to distinguish these reports, so that they are considered as a single event that may be generated at multiple nodes. Clearly, this influences the costs of a placement.

2.3.2 *Query Processing Semantics*

Consumption Policy. Semantics of streaming queries are fine-tuned by a *consumption policy* that dictates how to select events for query matching [3]. For instance, under an *unrestricted* policy, an event may be processed by multiple operators of the same query [2]. In contrast, a *continuous* policy allows each event to be processed by exactly one operator (i.e., the event is consumed by the operator).

Consider Figure 2, which illustrates a network of three nodes generating events of four types (a box next to a node shows the generated event types) with the same (local) rate r and the query Q_1. Under an *unrestricted* policy, the query would match once a set of three A events, a C event, and a D event have been observed in the network. With a *continuous* policy, a total of six A events is required for a match, though. The former policy leads to an event being sent from its generating node to all nodes hosting operators for the respective type. The latter policy, in turn, means that this is not required. Figure 2 also illustrates how the the consumption policy impacts the cost of an operator placement: It shows two placements, $p1$ and $p2$, with the query's sink node being placed at either node 1 or node 3, respectively. Each node optimizes query evaluation and sends partial matches instead of individual events, e.g. in placement p_1, node 2 sends matches for AND(A^3,C) to node 1 instead of all events of type A and C. For an *unrestricted* policy, p_1 yields minimal network costs, whereas p_2 is optimal under a *continuous* policy. Thus, the chosen policy impacts the rates with which A events are sent and leads to different optimal placement.

Selectivity. The cost of operator placement is also influenced by selectivity of an operator, i.e., the portion of events processed by it that fulfill the correlations predicates (see Section 2.1). The selectivity may be estimated based on the distribution of the events' attribute values. As the selectivity of an operator governs the output rate of a node hosting the operator, it influences the costs of a placement.

2.3.3 *Network Properties*

Push-Pull Communication. Instead of requiring nodes to send events to other nodes for processing (known as push-based communication), DSP may also exploit pull-based communication [4]. Then, operators pull their input events from other nodes, which potentially reduces network costs. Both communication paradigms induce space for optimisation, where low frequency events are pushed, while high frequency events are pulled on demand.

Load Balancing. A simple strategy for load balancing in DSP is to restrict the maximum number of operators that can be hosted per node. Yet, this number may be largely detached from the actual computational load (CPU cycles) and networking load (number of received events). To achieve effective load balancing, the types of operators have to be taken into account. Stateful operators, e.g. SEQ, are particularly demanding in terms of computational resources [19].

Constraints on Event Diffusion. The communication within the network might be restricted, e.g. due to privacy or organizational reasons. Such restrictions can be defined on the node level, prohibiting specific nodes to exchange events, or on the event level, prohibiting certain event types to be shared across the network. As a consequence, some nodes might be not eligible to become host for an operator, which leads to a reduced space of possible placements.

3. RELATED WORK

Operator placement has been investigated in the context of distributed stream processing [13–15], distributed CEP [9, 10, 18], and sensor networks [8, 16]. For non tree-structured queries, the problem is known to be NP-hard [7], so that most existing approaches yield heuristic placement algorithms. In [12] and [17], distributed stream processing and CEP applications were surveyed and it was concluded that they differ in their employed system and query model, optimization goal and in whether the placement algorithm is centralized or decentralized. In all of the works mentioned above, the operator placement problem is formulated such that an operator is placed at exactly one node leading to single-sink solutions, which are not suitable for our intent.

Distributed Stream Processing. In DSP, operators are usually considered as black-boxes, so that none of the existing works considers operator semantics. In [14] and [15], placement algorithms are introduced to optimize bandwidth and delay. While different types of event generation can be employed in the proposed cost model, none of the factors we discussed as *network properties* are addressed. Recently, Nardelli et al. [13] introduced several heuristics to compute placements, yet considering solely single-sink solutions.

Distributed Complex Event Processing. In [9, 10, 18], (single-sink) placement algorithms that optimize networking costs are introduced. Chen et al. [9] employ a language model similar to ours and support push-pull communication. However, neither load balancing strategies nor query semantics are taken into account. Cugola et al. [10] discuss placement strategies for CEP queries written in TESLA. While query semantics are incorporated, the approach ignores requirements in terms of load balancing. Starks et al. [18] investigate distributed CEP for mobile ad-hoc networks and propose a decentralized algorithm. They argue that due to the inherent dynamicity of such systems, placements need to be recomputed frequently, which calls for optimizations of the placement algorithm itself in terms of networking costs. Neither load balancing nor push-pull communication are supported by the proposed placement mechanism, though.

Sensor Networks. Srivastava et al. [16] introduce a placement algorithm that is optimal in bandwidth consumption and load balancing. However, their approach is only applicable for operators resembling filters, which strongly limits the query language and simplifies load balancing. In [8]

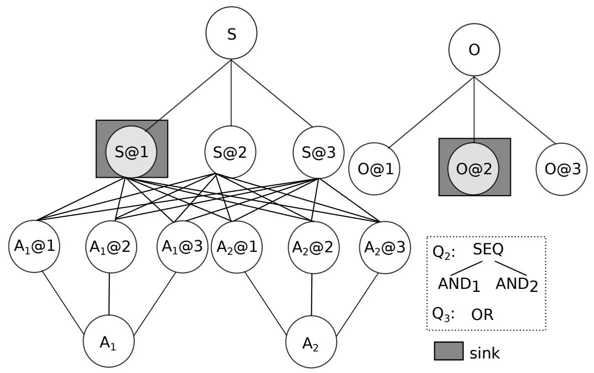


Figure 3: Graphs for the queries Q_2 (left) and Q_3(right) for a network consisting of three nodes.

decentralized, adaptive placement are proposed for continuous queries, minimizing networking costs. Yet, the influence of *network properties* on the placement is neglected.

4. PRELIMINARY RESULTS

As a first step towards an efficient multi-sink operator placement strategy, we adapted the shortest tree algorithm introduced in [6]. The algorithm can be applied for a query having a tree structure and produces an optimal placement for a query workload having a different sink for each query. The idea is to construct a graph such that its vertices reflect all possible placements and the edge weights the respective costs. Using a dynamic programming approach we can efficiently compute a shortest path tree for the graph. The vertices of the shortest path tree yield an optimal placement for which costs are given by the sum of the tree's edge weights. Figure 3 illustrates such a graph for the query Q_2 and a network consisting of three nodes: The vertices are labelled with tuples of operators from $O \setminus O_p$ (i.e. we do not consider primitive events as operators here) and node ids, e.g. the vertex $S_1@1$ reflects the placement of operator S_1 at node 1. The set of vertices describing all placement possibilities of one operator o is called the layer of o, e.g. the vertices $\{S_1@1, S_1@2, S_1@3\}$ form the layer of the SEQ operator. The edges of the graph connect the nodes according to the operator tree: If an operator o is a parent of an operator q in the query graph, then each node of the layer of o is connected to each node of the layer of q. The weight of an edge between two vertices $A_1@1$, $S@2$ reflects the event rates that have to be exchanged in order to place SEQ at node 2 when AND_1 is placed at node 1. The consumption policy, selectivity and event generation affect the rates events are exchanged with and thus can be encoded in the edges weights of the graph. Constrained event diffusion is also supported by the algorithm by adding a pruning step: if two nodes are not allowed to communicate the link weights between two vertices referring to those nodes are set to infinite. For an event type that is not to be shared across the network, a layer of its consuming operator contains only placements at nodes that generate the respective event type.

5. CONCLUSION AND NEXT STEPS

We introduced the multi-sink operator placement problem, thereby paving the way for truly decentralized event stream processing. Moreover, we sketched a first algorithm to incorporate various dimensions of the problem. It yields an optimal placement with a sink for each query of a workload.

As a next step, we plan to extend our approach to support load balancing strategies, the push-pull rationale, and operator reuse between different queries of a workload. Furthermore, we intend to investigate how to decentralize the computation of a placement itself by relying on local information in the neighbourhood of a node, thereby avoiding the necessity of having global knowledge on the network. We expect that decentralizing the placement algorithm favours efficient recomputation of operator placements to react to changes in the network. To cope with dynamicity and failures, we will also explore notions of robustness of a placement.

6. REFERENCES

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

[2] R. Adaikkalavan and S. Chakravarthy. Seamless event and data stream processing: Reconciling windows and consumption modes. In *DASFAA*, pages 341–356. Springer, 2011.

[3] A. Adi and O. Etzion. Amit - the situation manager. *VLDB J.*, 13(2):177–203, 2004.

[4] M. Akdere, U. Çetintemel, and N. Tatbul. Plan-based complex event detection across distributed sources. *VLDB*, 1(1):66–77, 2008.

[5] A. Artikis, A. Margara, M. Ugarte, S. Vansummeren, and M. Weidlich. Complex event recognition languages: Tutorial. In *DEBS*, pages 7–10. ACM, 2017.

[6] S. H. Bokhari. A shortest tree algorithm for optimal assignments across space and time in a distributed processor system. *IEEE TSE*, (6):583–589, 1981.

[7] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli. Optimal operator placement for distributed stream processing applications. In *DEBS*, pages 69–80. ACM, 2016.

[8] G. Chatzimilioudis, A. Cuzzocrea, D. Gunopulos, and N. Mamoulis. A novel distributed framework for optimizing query routing trees in wireless sensor networks via optimal operator placement. *JCSS*, 79(3):349–368, 2013.

[9] J. Chen, L. Ramaswamy, D. K. Lowenthal, and S. Kalyanaraman. Comet: Decentralized complex event detection in mobile delay tolerant networks. In *MDM*, pages 131–136. IEEE, 2012.

[10] G. Cugola and A. Margara. Deployment strategies for distributed complex event processing. *Computing*, 95(2):129–156, 2013.

[11] I. Flouris, N. Giatrakos, A. Deligiannakis, M. Garofalakis, M. Kamp, and M. Mock. Issues in complex event processing: Status and prospects in the big data era. *JSS*, 127:217–236, 2017.

[12] G. T. Lakshmanan, Y. Li, and R. Strom. Placement strategies for internet-scale data stream systems. *IEEE Internet Computing*, 12(6):50–60, 2008.

[13] M. Nardelli, V. Cardellini, V. Grassi, and F. L. PRESTI. Efficient operator placement for distributed data stream processing applications. *IEEE TPDS*, 2019.

[14] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-aware operator placement for stream-processing systems. In *ICDE*, pages 49–49. IEEE, 2006.

[15] S. Rizou, F. Durr, and K. Rothermel. Solving the multi-operator placement problem in large-scale operator networks. In *ICCCN*, pages 1–6. IEEE, 2010.

[16] U. Srivastava, K. Munagala, and J. Widom. Operator placement for in-network stream query processing. In *PODS*, pages 250–258. ACM, 2005.

[17] F. Starks, V. Goebel, S. Kristiansen, and T. Plagemann. Mobile distributed complex event processing—ubi sumus? quo vadimus? In *Mobile Big Data*, pages 147–180. Springer, 2018.

[18] F. Starks and T. P. Plagemann. Operator placement for efficient distributed complex event processing in manets. In *WiMob*, pages 83–90. IEEE, 2015.

[19] H. Zhang, Y. Diao, and N. Immerman. On complexity and optimization of expensive queries in complex event processing. In *SIGMOD*, pages 217–228. ACM, 2014.