Placement-Safe Operator-Graph Changes in Distributed Heterogeneous Data Stream Systems

Niko Pollner, Christian Steudtner, Klaus Meyer-Wegener
Computer Science 6 (Data Management)
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
niko.pollner@fau.de, c.steudtner@gmx.net, klaus.meyer-wegener@fau.de

Abstract: Data stream processing systems enable querying continuous data without first storing it. Data stream queries may combine data from distributed data sources like different sensors in an environmental sensing application. This suggests distributed query processing. Thus the amount of transferred data can be reduced and more processing resources are available.

However, distributed query processing on probably heterogeneous platforms complicates query optimization. This article investigates query optimization through operator graph changes and its interaction with operator placement on heterogeneous distributed systems. Pre-distribution operator graph changes may prevent certain operator placements. Thereby the resource consumption of the query execution may unexpectedly increase. Based on the operator placement problem modeled as a task assignment problem (TAP), we prove that it is NP-hard to decide in general whether an arbitrary operator graph change may negatively influence the best possible TAP solution. We present conditions for several specific operator graph changes that guarantee to preserve the best possible TAP solution.

1 Introduction

Data stream processing is a well suited technique for efficient analysis of streaming data. Possible application scenarios include queries on business data, the (pre-)processing of measurements gathered by environmental sensors or by logging computer network usage or online-services usage.

In such scenarios data often originate from distributed sources. Systems based on different software and hardware platforms acquire the data. With distributed data acquisition, it is feasible to distribute query processing as well, instead of first sending all data to a central place. Some query operators can be placed directly on or near the data acquisition systems. This omits unnecessary transfer of data that are not needed to answer the queries, and partitions the processing effort. Thus querying high frequency or high volume data becomes possible that would otherwise require expensive hardware or could not be processed at all. Also data acquisition devices like wireless sensor nodes profit from early operator execution. They can save energy if data is filtered directly at the source.

Published in: Ritter, N.; Henrich, A.; Lehner, W.; Thor, A.; Friedrich, S.; Wingerath, W. (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW 2015) – Workshopband, Gesellschaft für Informatik (GI), 2015, P-242, 61–70

Problem Statement Optimization of data stream queries for a distributed heterogeneous execution environment poses several challenges: The optimizer must decide for each operator on which processor it should be placed. Here and in the remainder of this article, the term processor stands for a system that is capable to execute operators on a data stream. A cost model is a generic base for the operator placement decision. It can be adapted to represent the requirements of specific application scenarios, so that minimal cost represents the best possible operator distribution. Resource restrictions on the processors and network links between them must also be considered. In a heterogeneous environment costs and capacities will vary among the available processors. The optimizer can optimize the query graph before and after the placement decision. Pre-placement changes of the query graph may however foil certain placements and a specific placement limits the possible post-placement algebraic optimization. For example, a change of the order of two operators can increase costs if the first operator of the original query was available directly on the data source and the now first operator in the changed query is not. This must be considered when using common rules and heuristics for query graph optimization.

Contribution We investigate the influence of common algebraic optimization techniques onto a following operator placement that is modeled as a task assignment problem (TAP). We prove that the general decision whether a certain change of the query graph worsens the best possible TAP solution is NP-hard. We then present analysis of different common operator graph changes and state the conditions under which they guarantee not to harm the best possible placement. We do not study any special operator placement algorithm, but focus on preconditions for graph changes.

Article Organization The following section gives an overview on related work from both the fields of classical database query optimization and data stream query optimization. Sect. 3 introduces the TAP model for the operator placement. It is the basis for the following sections. We prove the NP-hardness of the query-graph-change influence in Sect. 4 and present the preconditions for special graph changes in Sect. 5. The next section shows how to use the preconditions with an exemplary cost model for a realistic query. In the last section we conclude and present some ideas for further research.

2 Related Work

This section presents related work on operator graph optimization from the domains of data base systems (DBS) and data stream systems (DSS). Due to space limitations, we are unfortunately only able to give a very rough overview.

Query optimization in central [JK84] as well as in distributed [Kos00] DBS is a well studied field. Basic ideas like operator reordering are also applicable to DSS. Some operators, especially blocking operators, however have different semantics. Other techniques like optimization of data access have no direct match in DSS. Strict resource restrictions are also rarely considered with distributed DBS because they are not thought to run on highly restricted systems.

The authors of [HSS$^+$14] present a catalog of data stream query optimizations. For each optimization, realistic examples, preconditions, its profitability, and dynamic variants are listed. Among operator graph changes the article also presents other optimizations, like load-shedding, state sharing, operator placement and more. They impose the question for future research, in which order different optimizations should be performed. In the paper at hand, we go a first step into this direction by studying the influence of operator graph changes to following placement decisions. We detail the impact of all the five operator graph changes from [HSS$^+$14]. We think that these changes cover the common query graph optimizations.

The articles [TD03] and [NWL$^+$13] present different approaches to dynamic query optimization. The basic idea is that the order in which tuples visit operators is dynamically changed at runtime. The concept of distributed Eddies from [TD03] decides this on a per tuple basis. It does not take the placement of operators into account. Query Mesh [NWL$^+$13] precreates different routing plans and decides at runtime which plan to use for a set of tuples. It does not consider distributed query processing.

3 Operator Placement as Task Assignment Problem

The operator placement can be modeled as a TAP. Operators are represented as individual tasks. We use the following TAP definition, based on the definition in [DLB$^+$11].

P is the set of all query processors. L is the set of all communication channels. A single communication channel $l \in L$ is defined as $l \subseteq \{P \times P\}$. A communication channel subsumes the communication between processors that share a common medium.

T is the set of all operators. The data rate (in Byte) between two operators is given by $r_{t_1 t_2}, t_1, t_2 \in T$. The operators and rates represent the query graph.

c_{tp} are the processing costs of operator t on processor p. $k_{p_1 p_2}$ gives the cost of sending one Byte of data between processor p_1 and processor p_2. The costs are based on some cost model according to the optimization goal. Since cost models are highly system and application specific, we do not assume a specific cost model for our research of query-graph-change effects. Sect. 6 shows how to apply our findings to an exemplary cost model. [Dau11, 98–121] presents methods for the estimation of operator costs and data rates.

The distribution algorithm tries to minimize the overall cost. It does this by minimizing term (1), considering the constraints (2) - (5). The sought variables are x_{tp}. $x_{tp} = 1$ means that task t is executed on processor p. The first sum in equation (1) are the overall processing costs. The second sum are all communication costs.

$$\min \sum_{t \in T} \sum_{p \in P} c_{tp} x_{tp} + \sum_{t_1 \in T} \sum_{p_1 \in P} \sum_{t_2 \in T} \sum_{p_2 \in P} k_{p_1 p_2} r_{t_1 t_2} x_{t_1 p_1} x_{t_2 p_2} \tag{1}$$

$$\sum_{t \in T} c_{tp} x_{tp} \leq b(p), \forall p \in P \tag{2}$$

$$\sum_{(p_1,p_2) \in l} \sum_{t_1 \in T} \sum_{t_2 \in T} r_{t_1 t_2} x_{t_1 p_1} x_{t_2 p_2} \leq d(l), \forall l \in L \tag{3}$$

$$\sum_{p \in P} x_{tp} = 1, \forall t \in T \tag{4}$$

$$x_{tp} \in \{0,1\}, \forall p \in P, \forall t \in T \tag{5}$$

Constraint (2) limits the tuple processing cost of the operators on one processor to its capacity $b(p)$. Constraint (3) limits the communication rate on one communication channel to its capacity $d(l)$. Constraints (4) and (5) make sure that each task is distributed to exactly one single processor.

Our findings are solely based on the objective function together with the constraints. We do not assume any knowledge about the actual distribution algorithm. There exist different heuristics for solving a TAP. See e.g. [DLB$^+$11] and [Lo88].

4 Generic Operator-Graph-Change Influence Decision

A single algebraic query transformation changes the TAP in numerous ways. For example an operator reordering changes multiple data rates, which are part of multiple equations inside the TAP. When some of those factors increase, it is hard to tell how it affects a following operator placement. The transformed query might even become impossible to execute.

One way to determine the usefulness of a given transformation is to compare the minimum costs of both the original and the transformed query graph. If the transformed query graph has lower or equal cost for the optimal operator placement, i.e. lower or equal minimum cost, than the original query, the transformation has a non-negative effect. $U(Q)$ denotes the query graph that results from applying a change U to the original query Q. Since the operator placement needs to solve a TAP, an NP-complete problem, it is not efficient to compute the placement for each possible transformation. A function `CompareQuery(`Q`, `$U(Q)$`)`, that compares two queries and returns true iff $U(Q)$ has smaller or equal minimal costs than Q would solve the problem.

Sentence. `CompareQuery(`Q`, `$U(Q)$`)` *is NP-hard.*

Definition. *U_{tp} is a transformation that allows task t only to be performed by processor p. All other aspects of $U_{tp}(Q)$ are identical to Q. Both Q and $U_{tp}(Q)$ have equal costs when operators are placed in the same way, i.e. as long as t is placed on p.*

Proof. Given `CompareQuery(`Q`, `$U(Q)$`)` and transformations U_{tp} it is possible to compute the optimal distribution. For each task t it is possible to compare Q and $U_{tp}(Q)$ for each processor p. If CompareQuery(Q, $U_{tp}(Q)$) returns true $U_{tp}(Q)$ has the same minimum cost as Q. Thus the optimal placement of t is p. The algorithm in pseudo code:

```
ComputeDistribution(Q) {
  foreach (t in Tasks) {
    foreach (p in Processors) {
      if (CompareQuery(Q, U_tp(Q)) == true) {
        DistributeTaskProcessor(t, p);
          // makes sure t will be distributed to p
        break; // needed if multiple distributions exist
} } } }
```

ComputeDistribution(Q) calls CompareQuery(Q, $U(Q)$) at most $|T| \cdot |P|$ times. This is a polynomial time reduction of ComputeDistribution(Q). To compute the optimal distribution it is necessary to solve the TAP, an NP-complete problem. This proves that CompareQuery(Q, $U(Q)$) is NP-hard. □

5 Specific Query Graph Changes

While the general determination of a query graph change's impact is NP-hard, it can easily be determined for specific cases. If a transformation neither increases variables used for the TAP nor adds new variables to the TAP, it is trivial to see that all valid operator placement schemes are still valid after the transformation. For the transformed query exist operator placements with lower or equal costs than the original query's costs: the original optimal placement is still valid and has lesser or equal costs.

We establish preconditions for all the five operator graph changes from [HSS$^+$14]. If the preconditions are met the transformation is safe. That means for each valid operator placement scheme of the original query exists a valid scheme for the transformed query with equal costs. So the preconditions especially guarantee that the minimum costs do not rise. However, if heuristic algorithms are used for solving the TAP, they may fail to find an equally good solution for the transformed query as they did for the original query and vice versa, because local minima may change.

Table 1 shows all preconditions at a glance. We now justify why these preconditions hold.

Notation Most of the used notation directly follows from the TAP, especially c_{tp} and $r_{t_1t_2}$. The cost c_{Ap} of an operator A on the processor p depends on the input stream of A and thus on the overall query executed before that operator. Query graph changes affect the input streams of operators an thus also change the costs needed to execute those operators. In order to distinguish between the original and the changed query we use $U(A)$ to indicate the operator A with the applied query change. $U(A)$ and A behave in the same way, but may have different cost, since they work on different input streams. The costs $c_{U(A)p}$ are needed to execute $U(A)$ on p and the following operator t receives an input stream with the data rate $r_{U(A)t}$. In addition r_I denotes the input data stream and r_O denotes the output data stream.

Operator Reordering Operator reordering switches the order of two consecutive operators. The operator sequence $A \rightarrow B$ is transformed to $U(B) \rightarrow U(A)$. In the original query, operator A is placed on processor p_A and operator B on p_B. It is possible that p_A is the same processor as p_B, but it is not known whether both operators are on the same processor, so this cannot be assumed. $p_A = p_B$ would result in a set of preconditions that are easier to fulfill than the preconditions we present. The transformed query can place the operators $U(B)$ and $U(A)$ on any of the processors p_A and p_B.

Case 1: $U(B)$ is placed on p_A and $U(A)$ is placed on p_B. To insure the validity of all distributions, the transformed operators' cost must not exceed the cost of the other original operator, which results in equations (6) and (7). Since the reordering affects the data rates between operators, precondition (8) must hold.

Case 2: Both operators are placed on p_A. This adds an internal communication inside p_A to the operator graph. Equation (9) ensures that internal communication is not factored into the TAP constraints and cost function. The sum of the cost for both transformed operators must be smaller or equal than the cost of A, which is described by equation (10). The changed data rates are reflected in equation (11).

Case 3: Both operators are placed on p_B. This case is similar to case 2 and can be fulfilled with the preconditions given by equations (9), (12) and (13).

Case 4: The remaining option, $U(B)$ is placed on p_B and $U(A)$ is placed on p_A, can be viewed as changed routing. Since the remaining distribution of the query is unknown, the changed routing can be problematic and this option is inherently not safe. It is possible that p_A processes the operator that sends the input to A and that p_B has an operator that processes the output stream of B. In this situation the changed routing causes increased communication cost, since the tuples must be send from p_A to p_B (applying B) to p_A (applying A) to p_B instead of only sending them once from p_A to p_B.

If one of the operators has more than one input stream not all cases can be used. Even if the stream does not need to be duplicated, if A has additional input streams only case 2 is valid. The other cases are not safe anymore, because the transformation changes the routing of the second stream from destination p_A to destination p_B. Similar, if B has additional input streams only case 3 is safe.

Redundancy Elimination This query change eliminates a redundant operator: the query graph has an operator A on two different positions processing the same input stream, duplicated by another operator. This change works by removing one of the instances of A and duplicating its output.

The original query consist of three operators. Operator D (Dup Split in [HSS$^+$14]) is placed on p_D, while an instance of A is placed both on p_1 and p_2. The transformed query

Transformation	Case	Precondition ($\forall p \in P$)	
Operator reordering	Case 1: $U(B)$ on p_A $U(A)$ on p_B	$c_{Ap} \geq c_{U(B)p}$	(6)
		$c_{Bp} \geq c_{U(A)p}$	(7)
		$r_{AB} \geq r_{U(B)U(A)}$	(8)
	Case 2: $U(B)$ on p_A $U(A)$ on p_A	$k_{pp} = 0 \wedge \forall l \in L : (p,p) \notin l$	(9)
		$c_{Ap} \geq c_{U(B)p} + c_{U(A)p}$	(10)
		$r_{AB} \geq r_O$	(11)
	Case 3: $U(B)$ on p_B $U(A)$ on p_B	$k_{pp} = 0 \wedge \forall l \in L : (p,p) \notin l$	(9)
		$c_{Bp} \geq c_{U(B)p} + c_{U(A)p}$	(12)
		$r_{AB} \geq r_I$	(13)
Redundancy elimination	–	$k_{pp} = 0 \wedge \forall l \in L : (p,p) \notin l$	(9)
		$c_{Dp} \geq c_{U(A)p} + c_{U(D)p}$	(14)
Operator separation	–	$k_{pp} = 0 \wedge \forall l \in L : (p,p) \notin l$	(9)
		$c_{Ap} \geq c_{A_1p} + c_{A_2p}$	(15)
Fusion	Case 1: All on p_A	$c_{Ap} \geq c_{Cp}$	(16)
		$r_{AB} \geq r_O$	(17)
	Case 2: All on p_B	$c_{Bp} \geq c_{Cp}$	(18)
		$r_{AB} \geq r_I$	(19)
Fission	–	$k_{pp} = 0 \wedge \forall l \in L : (p,p) \notin l$	(9)
		$c_{Ap} \geq c_{Sp} + c_{Mp} + \sum_{U(A)} c_{U(A)p}$	(20)

Table 1: Preconditions for safe query graph changes that must be fulfilled for all processors. If an operator is not available on some processors, the preconditions can be assumed fulfilled for these processors. It is sufficient that the preconditions of one case are fulfilled.

consists of the operators $U(A)$ and $U(D)$, with $U(D)$ duplicating the output instead of the input. The only possibility to place the transformed query without changing routing is to place both $U(A)$ and $U(D)$ on p_D. The additional internal communication, due to the additional operator on p_D, again forces equation (9). To ensure that any processor can perform the transformed operators, equation (14) is necessary.

In some situations (when p_D is the same processor as p_1 or p_2) the change is safe as long as A does not increase the data rate. But since it is unknown how the operators will be placed, this requirement is not sufficient.

Operator Separation The operator separation splits an operator A into the two operators $A_1 \rightarrow A_2$. Additional internal communication results in precondition (9). Equation (15) ensures that the separated operators' costs are together less than or equal to A's cost.

Fusion Fusion is the opposite transformation to operator separation. The two operators $A \rightarrow B$ are combined to the single operator C (a superbox in [HSS$^+$14]). For the original query A is placed on p_A and B on p_B. The combined operator can be placed on either p_A or p_B. The cost for C must not exceed the cost of p_A or p_B respectively. In addition, the data rates are affected and thus also add preconditions. So either the fulfillment of equations (16) and (17) (if C is placed on p_A) or (18) and (19) (if C is placed on p_B) guarantee the safety of this change. A special case of the fusion is the elimination of an unneeded operator, i.e. removing the operator does not change the query result. Since the redundant operator can change the data rate of a stream (e.g. a filter applied before a more restrictive filter) it still needs to fulfill the preconditions to be safe.

Fission The original query is only the single operator A. Fission replaces A by a partitioned version of it, by applying a split operator S, multiple versions of $U(A)$, which can potentially be distributed across different processors, and finally a merge operator M to unify the streams again. Since it is unknown whether other processors exist that can share the workload profitably, the transformed operators must be placed on the processor that executed the original A. This is safe when preconditions (9) and (20) hold. These equations demand that the combined costs of the split, merge and all parallel versions of $U(A)$ can be executed by all processors with smaller or equal cost than the original A.

6 Application

Given a query and a DSS it is now possible to test whether a specific change is safe. Using an exemplary cost model we examine a simple example query.

Cost Model [Dau11, 91–98] presents a cost model that will be used for the following example. We use a filter and a map operator, which have the following costs:

$$C_{Filter} = \lambda_i C_{Fil} + \lambda_o C_{AppendOut} \tag{21}$$

$$C_{Map} = \lambda_i C_{proj} + \lambda_o C_{AppendOut} \tag{22}$$

C_{Fil} and C_{proj} are the costs associated with filtering respectively projecting an input tuple arriving at the operator. $C_{AppendOut}$ represents the costs of appending one tuple to the output stream. λ_i is the input stream tuple rate, while λ_o is the output stream tuple rate. For those operators λ_o is proportional to λ_i and the equations (21) and (22) can be simplified to $\lambda_i f_{Op}$, where f_{Op} is the cost factor of operator O on processor p for one tuple.

Using these simplified equations, the assumption that the tuple rate is proportional to the data rate and costs and selectivities are non-zero, equations (6) to (8) can be rewritten as:

$$\lambda_I f_{Ap} \geq \lambda_I f_{Bp} \qquad \Leftrightarrow \qquad \frac{f_{Ap}}{f_{Bp}} \geq 1 \tag{23}$$

$$\sigma_A \lambda_I f_{Bp} \geq \sigma_{U(B)} \lambda_I f_{Ap} \qquad \Leftrightarrow \qquad \frac{\sigma_A}{\sigma_{U(B)}} \geq \frac{f_{Ap}}{f_{Bp}} \tag{24}$$

$$\sigma_A \lambda_I \geq \sigma_{U(B)} \lambda_I \qquad \Leftrightarrow \qquad \frac{\sigma_A}{\sigma_{U(B)}} \geq 1 \tag{25}$$

The equations for the other two cases shown in table 1 can be similarly rewritten. Equations (23) to (25) show that there are relatively few values to compare: We need the ratio of the operator selectivity and for each processor the ratio of operator costs.

Example We examine the simple query of a map operator M followed by a filter F applied on a stream containing image data monitoring conveyor belts transporting freshly produced items. The query supports judging the quality of the current production run. Operator M classifies each tuple (and thus each observed produced item) into one of several quality classes and is rather expensive. F filters the stream for one conveyor belt, because different conveyor belts transport different items and are observed by different queries.

M does not change the data rate of the stream. It simply replaces the value *unclassified* already stored inside the input stream for each tuple with the correct classification and thus has a selectivity of 1. There are multiple types of processors available inside the production hall. Depending on the processor type the ratio $\frac{f_{Mp}}{f_{Fp}}$ differs quite a bit, but overall M is more expensive: this ratio fluctuates between 2 and 10. Equations (23) to (25) show that the selection push down is always safe if $\sigma_{U(F)}$ is smaller or equal than 0.1: In this case it is always possible that the two operators switch their places without violating additional constraints of the TAP. If $\sigma_{U(F)}$ is greater than 0.1 this change is not necessarily safe. It is possible that the preconditions of one of the other two cases (both operators on the same processor) are fulfilled or another good distribution is possible, but the latter cannot be tested in a reasonable time as we discussed in Sect. 4.

7 Conclusion

We presented our findings on the interaction between optimization through query graph changes and the placement of operators on different heterogeneous processing systems. We first motivated our research and defined the problem. Existing work on query optimization through operator graph changes in the context of DMS and DSS was presented, none of which studied the interaction with operator placement. The next section presented the TAP model of the distribution problem. We showed that it is NP-hard to decide in general if an arbitrary query graph change can negatively influence the best possible operator placement scheme. Based on a selection of common query graph changes from the literature, we deduced preconditions under which operator placement does not mind the changes. The last section showed the application of our findings with an exemplary cost model for a realistic query.

The preconditions for safe operator graph changes are quite restrictive. They severely limit the possible changes if followed strictly. As with general query optimization, development of heuristics to loosen certain preconditions seems promising. The preconditions presented in this article are the basis for such future work. Another interesting field is the direct integration of query graph optimization in the usually heuristic distribution algorithms. Distribution algorithms could be extended to consider query graph changes in addition to the operator placement. We plan to investigate these ideas in our future research.

References

[Dau11] M. Daum. *Verteilung globaler Anfragen auf heterogene Stromverarbeitungssysteme*. PhD thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 2011.

[DLB$^+$11] M. Daum, F. Lauterwald, P. Baumgärtel, N. Pollner, and K. Meyer-Wegener. Efficient and Cost-aware Operator Placement in Heterogeneous Stream-Processing Environments. In *Proceedings of the 5th ACM International Conference on Distributed Event-Based Systems (DEBS)*, pages 393–394, New York, NY, USA, 2011. ACM.

[HSS$^+$14] M. Hirzel, R. Soulé, S. Schneider, B. Gedik, and R. Grimm. A Catalog of Stream Processing Optimizations. *ACM Comput. Surv.*, 46(4):1–34, 2014.

[JK84] M. Jarke and J. Koch. Query Optimization in Database Systems. *ACM Comput. Surv.*, 16(2):111–152, 1984.

[Kos00] D. Kossmann. The State of the Art in Distributed Query Processing. *ACM Comput. Surv.*, 32(4):422–469, 2000.

[Lo88] V. M. Lo. Heuristic Algorithms for Task Assignment in Distributed Systems. *IEEE Transactions on Computers*, 37(11):1384–1397, 1988.

[NWL$^+$13] R. V. Nehme, K. Works, C. Lei, E. A. Rundensteiner, and E. Bertino. Multi-route Query Processing and Optimization. *J. Comput. System Sci.*, 79(3):312–329, 2013.

[TD03] F. Tian and D. J. DeWitt. Tuple Routing Strategies for Distributed Eddies. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, VLDB '03, pages 333–344. VLDB Endowment, 2003.