Distributed Streaming Query Planner in Calder System

Ying Liu, Beth Plale, Nithya Vijayakumar
Computer Science Department, Indiana University
{yingliu, plale, nvijayak}@cs.indiana.edu

1. Introduction

Recently, as sensors and instruments are applied in more and more applications, time-dependent data streams are becoming prevalent data resources. These highly distributed and time-dependent resources bring new challenges to build high-performance large-scale processing systems. To address these challenges, we develop Calder [2], a system which provides scientific researchers grid access to widely distributed, real-time data streams. In Calder, user requests are continuously running SQL queries. Moreover, these requests could not only extract data products from asynchronous multiple streams but also project subsets from a single real-time stream, and possibly over time constraints. Therefore, a *high-performance distributed query planner* becomes an critical part of the whole system.

While continuously running query processing [3] has been studied for a while and many query planning systems have been built, few research projects are focused on generating efficient query plans on large-scale distributed data streams with limited computational resources. We observe that in these systems, many long-running queries are at least partially overlapped with each other. Therefore, our distributed query planner achieves better performance and less computational resources by reusing existing queries.

The contribution of this work has two folds. First, we extend the current query planners' cost metric space by introducing *Network Bandwidth Cost*, *Query Deployment Cost* and *Query Re-using Cost*; second, we develop a suite of algorithms for re-using existing query fragments under different scenarios. One of the most important reusable queries is called *Structure-Sharable Query*.

2. Extended Cost Metrics Space

Network bandwidth cost: In distributed systems, processing data maximally in a local fashion significantly achieves high system performance. In Calder [2], we disintegrate centralized query requests and distribute their fragments to the computation nodes that are close to the corresponding data streams. However, the intermediate streams generated by the distributed query plans take noticeable network bandwidth. Therefore we count the network bandwidth cost into the evaluation as an extension to the planning cost metrics. An ideal plan is not supposed to have too much bandwidth cost.

*Query reusing and deployment cost:*We posit the *continuous query model* in Calder system, where each query will be kept in the system and repetitively executed for a while. By reusing these existing queries, we can save query processing time and physical computational resources from deploying and executing new query plans. The overhead of the reusing strategy lies in searching and tuning the resuable queries. Naturally we extend planning cost metric space with the following two new metrics: *Query Deployment Cost* and *Query Re-using Cost*. Only when the *query deployment cost* is larger than the *query re-using cost*, reusing techniques will be considered.

3. Query Reusing Algorithms and Initial Experimental Results

Query reusing process includes two core steps: finding the candidate queries which can be reused by the incoming queries and tuning such queries to make them practically reusable. Therefore, query reusing algorithms include *Query Searching Algorithm* and *Query Tuning Algorithm*. These algorithms varies in different scenarios. Our query planner is adaptable to 4 reusing cases and we briefly discuss their searching and tuning algorithms as follows.

Identical Query Two queries are exactly the same. *Searching Algorithm* directly works on the text-level by hashing input query text strings into fixed-length hash values and matching those hash values. Here we adopt the MD5 hashing function. In this case, no tuning algorithm is needed as the identical queries can simply share the results.

Parameterized Query It characterizes the group of queries, the only difference among which is the constants in the *WHERE* conditions. *Query Searching Algorithm* is *Tree Matching Algorithm* in which all the queries are compiled into query trees and pre-order traversal is used to compare two query trees. During this matching process, con-

stants in all the selected nodes are replaced with a uniform place holder, then the queries with different constants are still recognized as the matched ones. As for *Query Tuning Algorithm*, we adopt the *Group Optimization Algorithm* introduced in [4] by using a parameter table to match the queries' constants.

Contained Query Given a query pair (A, B), we define query A is contained by query B, if A's result set is the subset of B's. Based on the set theory, we impose a set of rules. (*e.g.*,$a \geq b \Rightarrow a > b$) Then, we use the *Rule-based Searching Algorithm* to find the contained query pairs. *Query Tuning Algorithm* for such cases is straightforward: if the incoming query A is contained by the existing query B, we use the query B's outputs as the inputs of query A.

Structure Sharable Query This category of reusable queries is found from sensor-based applications where there are hundreds of same-typed sensors generating data in the same format. Though the streams are in the same data format, they are recognized as different resources and separate queries are issued against them. We found those queries (*e.g.*, Query A and Query B in Figure 1) on the same formatted streams can share their *Query Structures*. A Query Structure is a query tree that is stripped of its leaf nodes (*i.e.*, it is only with the internal operation nodes).

Query searching algorithm is an application of the Tree Matching Algorithm on Query Structures. In order to share the Query Structures among multi-queries, we inject input streams from Query A and B into the same structure. Meanwhile for avoiding the events from different queries being mixed together, we attach an identifier to each input stream when it passes through the *Mark* node. At the top of the query tree, a special node, the *DeMark* Node, is added to route the results for different queries. Figure 1 shows how such reusing cases work in our system.

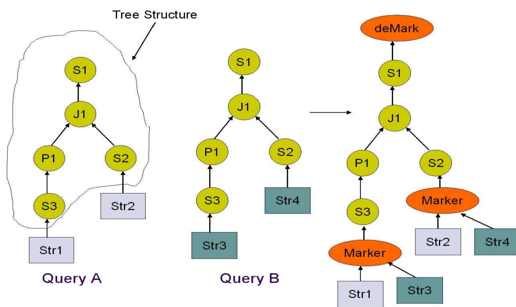

Figure 1. Structure Sharable Query

Experiment: Our query planner combines the query reusing model and the direct deployment model. At the beginning, there are not many queries in the system, so the matching query hit ratio is low. During this stage, we deploy a new query directly into the network without searching for similar queries. After the query matching ratio reaches the threshold value R, we activate the query re-using model.

The following experiment shows how we decide threshold value. We have 1000 queries in the existing query pool and *Tree Matching Algorithm* is used for query searching. But currently we only consider full query reusing. Figure 2 shows *Query Searching Time* t_s (i.e time used to locate the reusable query in existing query pool), *Query execution time* t_e and *Query Deployment Overhead* t_o which include running scripts compilation time, query runtime deployment time and Ringbuffer setting up time. From these data, we can see that deployment overhead t_o is observable; therefore saving the cost by reusing existing queries is a promising approach. With these experimental data, we can compute *Query response time* for both models. For *direct deployment model*, $t_{res1} = t_o + t_e = 50 + 5.8 = 55.8ms$; for *reusing model* $t_{res2} = (t_s + t_e) * R + (t_s + t_o + t_e) * (1 - R) = 10.8R + 60.8 * (1 - R)$. From this, we get the threshold R is 10%.

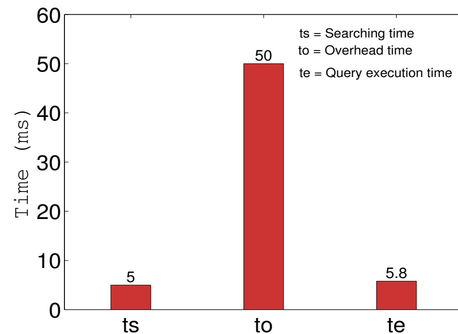

Figure 2. Processing Time Distribution

4. Conclusion and Future Work

Our planner combines the query reusing model and the direct deployment model for generating high performance distributed streaming query plans. We extend the query planning metrics and propose novel re-using algorithm against the Structure Sharable queries. Currently we are evaluating these query reusing algorithms against the query load from real applications such as LEAD [1]. Some other efficient reusing alternatives are also under our consideration.

References

[1] LEAD: Linked environments for atmospheric discovery. http://lead.ou.edu, 2004.

[2] Calder - grid enabled data stream processing. In *URL: http://www.cs.indiana.edu/dde/projects/Calder.html*, 2005.

[3] D. J. Abadi and et al. The Design of the Borealis Stream Processing Engine. In *CIDR*, 2005.

[4] J. Chen and et al. NiagaraCQ: a scalable continuous query system for Internet databases. In *ACM SIGMOD ICMD*, 2000.