Minimizing Communication Cost in Distributed Multi-query Processing *

Luis Galárraga, Fateme Shirazi
Saarland University

July 22, 2010

Abstract

The increasing emergence of large-scale distributed applications has motivated a great amount of work related to optimization schemes for such systems in the areas of load balancing, optimal scheduling and communication cost. This report is focused in the last feature and presents a series of methods to address the problem of minimizing the communication cost in a multi-query environment. This work has direct applicability in systems like sensor networks and monitoring systems. For distributed databases it relies on the integration with other optimization schemes like load balancing and join order optimization. Since the problem is proved to be NP-hard, it has been addressed through a series of approximation algorithms and instance restrictions which most of the time exhibit more than satisfactory solutions.

*Inspired on the work by Jian Li, Amol Deshpande and Samir Khuller from the Department of Computer Science, University of Maryland

1 Introduction

The increasing emergence of large-scale distributed applications like wireless sensor networks, publish-subscribe systems and distributed stream processing applications has motivated a great amount of work related to optimization schemes for such systems in the areas of load balancing, optimal scheduling and communication cost. This report is focused in the last feature and presents a series of methods to address the problem of minimizing the communication cost in a multi-query environment. Depending on the application, the interpretation of cost may vary from the amount of energy consumed in a sensor network to the network bandwith utilized in a publish-subscribe system. In all cases, our goal is to optimize the data movement across the nodes which take part of a retrieval operation. The applicability of the proposed solution is independent from the strategy for serving the data (pull or push) and the nature of the retrieval operation (joins or aggregations) which makes it also suitable for standard distributed databases as long as it is combined with other optimization schemes like load balancing and join order optimization. Since the problem is proved to be NP-hard, it has been addressed through a series of approximation algorithms and instance restrictions which most of the time exhibit more than satisfactory solutions.

2 Problem formulation

Our goal is to minimize data movement in a distributed system in presence of multiple queries that may have data sources in common.

Input: set of relations or data sources $S = \{S_1, S_2, ...S_n\}$ with their sizes $z : S \to \Re$, the topology graph as an undirected graph $G_c = (V, E), w : E \to \Re$, an assignment of data sources to nodes $f : S \to V$, a set of queries, each involving a subset of the data sources $Q = \{Q_1, Q_2, ...Q_m\}, Q_i^R \subseteq S$. Each query plan is encoded as a directed tree where the leaves are the data sources, the internal nodes are the intermediate data joins and the root is the destination node as depicted in the picture 1.

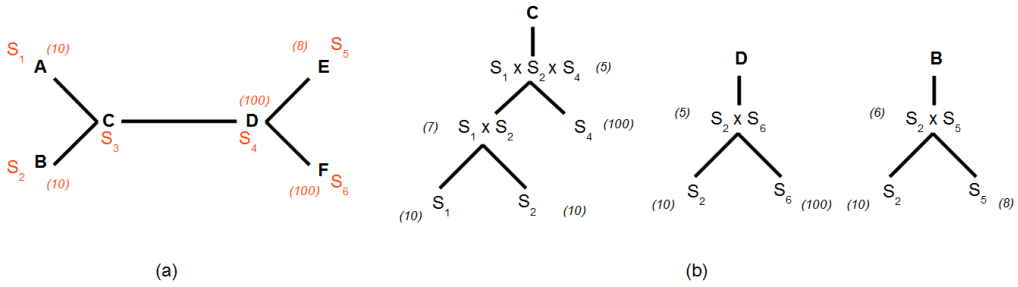

Figure 1: The input of our problem: (a) the topology graph and (b) the query plans

Output: data movement plan that minimizes the total communication cost incurred while executing the queries. If a block of data sized S is sent across an edge e, then the communication cost is given by $S \times w(e)$.

3 Proposed methods and analysis

The problem can be reduced to the Steiner Tree problem, however there exists an algorithm which solves it optimally when the topology graph G_c is a tree. For general topologies, there are approximation schemes, some of them relying on certain structure for the input queries.

3.1 Graph theory concepts

Before explaining the proposed methods, it is necessary to provide a summary of the graph theory concepts that are used in the developed algorithms.

3.1.1 Maximum flow and minimum cut in hypergraphs

An hypergraph is the generalization of a graph where each edge can be compounded by the association of any number of vertices, instead on only two like in standard graphs. We call hyperedges, to those edges associating any number of vertices, but two. Most of the concepts developed for standard graphs are applicable to hypergraphs through convenient transformations of the hyperedges.

Max-flow and min-cut are one of the most emblematic problems in graph theory. Given a directed weighted graph and two nodes known as source and sink, the maximum flow problem asks for assigning a real number to every edge in the graph, known as the flow, under the conditions that it cannot exceed the capacity of the edge and that for every vertex in the graph (except the source and the sink), the incoming flow has to be equal to the outgoing flow. We want to find the biggest possible flow across the network defined by the graph. On the other hand, the minimum cut asks for a set of edges of minimum weight such that if removed, there is no path from the source to the sink. The graph is divided into two connected components and the edges of the minimum cut have each endpoint in a different component. The max-flow/min-cut theorem claims that the value of the maximum flow in a network is equal to the weight of the minimum cut.

The max-flow and min-cut problems can be solved in polynomial time. For the first one, Ford-Fulkerson and Dinitz algorithms offer solutions in $O(|V|f)$ and $O(|E||V|^2)$. For min-cut, Edmonds-Karp algorithm solves the problem in $O(|V||E|^2)$ where $|E|$ and $|V|$ denote the size of edges and vertices sets respectively.

As stated before, standard graph theory can be easily extended to hypergraphs. In this case we need to turn our hypergraph into an standard graph like depicted in picture 2. Given an hyperedge of weight w, add two new vertices connected by a standard edge of weight w. Then, add edges of infinite weight from every vertex in the hyperedge to the first node and other set of edges from the second one to the vertices of the hyperedge. For min-cut formulations, such construction allows us to treat hyperedges as standard edges because any min-cut solver will take either all the added edges or none of them, which can be interpreted as either the hyperedge takes part of the minimum cut or not.

3.1.2 The weighted hypergraph partition problem

Instead of a single source and sink, we are given an hypergraph with a set of sources L_s and a set of sink L_t, such that they are disjoint. We want to find a minimum cut such that those sets lie on two different partitions if the edges of the cut are removed. This problem can be easily transformed into a standard min-cut formulation by adding artificial source and sink and connecting them to the set L_s and L_t with infinite weight edges.

3.1.3 Steiner Tree problem

Given an undirected graph $G = (V, E)$ with non-negative edge weights c_e and a set of terminals, $T \subset V$, the Steiner tree problem asks for the minimum weight tree, subgraph of G that connects all the terminals. This problem

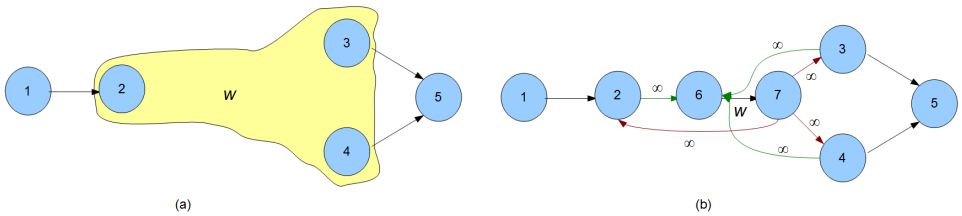

Figure 2: (a) An hypergraph containing an hyperedge associating 3 vertices. (b) The conversion of the hypergraph into a standard graph.

is known to be NP-Hard and several approximation algorithms are known for it [2]. A generalization of the Steiner tree problem is the Connection-Facility location problem [3]. Here, along with a graph G as defined above, we are given a set $D \subset V$ of demands and a parameter $M > 1$. Each demand j in D has a non-negative weight d_j . A solution consists of a set of facilities $F \subset V$ to be opened, a tree T of G spanning F, and an assignment, $f()$, of demands to the open facilities. If this solution assigns demand j to the open facility $f(j) \in F$, then the cost is given by $\sum_{j \in D} d(j) * l(j, f(j)) + M \sum_{e \in T} c_e$ where the function l denotes the shortest path distance using edge lengths c_e.

3.2 Algorithm for tree topology

The algorithm for minimizing the communication cost in tree topologies consists of tree steps:

1. Build a directed weighted hypergraph H_D, by combining the query plan trees for all the queries in the input. Edges are oriented from children to parent. H_D explicitly captures all the opportunities for sharing the movement of data sources among the queries. If a data source appears in more than one query plan, then it forms an hyperedge with all its parent nodes in the different query plans.

2. For every edge in the topology graph G_c decide which data sources and intermediate results move across that edge by solving an instance of the weighted hypergraph partition problem.

   Consider the topology graph G_c. Since it is a tree, every single edge defines two isolated connected components. For edge (C, D) in our example, call them G_c^C and G_c^D (see picture 3). Now label every node in H_D as C or D depending on whether it lies on G_c^C or G_c^D. Those labeled nodes define our input sets L_C and L_D for the weighted hypergraph partition problem. A solution is a minimum cut, a set edges whose endpoints are one in L_C and one in L_D and induces a labeling for the internal nodes in the hypergraph. We interpret those edges as data movement from node C and D (or viceversa).

3. Combine the local solutions for all the edges of the topology into a single global data movement plan. Local solutions may not agree on where the internal nodes should be evaluated.

   The solutions for every edge of G_c are local and even though they describe the optimal data movement in that edge, they might disagree in the place where the intermediate data sources are evaluated. For every internal node i in H_D, construct a graph J^i with vertex set equal to the topology graph G_c and edges according to this rule: $\forall u, v \in V(J^i), (u, v) \in E(J^i)$ if i is assigned label v in H_D. J^i encodes information about how the different local plans locate evaluation of node i and defines a path towards the node where the evaluation must be done.

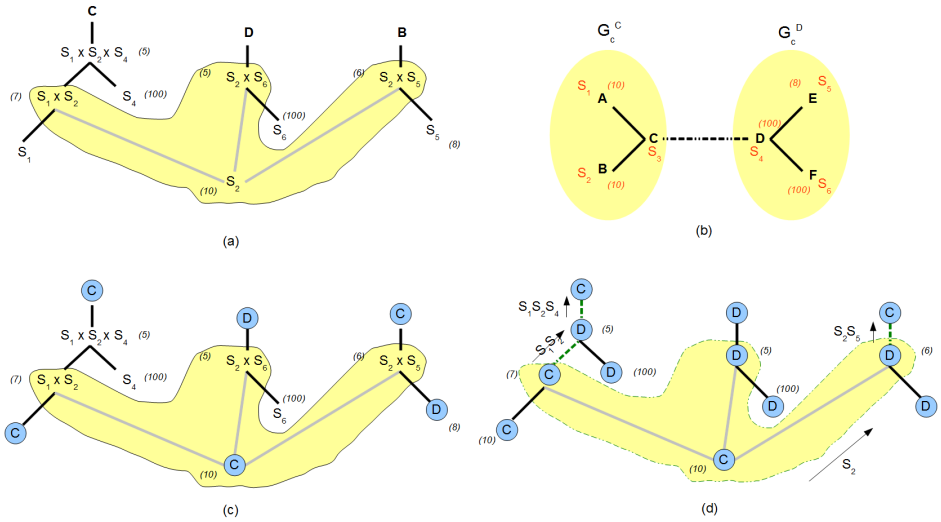

Figure 3: (a) The hypergraph H_D constructed by merging the query plans. (b) Connected components defined by edge (C, D). (c) Label C nodes lying in G_c^C and D the ones in G_c^D. (d) A solution for the weighted hypergraph partition problem induces a labeling for internal nodes.

3.3 Arbitrary graph topologies

If the topology is an arbitrary graph, a dynamic programming approach can solve the problem optimally for a single query. For multiple queries, there is a $O(lg(n))$-approximation achieved by applying a tree-metric transformation to the original topology graph G_c and solving the problem optimally for this tree topology.

3.3.1 The pairs problem

Instead of imposing restrictions on the topology graph G_c, we can try to solve the problem for arbitrary graph topologies when the queries have certain structure. Consider a special type of queries, restricted to be over two nodes, with results of size 0 (in other words, the query results do not need to be shipped to any sinks). We define the query overlap graph H for a set of queries, as the graph where the vertices correspond to the set of data items and each edge corresponds to a pair query.

We are interested in star shaped query overlap graphs. For this case, the problem of finding an optimal data movement for a set of pair queries can be reduced to the Steiner Tree problem if the data sources have the same size, or to the Connected Facility Location Problem in case of heterogenous data sizes. If the first case, it is easy to see that the cost of the data movement is dominated by the cost of the topology, therefore it suffices to find the cheapest tree that connects all sources because the evaluation place becomes irrelevant. On the other hand if data sources have different sizes, they can be interpreted as demands which have to deliver the data to the right facility. Furthermore, such facilities have to be connected in the cheapest way.

Finally, recall the nature of the defined queries looks pretty ideal. In practice queries normally involve more than two data sources and the size of the intermediate results can be far from negligible. The pairs problem formulation was

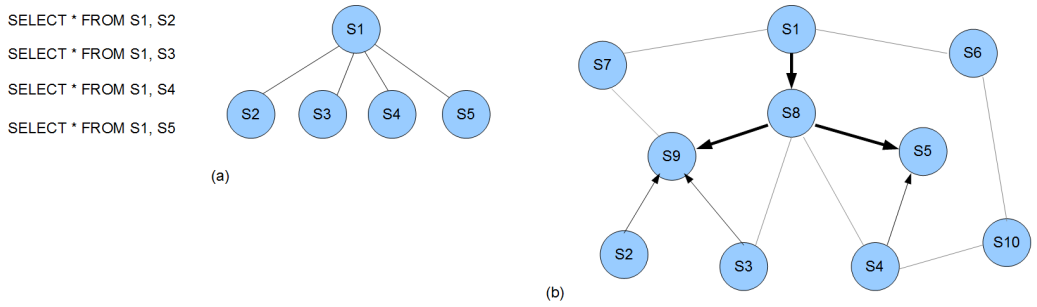

Figure 4: (a) A set of pair-queries with their overlap graph. (b) An optimal movement plan for the queries in the topology graph, assuming data sizes are equal. It is equivalent to an instance of the Connected Facility-Location problem

initially develop to relate this work with the problem of finding an optimal movement plan for agregations over a set of sources where normally, intermediate results (partial aggregations) have small or constant size. Let a query Q be an aggregate query over data sources $\{S_1, ..., S_k\}$. Let the destination be node S_d. We introduce a new data source, SR_d, with size equal to the size of a partial aggregate, and attach this data source to the node Sd with a zero cost edge. Then create k pair queries: $\{(S_1, SR_d), ..., (S_k, SR_d)\}$ and solve the problem using the algorithms described in previous paragraph.

4 Conclusion

Even though the idea of sharing data movement was initially motivated by systems like sensor networks, where the notion of cost is normally associated with energy comsuption, it could be easily adapted to general notions of cost like bandwith usage or delay time. Communication cost can be the main bottleneck in certain applications like distributed databases and publish-subscribe systems where a non-optimal data transmission plan might lead to a saturation of the underlying communication infraestructure and therefore to a degradation of the system performance. Thus, optimizing data movement is vital in system design but hard at the same time. In spite of being an NP-Hard problem in its general formulation, help from graph theory and approximation algorithms has allowed to develop a set of algorithms that solve the problem optimally or offer sub-optimal solutions with good quality guarantees. Furthermore, the methods explained above accept several enhancements like sharing of intermediate results and load balancing. Also notice it assumes the queries are given at once, which implicitly suggests a scheduling process in order to collect them as they arrive in the system. To make things harder, it does not also consider replication of data sources, a facto standard in modern distributed systems. Designing extensions for the algorithm such that it does not have to be run for all the possible locations of a replica seems like a challenging problem.

Finally, I claim this approach can be also ported to any other system with an arbitrary notion of cost like peer to peer systems, where sharing of data movement would be highly convenient for popular pieces of data. It is worth mentioning that in any case, data movement optimization is just one of the dimensions of an efficient distributed system design, since it must combined with other schemes like paralell query optimization (in the particular case of distributed databases) and load balancing.

References

[1] J. Li, A. Deshpande, and S. Khuller, "Minimizing communication cost in distributed multi-query processing," in *ICDE*, pp. 772–783, 2009.

[2] G. Robins and A. Zelikovsky, "Improved steiner tree approximation in graphs," *SODA*, 2000.

[3] F. Eisenbrand, F. Grandoni, T. Rothvoß, and G. Schäfer, "Approximating connected facility location problems via random facility sampling and core detouring," in *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, (Philadelphia, PA, USA), pp. 1174–1183, Society for Industrial and Applied Mathematics, 2008.