Query Optimization- An Experimental Approach for Distributed Environment

Sagar Yeruva
Associate Professor, CSE, St.Peter’s Engineering College, Hyderabad, India-500 014. E-mail: sagaryeruva@yahoo.com

Dr.P.Padmanabham
Principal, Bharat Institute of Engineering & Technology, Ranga Reddy (Dt), (A.P), India-501 510. E-Mail: padmanabham46@yahoo.com

Dr.P.V.Kumar
Professor, Head, CSE, Bharat Institute of Engineering & Technology, Ranga Reddy (Dt), (A.P), India-501 510 E-Mail: pvkumar58@gmail.com

Abstract-This paper strongly emphasizes the approach for query optimization which is a framework model for distributed computing environment systems. We have two popular methods for query optimization. One which is traditional which follows the stages like query planning, deployment and adaptation. The second one which is our main experimental approach which follows the stages like query planning and deployment together as a single stage followed by adaptation. The approach of integration of planning and deployment while writing for distributed queries which involve many sub-queries in distributed data stream systems and applications. This method makes use of hierarchical network partitions which provides operator level-reuse which utilizing network characteristics to maintain an appropriate search space during query planning and deployment. The approach has been practically experimented and proved its efficiency over the traditional methods.

Key Words-Distributed Data Bases, Distributed Systems, Optimization, Communication networks, query processing and deployment

1 INTRODUCTION

Now-a-days we have many systems which produce stream data at multiple geographical distributed sites or locations. In this case it is common that there should be a centralized query processor, because of the high communication cost and yet continuously changing processing load at the central server. To overcome this problem and ensure efficiency and scalability, naturally distributed applications adopt a distributed processing paradigm. This can be identified with many features like Computing environment, Data from multiple sources, Continuous and multiple query execution and Delivery over network with different input and output rates.

Instead of shipping all data streams to a single node and processing all the stream queries in a centralized server, many have shown that performing distributed processing of stream queries using techniques such as In-Network Processing [3], [4], [5] and filtering at the source [6] minimizes the communication overhead, significantly improving performance. In reference [1] we have come to a popular approach which opens a new way of optimizing distributed queries in several applications like in [3],[4],[5] and [6]. Here we are briefly explaining that approach for the clarity purpose:

Given that data streams are typically produced from multiple disparate nodes, stream queries naturally consist of many operators (filters, joins, etc.) on multiple data streams of interest. We can think of a data stream query as a continual query
being “deployed” in the network, with data streams flowing between operators associated with distributed physical streaming nodes, which may either be sensor nodes or the relay nodes in a data stream delivery network. The conventional approach to stream query processing used in many existing distributed data stream management systems \(^7\)\(^8\) consists of three consecutive phases:

1. Query planning.
2. Query deployment and
3. Query adaptation.

Concretely, the system constructs a query plan (e.g., the stream query processing should follow a specified join ordering) at compile time and deploys this plan at runtime to improve performance. Fig. 1a gives a sketch of this approach. A fundamental problem with this static optimization approach is its inability to respond to the unexpected data and resource changes occurring at runtime. For example, the join order chosen at compile time may require intermediate results to be transported to another network node over a long distance, even though there exists an alternate join order that is more efficient. Similarly, a predefined join order may involve a transfer or a processing of an intermediate result to a node that is currently unavailable, thus causing the query to halt even though an alternate join order exists and is available. Furthermore, given that each query plan is computed at compile time independently and once for all, the predefined join order from one query plan may prevent us from reusing the results of an already deployed join from another query at runtime.

Figure-1. Approaches. (a) Plan, then deploy. (b) New approach.

limits the scope of the adaptation which aims at exploiting runtime environment properties to further optimize the efficiency of distributed stream query deliveries. Bearing these issues in mind, we have a distributed stream query optimization framework that considers the query plan and the deployment simultaneously (Fig. 1b). This consists of the system architecture for integrating distributed stream query planning and query plan deployment and a suite of techniques for performing query planning in conjunction with deployment planning. One of the key ideas in this framework is to use hierarchical network partitions to scalable exploit various opportunities for operator-level reuse in the processing of multiple stream queries. Fig. 2 compares the approach of integrating planning and deployment through operator reuse with two existing “Plan, then deploy” approaches—the Relaxation algorithm \(^9\) and an optimal deployment through exhaustive search. The graph shows the total communication cost (the total data transferred along each link times the link cost) incurred by 100 queries over five stream sources each, on a network. The figure shows that significant (> 50 percent)
cost savings can be achieved by combining the planning and deployment phases. This is practically shown in results and so we emphasizing. It is well known that, as the size of the network grows the number of possible plan and deployment combinations can grow exponentially. The cost of considering all possibilities exhaustively is prohibitive. Consider Fig. 2. With a network of 64 nodes, combining query plans and plan deployments simultaneously required us to examine nearly thousands of plans for a single query over five streams. Clearly, a key technical challenge for effectively combining query planning and plan deployment is to reduce the search space in the presence of large networks and a large number of query operators. One idea we explore in this paper is to address this challenge by using hierarchical network partitions as a heuristic, aiming at trading some optimality for a much smaller search space. Concretely, we organize the network of physical nodes into a virtual hierarchy

Figure- 2. Comparison with typical approaches.

and utilize this hierarchy along with “stream advertisements” to guide query planning and deployment. We can make use of various algorithms to facilitate operator reuse to guide the problem of partitions of the network.

2 OPTIMIZATION PROBLEM

Our problem definition addresses the continual query equivalent of “select-project-join” queries that involve simple selection, projection, and join operations on one or more data streams. The focus of this paper is on join ordering and the initial placement of operators. Note that it may be possible to modify existing deployments to get a better solution. However, such modifications require us to consider the cost of reconfigurations and deal with translation of state as well. We assume stream joins are performed using standard techniques. We assume that potentially, any operator can be deployed at any node in the system. Given a query, there could possibly be multiple execution plans that the system could follow to produce results. We assume that all such plans produce equivalent results.

2.1 Query-optimization problem

Given a query Q to be deployed over a network N, and a (possibly empty) set of existing query deployments \( D = \{ D_1, D_2, \ldots, D_n \} \), find a query tree \( \{ P_i^Q \} \) and a deployment \( \{ P_i^Q, N \} \) for Q such that Cost \( (M \{ P_i^Q, N \}) \) is minimum over all possible query trees and deployments.

3 DEPLOYMENT TIME

The results shown by the prototype experiments on Emulab using IFLOW \[^{[10]}\], our implementation of the distributed data stream system which supports hierarchies and advertisements as described earlier. The testbed on Emulab consisted of nodes (Intel XEON, 2.8 GHz, 512-Mbyte RAM, RedHat Linux 9), organized into a topology that was again generated with the internodes delays were set between 1 and 6 ms. The workload for the following experiments consisted of 25 queries over eight stream sources and sinks distributed across the system. The number of joins per query varied from 1 to 3. The experiment conducted on Emulab was aimed at measuring the time to deployment of a query over the system when using different algorithms. Our experiment allows us to conclude that our algorithms can greatly reduce the search space for the query deployment problem while offering efficient deployments with acceptable sub optimality.

4 EXPERIMENTATION STAGES

In the following section we present our experiment in various phases which includes like the following
4.1 Communication networking

Computing or networking is a distributed application architecture that partitions tasks or workloads between service providers (servers) and service requesters, called clients. Often clients and servers operate over a computer network on separate hardware. A server machine is a high-performance host that is running one or more server programs which share its resources with clients. A client also shares any of its resources; Clients therefore initiate communication sessions with servers which await (listen to) incoming requests.

4.2 Distributed Data Stream:

Distributed data streams systems are distinguished by a number of characteristics. First, a network of computing nodes with heterogeneous bandwidth and computing resources together serves as a distributed data stream delivery system. Second, data streams originate from multiple sources and are disseminated to multiple receivers. Third, multiple continuous stream queries are executing simultaneously on the stream delivery network with different input and output rates. Instead of shipping all data streams to a single node and processing all the stream queries in a centralized server, many have shown that performing distributed processing of stream queries using techniques such as in-network processing and filtering at the source minimizes the communication overhead on the system and helps spread processing load, significantly improving performance.

4.3 Planning

A predefined join order may involve a transfer or a processing of an intermediate result to a node that is currently unavailable, thus causing the query to halt even though an alternate join order exists and is available.

4.4 Adaptation

Given that each query plan is computed at compile time independently and once for all, the predefined join order from one query plan may prevent us from reusing the results of an already deployed join from another query at runtime. This limits the scope of the adaptation which aims at exploiting runtime environment properties to further optimize the efficiency of distributed stream query deliveries.

4.5 Query processing

The paradigm of in-network query processing has been used earlier in sensor networks, and also in scientific data flows and large-scale visualizations with data manipulations sometimes pushed to the source for efficiency. The use of this technique in stream-based systems to only decide operator placement when the query tree is already known, The network aware algorithms in first perform phased deployments which we have shown to be suboptimal. Second, they do not address the important question of how the query should be divided and assigned to different portions of the network. Clearly, as seen from our experiments on varying cluster sizes, this decision can impact the efficiency of the resulting deployments. Also, no analysis is provided on the impact of the number of zones and the placement heuristics on the computational complexity of the algorithms.

4.6 View of Experimental results

Figure-4.6.1 specifies the communication between the nodes in the network.
environment that integrates query planning and deployment through hierarchical network partitions. The framework consists of two key components: a hierarchical clustering of network nodes that allows network approximations and stream advertisements that enable operator reuse. We have shown through both experimental and analytical results that our algorithms are efficient and scalable at costs comparable to optimal while exploring much fewer plans. The time complexity for the execution of this kind of query is also very much minimized when it is compared to the conventional model or methods.

REFERENCES


