On the Complexity of Multi-Query Optimization in Stream Grids

Saikat Mukherjee Srinath Srinivasa Krithi Ramamritham

International Institute of Information Technology
Bangalore, India
{saikat.mukherjee,sri}@iiitb.ac.in

Indian Institute of Technology
Bombay, India
krithi@cse.iitb.ac.in

Abstract

Stream grids are wide-area grid computing environments that are fed by a set of stream data sources. Such grids are becoming more wide-spread due to the large scale deployment of sensor networks for a wide range of applications, from monitoring geophysical activities to supply chain management coupled with applications like network monitoring. Queries external to the system arrive on any node in the grid seeking data from one or more data streams. The kind of queries considered in this work are (1) lifetime queries and (2) long running queries where new query arrivals and query revocations are infrequent. From the system perspective, computing the optimal query plan for the set of queries incident on the grid would ensure minimal system-wide resource usage, thereby maximizing the number of concurrent queries that can be supported. The key challenge in such a system is multi-query optimization. In this work, we analyze the complexity of multi-query optimization for select, project and join queries in isolation and propose algorithms for computing optimal query plans if polynomial time algorithms exist.

1 Introduction

Stream grids are grid computing environments that are fed with streaming data sources from instrumentation devices like cameras, RFID (radio-frequency identification) sensors or other applications. Queries by users or applications seek to tap into one or more such streams. From the system perspective, the important optimization goal is reduced bandwidth consumption which can be achieved by efficient routing of data streams.

Queries in such grids may originate on any node and seek data from any stream or a set of streams. Such queries are typically long lived, but not necessarily infinitely long lived. Traditionally, query optimization has been addressed for two classes of queries: “one-shot” queries and infinite or “standing” queries [6]. One-shot queries are transient in nature and have very short life spans. In such environments, the speed of query processing takes precedence over computing the execution plan with optimal data stream routing. On the other hand, for standing queries whose lifetimes are practically infinitely long and systems where new query arrivals and query revocations are very infrequent, it is desirable to invest time and resources to obtain optimal execution plans.

The problem of generating globally optimal query plans has been considered earlier in the context of databases [26, 10, 15], data warehousing [21] and more recently in the context of streaming data sources like sensor networks [19, 32]. While, the primary challenge in computation of optimal query plans in databases have been joins, in the data warehousing context it has been the re-use of materialized views and in the streaming data context, aggregate queries.

The key results from the research into complexity of query optimization can be summarized as, (a) the optimal join ordering problem in distributed databases is NP hard [31], (b) the optimal materialized view selection problem using selection granularities in data warehousing is NP hard [21], and (c) the problem of minimizing communication cost is NP-hard for *max* and *min* queries [30]. While in the context of databases, considering selects, projects and joins as part of a single query makes sense, for streaming data, the possibility of data sharing introduces use cases where project, select and join queries can be required in isolation.

In this work, we consider systems which may require project only, select only, and join only queries thereby necessitating a re-look at the complexity of multi-query optimization for such individual query types. We show that for project queries, polynomial time algorithms exists for computing globally optimal query plans. We also show data sharing coupled with infinite data sets allow projection results to be composed from a number of sources, with each source having a subset of data required to answer a query. Finally, by using a variation of the traditional two step optimization process [9] involving a) data source selection and access paths computation at runtime and b) operator site selection at compile time, we show that for a particular class of queries, it is possible to achieve minimum communication costs.

15th International Conference on Management of Data
COMAD 2009, Mysore, India, December 9–12, 2009
©Computer Society of India, 2009

2 Related Work

Query optimization in databases is a very well studied area in data management. Original query optimizers search the plan space using dynamic programming [26], applying a number of heuristics to reduce the number of options and ensuring tractable optimization. One of the key observations made in [26] was the notion of pushing down projections and sometimes selections in the query tree to reduce the data transfer between operators.

The focus of subsequent research was primarily on optimizing joins [17, 10, 16, 25] and plan enumeration with other operators in Starburst [15] and Volcano [13]. Cost models for optimization using resource consumption as a metric were described in [18]. The inability of resource consumption models to incorporate operator parallelism led to the development of response time models as described in [10].

A framework for determining the complexity of a general class of distributed query processing is discussed in [31], while the NP hard nature of optimal materialized view selection based on selection granularities is discussed in [21].

The concept of data, query and hybrid shipping were introduced in SHORE [5] and has evolved as the operator placement problem in network aware query processing [1, 22]. The two step optimization process where query plans are generated in parts at compile time and runtime is discussed in [4, 9].

Multi-query optimization in [27, 24, 29] provides heuristics for computing the query plan, while [14, 7, 12] discuss issues with scheduling, pipelining and caching techniques in multi-query optimization. The NP hard nature of computing multi-query optimization in databases is discussed in [28].

Sensor networks [19] and in network query processing [32] primarily focus on aggregate query optimization to maximize the energy efficiency of sensors. The complexity of multi-query optimization for aggregate queries in sensor network is evaluated in [30].

Network aware query processing techniques described in [1, 22] focus on the correct placement of operators in the network. A spring relaxation technique to place operators in the network integrating the two step optimization process into a single step optimization process is also introduced in [22].

3 The Grid Model

Mathematically, the stream grid is modeled as: $\mathcal{G} = (X, d)$, where X represents all the grid nodes. A subset of the grid nodes, $S \subseteq X$ are also stream sources. $d : X \times X \rightarrow \Re^+$ is a distance function encapsulating latency between nodes. The distance function is assumed to have the following characteristics:

- $\forall x \in X, d(x,x) = 0$, and
- Triangle inequality:

$$\forall x, y, z \in X, x \neq y \neq z, d(x,z) \leq d(x,y) + d(y,z)$$

It is important to note that the distance function d represents the latency incurred by the *best path* between pairs of nodes. In this sense, even though the triangle inequality doesn't hold for packet routing on the Internet [2], it still holds for the distance function. The space described is a logical space, which need not directly correspond to any geography and/or network topology.

Stream data is considered to be in the form of tuples, with each tuple representing a row in an infinitely long table.

4 Query Types

Queries may arrive on any node in the grid requesting for one or more streams. Queries are represented as relational algebra expressions over the data streams. For the purposes of grid-level optimization, we consider three basic relational operations: projections, selections and joins.

1. Projection query: $q = \pi_{s_{i1},\ldots s_{i_k}}(S_i)$
2. Selection query: $q = \sigma_{(condition)} S_i$
3. Join query: $q = S_i \bowtie S_j$

At any given grid node $x \in X$ a subset of one or more streams may be available as part of current query execution plan. These streams can be reused to serve other queries in the vicinity without them having to go all the way to the required stream sources. A grid node which is the source of a data stream is termed as a "primary source" and a grid node providing data which is derived or processed from another grid node is termed as a "secondary source."

5 Optimization Objective

From the user perspective, the key optimization goal is to ensure reduced response times or latency, while from the system perspective, the key objective is to reduce the required bandwidth. To ensure minimal response times, each query would need to be satisfied by connecting to the nearest node(s) having the required data (minimum d). It should be noted here that minimizing network latency may result in increased bandwidth usage if a geographically distant node provides low latency. However, once the data source node(s) are identified, the bandwidth required can be reduced by optimal operator placement. A bandwidth-delay product combines both requirements and is termed as *network usage* in this work. The optimization objective is to minimize network usage.

The other parameter which influences response time is load on a node. Nodes with heavy loads would be a bottleneck increasing the overall response time of queries. The load on a node is a combination of communication and computational load. In the work presented in this paper, we do not evaluate the complexity query optimization considering the load on a node.

Network Usage: At any node $x \in X$, given a query q, it is ultimately answered by returning a set of *stream links* $L(q) = \{l_1, l_1, l_2, \ldots, l_n\}$. For instance in Figure 1, a query on node CN_1 requesting for $s_1 \bowtie s_2 \bowtie s_3$ can be answered by forming the stream link set $\{l_1, l_2, l_3\}$.

A link is a directed edge, represented as an ordered pair, $l_p = (x_p, y_p)$. Data flows from data source y_p to destination x_p, to satisfy in part or completely, a query at x_p. In Figure 1, the link l_1 would be represented as (CN_1, SN_1).

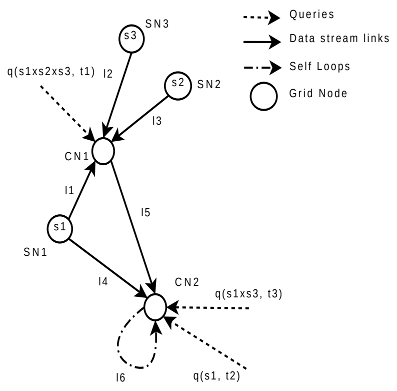

Figure 1: Query Result Generation using Streams Links

For a link $l_i = (x_i, y_i)$, its *network usage* is given as,

$$u(l_i) = Bandwidth(l_i) \cdot d(l_i) \tag{1}$$

where, $Bandwidth(l_i)$ is the data rate of the stream l_i, and $d(l_i) = d(x_i, y_i)$ as described earlier, is the latency of the data stream.

Let Q be the set of all queries incident on the grid $\mathcal{G}$ at any instance of time. Let $\mathcal{L}$ denote the set of all links that have been returned in response to queries in Q. We refer to $\mathcal{L}$ as the "Estuary graph" or the "link graph" of the grid $\mathcal{G}$ for the present time. The Estuary graph is formally defined as:

$$\mathcal{L} = \bigcup_{q \in Q} L(q) \tag{2}$$

The global optimization objective on network usage is to obtain an Estuary graph such that the overall network usage is minimized. This is stated as:

$$\arg\min_{\mathcal{L}} \sum_{l_i \in \mathcal{L}} u(l_i) \tag{3}$$

6 Components of Network Usage Optimization

From Equation 1 it is evident that there are two parts to the optimization process, a) correct data source selection leading to minimal d, and b) correct operator placement reducing the bandwidth required to transfer a data stream from one node to another.

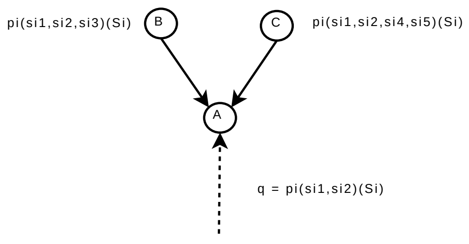

Figure 2: Non Composable Projection

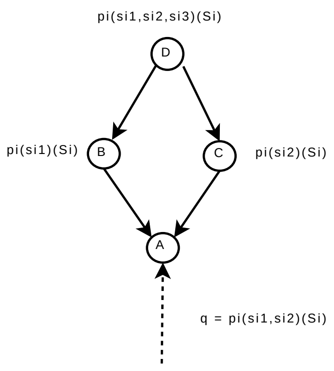

Figure 3: Projection Composition

6.1 Data Source Selection

Correct data source selection depends on the type of query plans supported in the system. The two types of plans we consider in this work are, *composable* and *non-composable* plans.

Non-Composable Query Plans

Consider a grid node $x \in X$ processing a query of the form $q_x = \pi_{s_{i1}, s_{i2}}(S_i)$. This query can be answered by any node $x' \in X$, that contains a stream covering the requirements of q_x. In other words, a stream that contains either the same or a superset of all the attributes of S_i that is required by q_x.

This is shown in Figure 2 where a query on node A is answered by either node B or node C. A query plan that always taps into such covering sources is termed as a "non-composable" plan.

Composable Query Plans

Alternatively, q_x can also be answered by sourcing streams from two or more grid nodes even when none of them individually cover q_x, as long as the union of all the attributes sourced from the streams covers q_i. Such a kind of query plan is called a "composable" query plan.

This is shown in Figure 3, where node D is the source of the data S_i with attributes s_{i1}, s_{i2} and s_{i3}. Node B and C have $\pi_{si1}(S_i)$ and $\pi_{si2}(S_i)$ respectively and a query q arrives on node A for $\pi_{si1,si2}(S_i)$. The query on node A is satisfied using subset data available both at node B and C.

$D(S_i)$	$B = \pi_{si1}(S_i)$	$C = \pi_{si2}(S_i)$	$B \cup C = \pi_{si1,si2}(S_i)$
..	..	..	..
(1,1,1)	(1)	(1)	(1,1)
(1,1,2)	(1)	(1)	(1,1)
(1,2,3)	(1)	(2)	(1,2)
(2,3,4)	(2)	(3)	(2,3)
..	..	..	..

Table 1: Continuous Data Projection Composition Possibility - No Duplicate Removal

It should be noted here that *composable query plans are not possible for project operators in traditional relational algebra working on finite data sets*. However, it is possible to compose query results for projection operations in streaming data, if duplicates are not removed from the data stream.

Duplicate removal in stream data itself has been addressed using various techniques like buffering [11], Bloom filters [20] and [8] and windowing techniques [3]. If such duplicate removal techniques are used, query result composition would not be possible. For many practical applications involving aggregations of data elements (like counting or computing averages over the query result) duplicate removal is not advisable. These kinds of queries are amenable to be answered by composable query plans.

Composability is possible with selects as well. A given select operator σ can be answered by two or more streams $\sigma_1 \ldots \sigma_k$ that each have a smaller selectivity than σ as long as the combined selectivity of $\sigma_1 \cup \cdots \cup \sigma_k$ *covers* the selectivity of σ. Composing query results based on selections is similar to computing a query over a larger table from two or more smaller materialized views [21]. In contrast, a query representing a join between two or more streams has to be always composed from the different streams.

Hence, given a query q comprising a single operation (either select, project or join), a query plan can compute the result in three different ways:

1. Fetch the data from the relevant node hosting the data stream or the *primary source*.

2. Fetch the data from a *secondary source* which can satisfy the query. A secondary source is a node which shares data fetched from another node. A secondary source satisfies q if the streams it hosts covers the selectivity or attribute requirements for q.

3. Compose the query result using two or more sources.

While composability allows for query plans resulting in lesser network usage than non-composable operations, it also adds an extra layer of complexity. By allowing compositions, determining the optimal query plan not only involves identifying single sources which can satisfy the query, but also consider *all possibilities* where a combination of sources could satisfy the query.

6.2 Operator Placement

The notion of operator placement has been discussed with respect to query tree optimization in databases [R*], data and query shipping and network aware query processing. For the SPJ queries considered in this work, we consider the correct placement of operator resulting in minimal network usage for individual project, select and join queries for composable and non-composable query plans discussed in the previous section. To decide on operator placement, each operator O, a *selectivity(O)* parameter is used and is defined as,

$$selectivity(O) = \frac{b_{O(result)}}{\sum_{i \in \mathcal{I}} b_{(DS_i)}} \qquad (4)$$

where, $\mathcal{I}$ is the set of input streams to the operator O, $b_{(DS_i)}$ is the bandwidth required to transmit the i^{th} input stream in $\mathcal{I}$, $b_{O(result)}$ is the bandwidth required to transmit the resultant data stream.

The complexity of query optimization is the combined complexity of source selection coupled with operator placement. If any of the two parts of the plan result in non-polynomial time algorithms, heuristics can be used.

7 Complexity of Network Usage Optimization

For the purpose of computing the complexity of computing a globally optimal plan, leading to minimal network usage, at any instance, we assume that the system is frozen on any query arrival or revocation until the plan is computed. In other words, query plan computation is a system-wide atomic step.

We now consider each of the operations (projects, selects and joins) separately for complexity calculation.

7.1 Complexity of Projections

The complexity of projection queries for composable and non-composable query plans are as follows.

7.1.1 Composable Projects

Data Source Selection: To find the optimal query plan for a given query set, we re-write all project queries requesting for multiple attributes of a stream to single attribute queries (SAQ) of the same stream. For instance, in Figure 3, the SAQ for query q_A at node A is given as, $SAQ(q_A) = \{\pi_{si1}(S_i), \pi_{si2}(S_i)\}$.

In a graph theoretic sense, all nodes requiring a given stream data attribute collectively form a directed acyclic graph (DAG) with at least one node connected to the original data source. For instance the minimum spanning tree (MST) overlays for the example in Figure 3 are given in Figure 4 where, a MST is the tree incurring minimal network usage that spans across all nodes in the DAG.

A query at a given node is satisfied by combining the required attributes from individual data streams. For instance in Figure 3, the query at node A is satisfied by taking a "union" the streams $\pi_{si1}(S_i)$ and $\pi_{si2}(S_i)\}$ arriving at node A as shown in Table 1.

To find the DAG with the minimal network usage, we use the following rationale:

1. Ignore the direction of DAG edges and consider all stream connections between all pairs of nodes in the grid
2. Compute a MST for the grid based on the stream connections.
3. The overlay with the minimum network usage between a given source and destination is the path between them that lies on the MST.

Algorithm 1 Minimum Spanning Tree Overlay Algorithm

```
Require: Grid $\mathcal{G}$ and project query set $Q$ incident on $\mathcal{G}$
Ensure: Overlay of MSTs $M$ with minimum network usage $U_{min}$
1: $S \leftarrow \{\}, N \leftarrow \{\}, M \leftarrow \{\}, U_{min} \leftarrow 0$
2: for all $q \in Q$ do
3:     $S = S \cup SAQ(q)$
4: end for
5: for all $SAQ_i \in S$ do
6:     $N_i = \{x : x \in X \wedge \exists q_x : SAQ(q_x) \supseteq SAQ_i\}$
7:     $N = N \cup N_i$
8: end for
9: for all $N_i \in N$ do
10:    $M_i = MST(N_i, \mathcal{G})$
11:    $M = M \cup M_i$
12: end for
13: for all $M_i \in M$ do
14:    $U_{min} = U_{min} + U(M_i)$
15: end for
```

The overall algorithm as shown in Algorithm 1 is explained as follows. Each query $q \in Q$ incident on the grid is decomposed into individual SAQs required to satisfy q using the $SAQ(q)$ operator. These individual SAQs are then added to a set S which contains all the unique, individual SAQs required to satisfy all the queries Q incident on the grid. For each $SAQ_i \in S$, the set of grid nodes $N_i \in \mathcal{G}$ which require SAQ_i to satisfy some query incident on it are identified. N is the set of all N_is corresponding to each SAQ_i. All nodes in N_i and the source of SAQ_i are connected together to create an overlay of edges M_i using a minimum spanning tree algorithm $MST(N_i, \mathcal{G})$. The network usage for the overlay M_i is given by $U(M_i)$ and the network usage for the set of all overlays M results in the minimum network usage U_{min}.

Theorem 1 *An overlay of minimum spanning trees M as computed by Algorithm 1 gives the minimum network usage query plan U_{min} for a set of stream projection queries Q if compositions are allowed.*

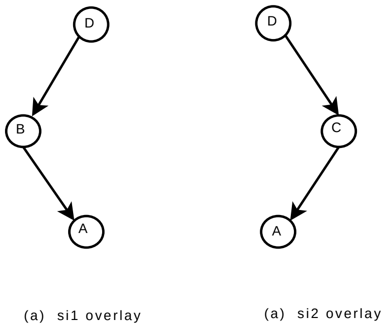

Figure 4: Minimum Spanning Tree Overlays

Proof *We prove the above theorem by refutation. Consider one of the MST overlays M_i, requiring minimal access paths over the set of nodes N_i. Suppose there exists another topology M_i' to connect nodes in N_i with a network usage $U(M_i')$ such that $U(M_i') < U(M_i)$. This would mean that if we replace the overlay path in M_i with M_i' we would get a spanning tree of smaller weight. This is a contradiction since M_i is the minimum spanning tree.*

Thus if compositions are allowed, then the optimal query plan complexity is polynomial time with the optimal query plan being an overlay of minimum spanning trees, as the complexity of computing each minimum spanning tree is polynomial [23].

Operator Placement: The project operator is not used at all for composable projects. Instead for each query q at a node, the required data streams are fetched from other nodes and a union operator used to combine the individual streams to get the required result. Since the selectivity of the union operator is unity, there is no possibility of reducing the bandwidth required to transmit the data by "better" operator placement.

7.1.2 Non Composable Projects

Data Source Selection: If compositions are not allowed, the only way to satisfy a query request is to get it from either the primary source or a secondary source which has a superset of the required data.

To determine the set of possible sources to answer a query, we check for the covering property using the SAQ concept introduced earlier. A query q_x on node x can be *covered* or *satisfied* by either the source of the data, or another node y which answers a query q_y where $SAQ(q_x) \subseteq SAQ(q_y)$.

The set of all sources and queries can be represented as a poset of covering hierarchy based on the streams that they possess or require. The covering poset for the example in Figure 3 is shown in Figure 5. Node D being the source can satisfy any query and is therefore at the top of the poset and at level 0. The query at node A requiring both $\pi_{si1}(S_i)$ and $\pi_{si2}(S_i)$ is next and can be satisfied only by source D. Queries at nodes B and C can be satisfied by both node A

and source node D and are hence at the highest level of the poset.

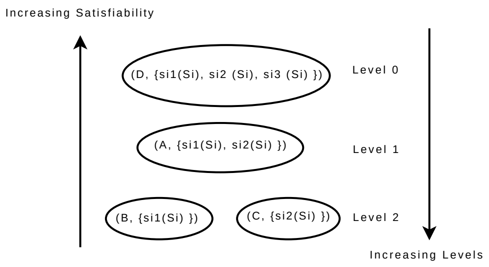

Figure 5: Satisfiability Poset for Figure 3

Each poset element is represented as e_i, where i uniquely identifies the poset element. A function $L(e_i)$ is defined to denote the *level* of poset element i and $SAQ(e_i) \sqsubseteq SAQ(e_j)$ indicates the satisfiability of poset element e_i by another poset element e_j.

The hierarchy of the poset ensures if $SAQ(e_i) \sqsubseteq SAQ(e_j)$ or $L(e_j) \leq L(e_i)$. A source node $s \in S$ can answer a query for any attribute related to the source and hence poset elements corresponding to sources are placed at the top most level. All poset elements representing queries are hierarchically organized below the source elements. Algorithm 2 explains the poset hierarchy formation process.

Algorithm 2 Poset Hierarchy Formation Algorithm

```
Require: Grid $\mathcal{G}$ and project query set $Q$ incident on $\mathcal{G}$
Ensure: Hierarchically ordered poset $P$
1:  for all $e_i \in P$ do
2:      if $e_i \in S$ then
3:          $L(e_i) = 0$
4:      else
5:          $L(e_i) = 1$
6:      end if
7:  end for
8:  repeat
9:      $finish \leftarrow$ true
10:     for all $e_i \in P$ do
11:         for all $e_j \in P$ do
12:             if $SAQ(e_i) \sqsubset SAQ(e_j)$ then
13:                 $L(e_i) = max[L(e_i), (L(e_j) + 1)]$
14:                 $finish \leftarrow$ false
15:             end if
16:         end for
17:     end for
18: until $finish$
```

Lemma 2 *At the end of the poset hierarchy formation process, the i^{th} poset element e_i at level $k = L(e_i)$ can only be satisfied by,*

1. *poset element e_j at level k if and only if $SAQ(e_i) = SAQ(e_j)$*
2. *poset element e_j at level $l = L(e_j)$, where $l < k$, if and only if $SAQ(e_i) \sqsubset SAQ(e_j)$*

Proof *We prove this by refutation. Assuming there exists a poset element e_j at level $l > k$ which can satisfy e_i, then either (a)$SAQ(e_i) \sqsubset SAQ(e_j)$, or (b) $SAQ(e_i) = SAQ(e_j)$.*

- *Refutation for (a): If there exists some e_j such that $SAQ(e_i) \sqsubset SAQ(e_j)$, then from $line$ 13 of Algorithm 2, $k \geq l + 1$. Hence if $k < l$, then such an e_j cannot exist.*
- *Refutation for (b): If e_j is at level l there must be some e_q at level $l - 1$ such that $SAQ(e_j) \sqsubset SAQ(e_q)$. If $SAQ(e_j) = SAQ(e_i)$, then $k = l$ as $SAQ(e_i) \subset SAQ(e_q)$. Hence if $k < l$, then such an e_j cannot exist.*

Once the poset P is ordered according to satisfiability, we now create a "minimum network usage graph" $MinGraph$. To create the $MinGraph$, each poset element is considered as a node in the graph and the set of edges determined ensuring minimum network usage. From Lemma 2 all poset elements requiring the same data are at the same level l and are grouped into a set X and referred to as the destination nodes. The poset elements or nodes which can satisfy the poset elements in set X are in lower levels and are grouped together into set Y or the source nodes. To determine the edges resulting in the minimum network usage, Prim's minimum spanning tree algorithm [23] is used where Y is considered to be the set of nodes which are already in the tree and X is the set of nodes still requiring to be connected.

Algorithm 3 determines the set of edges resulting in minimum network usage.

Algorithm 3 Minimum Cost Network Usage Graph

```
Require: Grid $\mathcal{G}$ and ordered poset $P$
Ensure: $Edges$ of $MinGraph$
1:  $Edges \leftarrow \{\}$
2:  for all $e_i \in P$ do
3:      $X \leftarrow \{\}$ $\{X$ is the set of destination nodes$\}$
4:      $Y \leftarrow \{\}$ $\{Y$ is the set of source nodes$\}$
5:      for all $e_j \in P$ do
6:          if $(L(e_j) = L(e_i)$ & $SAQ(e_i) = SAQ(e_j))$
            then
7:              $X = X \cup e_j$
8:          end if
9:          if $(L(e_j) < L(e_i)$ & $SAQ(e_i) \subset SAQ(e_j))$
            then
10:             $Y = Y \cup e_j$
11:         end if
12:     end for
13:     $Edges = Edges \cup PrimsMST(X, Y, \mathcal{G})$
14: end for
```

Theorem 3 *The set $Edges$ determined using Algorithm 3 results in minimum network usage.*

Proof *We use proof by refutation to prove the above algorithm. The incorrectness can arise from,*

- *Incorrect selection of source set: Incorrect source selection can occur if any possible source is being not considered while considering the best source to select. Given $line$ 9 of Algorithm 3, if there is such a source present, it must be represented by a poset element with a level greater than the concerned poset element. However this is not possible because of Lemma 2.*
- *Incorrect selection of destination set: Incorrect destination selection can occur if any destination requiring the same data is being not considered. $Line$ 6 of Algorithm 3 ensures that all equal sources are considered.*
- *Incorrect selection of edge: If there is an incorrect edge selected, then there exists another edge with lesser weight than the selected edge. This is not possible because of the use of Prim's algorithm which selects the minimum cost edge.*

Operator Placement: The project operator is always placed on the data source as $selectivity(\pi) \leq 1$.

7.2 Complexity of Selections

Like project queries, we consider the complexity of composable and non-composable selection query plans.

7.2.1 Composable Selects

Data Source Selection: The main issue in selection queries involving compositions is identifying the set of data sources which would lead to the minimal network loss. The quintessential notion which determines if a source can serve a query is the *selection granularity* available at the source and the selection granularity required by the query. For instance a query $q_1 = \sigma_{(b1=1\&b2=5)}S_i$ can be answered by composing the result from two secondary data sources having data $\sigma_{(b1<3)}S_i$ and $\sigma_{(b2>4)}S_i$.

Using selection granularities to determine reuse of data is a well studied in the area of materialized view selection techniques in data-warehouses [21].

In [21], for a given query Q, there are a set of candidate materialized views $V(Q)$ to satisfy Q and a cost function $cost(MV_i, QR_i)$ which provides the cost for a materialized view $MV_i \in V(Q)$ with a query region of QR_i. The optimal MV set problem is to find an optimal set S of pairs (MV_i, QR_i) which can answer query Q, minimizing the cost of S or,

$$\arg\min_{S} \sum_{(MV_i,QR_i)\in S} cost(MV_i, QR_i) \qquad (5)$$

[21] shows that the minimum set cover decision problem, which is NP-complete can be transformed in polynomial time to this decision problem thereby rendering the optimization version as NP-hard.

We map the optimal MV set problem to the problem of identifying the correct set of sources to satisfy a query q incident on the grid. The candidate sources and secondary sources $S(q)$ for answering the query can be considered to be the set of candidate MVs $V(Q)$. The cost for a materialized view can be considered to be the network usage $U(s_i, r_i)$ for fetching data from the node $s_i \in S(q)$ with selection granularity r_i. The optimal network usage problem is to find the optimal set S of pairs (s_i, r_i) to minimize the network usage of S or,

$$\arg\min_{S} \sum_{(s_i,r_i)\in S} U(s_i, r_i) \qquad (6)$$

Hence the optimal network usage problem is NP-hard as well.

7.2.2 Non Composable Selects

Data Source Selection: If compositions are not allowed, the problem becomes very similar to the projection without compositions problem. Since queries can be answered from only sources with higher selection granularities, a single query stream is sufficient to answer the query. In such a scenario we need to create a hierarchical poset for a given query set using selection granularities to set the levels. The rest of the algorithm will be the same as in projection queries.
Operator Placement: The select operator is always placed on the data source as $selectivity(\sigma) \leq 1$.

7.3 Complexity of Joins

Operator Placement: In a distributed environment with data being available at different sites, a join query with n relations is formulated as a graph problem [31]. A directed graph with $n + 1$ nodes is constructed where one node corresponds to the final destination site D and the remaining n nodes have a one-to-one association with a relation. An edge (R_i, R_j) indicates relation R_i being sent to node with relation R_j to perform a join. An edge (R_i, D) indicates relation R_i being sent to the destination site directly. The objective is to find an inversely directed spanning tree toward D with the minimal transmission cost. Finding the optimal join sequence to minimize the transmission cost is NP hard [31]. By replacing the transmission cost with the network usage associated with shipping relations between nodes, our problem also becomes NP-hard.

Trivially, joins cannot be performed without compositions as no single source will have the joined data.

8 Conclusion and Future Work

This work shows that when composable query plans are not allowed, polynomial time algorithms exists for computing globally optimal non-composable plan for selects and projects. However, the resultant network usage is not the minimum possible value. Better query plans involving lesser network usage are possible when selects and projects are composed from streams with lower resolution. This however, makes the optimization problem intractable except for project-only queries.

Conditions	Projects	Selection	Joins
Is query result possible without composition?	Yes	Yes	No
Complexity of globally optimal composable plan	P	NP-hard	NP-hard
Complexity of globally optimal non-composable plan	P	P	-

Table 2: Summary of Query Processing Complexities for Network Usage Optimization

However, it must be noted that the key assumption for computing the globally optimal query plan is that the system is frozen on any query arrival or revocation for query plan computation. While this may be feasible for a system where query arrivals and revocations occur infrequently, it would not be practically feasible in a system where query arrivals and revocations occur frequently.

References

[1] Y. Ahmad, U. Cetintemel, J. Jannotti, A. Zgolinski, and S. Zdonik. Network awareness in internet-scale stream processing. In *IEEE Data Engineering Bulletin*, 2005.

[2] D. Anderson. Resilient overlay networks. In *MS Thesis, Department of Electrical Engineering and Computer Science, MIT*, 2001.

[3] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. In *In Communications of the ACM*, 1970.

[4] M. Carey and H. Lu. Load balancing in a locally distributed database system. In *In Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 108–119, 1986.

[5] M. J. Carey, D. J. DeWitt, M. J. Franklin, N. E. Hall, M. L. McAuliffe, J. F. Naughton, D. T. Schuh, M. H. Solomon, C. K. Tan, O. G. Tsatalos, S. J. White, and M. J. Zwilling. Shoring up persistent applications. In *ACM SIGMOD International Conference on Management of Data*, pages 383–394, 1994.

[6] G. Cormode and M. Garofalakis. Streaming in a connected world:quering and tracking distributed data streams. In *International Conf. in Data Engineering*, 2007.

[7] N. Dalvi, S. Sanghai, P. Roy, and S. Sudarshan. Pipelining in multi-query optimization. In *Journal of Computers and System Science (JCSS)*, pages 728–762, 2003.

[8] F. Deng and D. Rafiei. Approximately detecting duplicates from streaming data using bloom filters. In *In SIGMOD*, 2006.

[9] S. Ganguly, A. Goel, and A. Silberschatz. Efficient and accurate cost models for parallel query optimizaton. In *In Proceedings of the ACM SIGMOD/SIGACT Conference on Principles of Database Systems*, pages 172–181, 1996.

[10] S. Ganguly, W. Hasan, and R. Krishnamurthy. Query optimization for parallel execution. In *The ACM SIGMOD Conference on Management of Data*, pages 9–18, 1992.

[11] H. Garcia-Molina, J. D. Ullman, and J. Widom. Database system implementation. In *Prentice Hall*, 2000.

[12] K. Gorman, D. Agarwal, and A. E. Abbadi. Multiple query optimization by cache-aware middleware using query teamwork. In *The 18th Intl Conf. on Data Engineering*, 2002.

[13] G. Graefe and W. J. McKenna. The volcano optimizer generator: Extensibility and efficient search. In *Proceedings of the Ninth International Conference on Data Engineering*, pages 209–218, April 1993.

[14] A. Gupta, S. Sudarshan, and S. Viswanathan. Query scheduling in multi query optimization. In *The Intl. Symposium on Database Engineering and Applications*, pages 11–19, 2001.

[15] L. M. Haas, J. C. Freytag, G. M. Lohman, and H. Pirahesh. Extensible query processing in starburst. In *Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, pages 377–388, 1989.

[16] Y. Ioannidis and Y. Kang. Left-deep vs. bushy trees: an analysis of strategy spaces and its implications for query optimization. In *The ACM SIGMOD Conference on Management of Data*, pages 168–177, 1991.

[17] D. Kossmann and K. Stocker. Iterative dynamic programming: A new class of query optimization algorithms. In *ACM Transactions on Database Systems*, March 2000.

[18] L. Mackert and G. Lohman. R* optimizer validation and performance evaluation for distributed queries. In *In Proceedings of the Conference on Very Large Data Bases*, pages 149–159, 1986.

[19] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD*, 2003.

[20] A. Metwally, D. Agrawal, and A. E. Abbadi. Duplicate detection in click streams. In *In Proc. of WWW*, 2005.

[21] C. S. Park, M. H. Kim, and Y. J. Lee. Finding an efficient rewriting of olap queries using materialized views in datawarehouses. In *Decision Support Systems, Vol 32, Issue 4*, 2002.

[22] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-aware operator placement for stream-processing systems. In *International Conference on Data Engineering*, 2006.

[23] R. Prim. Shortest connection networks and some generalizations. In *Bell System Technical Journal, 36*, pages 1389–1401, 1957.

[24] P. Roy, A. Seshadri, A. Sudarshan, and S. Bhobhe. Efficient and extensible algorithms for multi query optimization. In *ACM SIGMOD Conf. on Management of Data*, pages 249–260, 2000.

[25] D. Schneider and D. Dewitt. Tradeoffs in processing complex join queries via hashing in multiprocessor database machines. In *The Conference on Very Large Data Bases (VLDB)*, pages 469–480, 1990.

[26] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *The 1979 ACM SIGMOD International Conference on Management of Data*, 1979.

[27] T. Sellis. Multiple-query optimization. In *ACM Trans. on Database Systems*, pages 23–52, 1988.

[28] T. Sellis and S. Ghosh. On the multiple-query optimization problem. In *IEEE Transactions on Knowledge and Data Engineering*, pages 262–266, 1990.

[29] K. Shim, T. Sellis, and D. Nau. Improvements on a heuristic algorithm for multiple-query optimization. In *Data and Knowledge Engineering*, pages 197–222, 1994.

[30] N. Trigoni, Y. Yao, A. Demers, J. Gehrke, and R. Rajaraman. Multi-query optimization for sensor networks. In *IEEE International Conference on Distributed Computing in Sensor Systems*, 2005.

[31] C. Wang and M. S. Chen. On the complexity of distributed query optimization. In *IEEE Transactions on Knowledge and Data Engineering*, pages 650–662, 1996.

[32] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. In *ACM SIGMOD Record 31(3)*, pages 9–18, 2002.