Maximizing the Sustained Throughput of Distributed Continuous Queries

Ioana Stanoi, George Mihaila, Themis Palpanas, Christian Lang
IBM Research
irs,mihaila,themis,langc@us.ibm.com

ABSTRACT

Monitoring systems today often involve continuous queries over streaming data, in a distributed collaborative system. The distribution of query operators over a network of processors, and their processing sequence, form a query configuration with inherent constraints on the throughput it can support. In this paper we propose to optimize stream queries with respect to a version of throughput measure, the *profiled input throughput*. This measure is focused on matching the expected behavior of the input streams. To prune the search space we used hill-climbing techniques that proved to be efficient and effective.

Categories and Subject Descriptors: H.2 DATABASE MANAGEMENT Miscellaneous

General Terms: Algorithms, Performance

Keywords: Data Management, Stream, Continuous Queries, Query Optimization

1. INTRODUCTION

A continuous monitoring query can be deployed in various configurations, some better than others with respect to optimization criteria such as latency, work, or throughput. Most previous work focused on choosing the query configuration that minimizes total latency and/or work. Each operator of a continuous query requires a certain amount of execution time for every incoming data tuple, which leads to an upper bound on the rate at which tuples can be processed. If the input streams exhibit higher rates than the query operators can process, then special mechanisms need to be in place to handle them. When high input rates represent only short bursts, buffers can be used to temporarily store the overflow of incoming data. If, instead, the high rates have to be supported for a long period of time, then data need to be purged out of the input to the operators. This approach cannot avoid the deterioration of the quality of query results. There has been a large body of recent research that focused on which events to shed in order to return a high-quality result. However, some loss of quality is unavoidable when information is discarded. For some applications any event may contain critical information and the reduction in the quality of results should be minimized.

We focus on a problem complementary to that of load shedding: finding a query configuration that, given resource and quality constraints, can successfully process the highest incoming stream rates. This translates into finding an order of query operators and a placement of the operators on physical nodes that maximize throughput. The measure of *throughput* quantifies the number of tuples that can be processed by the system in a unit of time. Since our goal is to maximize the input rate that can be processed without bottlenecks, we express throughput as a vector that quantifies the processing of each input stream. Moreover, input streams vary in behavior, knowledge that is incorporated into an *input profile* (or simply a *profile*). The goal of maximizing throughput is often times relevant *only* if it satisfies the requirements of the input profile. Out of the large space of possible query configurations, our goal is to find the query plans that maximize throughput and adhere to the input profile.

Examples of data stream management systems are Aurora/Borealis [4, 1], STREAM [9], TelegraphCQ [3], and Gigascope [5]. Designed to operated on single or multiple nodes, they mostly focus on minimizing the end-to-end data processing latency. Viglas and Naughton [8] proposed a rate-based optimization technique for streams, that maximizes the output rates. The problem of query execution for fixed plans in widely-distributed environments is also studied by Ahmad and Cetintemel [2] (minimize the bandwidth use, or meet certain quality of service requirements) and Pietzuch et al. [6](targeted to minimize end-to-end latency in distributed applications). Another recent study [7] describes algorithms for efficient, in-network execution of filter queries on streaming data.

2. MAXIMIZING A PROFILED THROUGHPUT

A query may receive input from multiple data streams with different rate fluctuations. One stream may come from a source that rarely emits events, while another stream may be characterized by long bursts of data at very high rates. If the query optimizer is given even coarse information on the expected input behavior, it can generate a query plan that is appropriate under these assumptions. Note that without this additional knowledge, the query optimizer will have no way of distinguishing between many feasible solutions, and may decide that the best solution is one that accepts a high input rate on the slower stream and a low input rate on the fast stream. We profile the input as an assignment of values to the input rates that becomes a target for supported throughput: $< r_1^p, r_2^p, \cdots r_n^p >$. *A solution $C.S$ is an assignment of values to the input stream rate variables of a given configuration C such that all the constraints are satisfied.* The quality $Q^p(C.S)$ of a solution $C.S$ should then quantify how much the solution achieves towards the goal of maximizing the throughput with respect to the profile. Note that the goal can also be surpassed. *The quality $Q^p(C.S)$ of a solution $C.S$ with respect to an input profile vector p is defined as:* $Q^p(C.S) = \min_{1 \le i \le n} \left(\frac{r_i^s}{r_i^p} \right)$.

Note that a configuration has an infinite number of solutions. Consider one solution $C.S =< r_1^s, r_2^s, \cdots r_n^s >$. Then all possible $C.S' =< r_1^{s'}, r_2^{s'}, \cdots r_n^{s'} >$ such that $r_i^{s'} \le r_i^s$ are also solutions for this configuration. The quality of a configuration is characterized by the best solution under that configuration.

Let a set of constraints $\{f_i\}$ have the following properties 1)constraint f_i is of the form $f_i(r_1, \cdots, r_n) \le c_i$, 2) $f_i()$ is a monotonically increasing function, and 3) c_i is a constant that measures the capacity of a resource or a quality of service requirement. The constraints $\{f_i\}$ describe the restrictions imposed in the distributed system by the physical resources (such as cpu and memory constraints of the physical nodes) and the service quality guarantees (such as application requirements on latency). Under these definitions, the throughput optimization problem becomes a nonlinear programming problem, and can be formally defined as follows: *Given a query plan Q (representing multiple continuous queries), a set of physical nodes $\{N_i\}$ organized in a network, and a set of constraints $\{f_i\}$, find an assignment of the operators in Q to the nodes $\{N_i\}$ (called a configuration), so as to maximize the objective function $Q^p(C)$, for all the configurations C that satisfy the constraints $\{f_i\}$.*

To find a solution, the query optimizer needs to traverse the search space of configurations, and compare each visited configuration with the configuration that was the best so far. Finding an optimal configuration is a hard problem, and hill-climbing techniques can reach very good results efficiently. Recall that each configuration can have an infinite number of solutions that satisfy the given constraints. To quickly identify the best solution for each configuration, we observe the following: *Let a query configuration C be restricted by constraints that are of the form $f(r_1, \cdots, r_n) \le c$, where c is a constant and $f()$ is monotonically increasing. For a profile $p =< r_1^p, r_2^p, \cdots r_n^p >$, a solution with greatest $Q^p(C.S)$ lies on the surface bounding the region of feasible solutions and on the line through origin and p.* The proof is omitted due to lack of space.

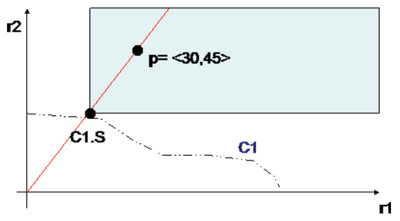

Figure 1: Example of best solution for a configuration C_1

To illustrate this point in two dimensions, consider the example in Figure 1. For configuration C_1 the intersection of the constraint boundary with the line through origin and $< 30, 45 >$ is at the solution $C_1.S$. Any solution with same or better quality according to the $Q^p(C.S)$ measure increases either r_1 or r_2, or both. This solution lies in the darker region where $C_1.S$ is the lower left corner. One can see that, due to the shape of the feasible space imposed by the properties of the constraints, no point in the feasible space can also be in the shaded region. Note that Proposition 1 allows us to compare two configurations by comparing the intersection points of the feasible space boundaries with the line from origin to the profile point p.

In our implementation of a distributed query optimizer we used hill-climbing techniques to prune the search space. We experimented with greedy algorithms, Tabu, Reactive Tabu, and Simulated Annealing. The various hill-climbing techniques performed similarly, obtaining the optimal or near-optimal solution in more than 90% of the time. Simulated Annealing is orders of magnitude faster than the other approaches, and therefore it may be preferable especially for large query plans.

3. CONCLUSION

In this paper we explore continuous query optimization, which maximizes the system's runtime capacity with respect to input rates that have an observed profile. An input profile represents the knowledge on input behavior, that is useful in targeting solutions appropriate for the specific runtime requirements of the system. The notion of a profile can be generalized further to include a set of target points, representing characteristics of input streams. Fitting the solution to the line segments described in this way by the profile corresponds to finding a solution that can support an evolving set of requirements. We plan to look into optimizing throughput according to various types of profiles, incremental query re-configuration, and improving parameters through learning.

4. REFERENCES

[1] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The Design of the Borealis Stream Processing Engine. In *CIDR*, Asilomar, CA, January 2005.

[2] Y. Ahmad and U. Çetintemel. Networked query processing for distributed stream-based applications. In *VLDB*, 2004.

[3] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah. Telegraphcq: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[4] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. Zdonik. Scalable Distributed Stream Processing. In *CIDR*, Asilomar, CA, 2003.

[5] C. D. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *SIGMOD*, 2003.

[6] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-Aware Operator Placement for Stream-Processing Systems. In *ICDE*, 2006.

[7] U. Srivastava, K. Munagala, and J. Widom. Operator placement for in-network stream query processing. In *PODS*, New York, NY, USA, 2005. ACM Press.

[8] S. Viglas and J. F. Naughton. Rate-based query optimization for streaming information sources. In *SIGMOD*, 2002.

[9] J. Widom and R. Motwani. Query processing, resource management, and approximation in a data stream management system, 2003.