ISSN No. 0976-5697

Volume 4, No. 2, Jan-Feb 2013

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Implementation of Stream Query Optimization Framework in Distributed Data Mining

S.Priya*
Research Scholar, Department of Computer Science
Gobi Arts& Science College, Gobichettipalayam India
Priyasubramani88@yahoo.com

B.Srinivasan
Associative Professor, Department of Computer Science
Gobi Arts& Science College, Gobichettipalayam India
srinivasan_gasc@yahoo.com

Mr.P.Narendran
Associative Professor, Department of Computer Science
Gobi Arts& Science College, Gobichettipalayam, India
narendranp@gmail.com

Abstract: Distributed stream query services must simultaneously process a large number of complex, continuous queries with stringent performance requirements while utilizing distributed processing resources. Query addresses the problem of optimizing multiple distributed stream queries that are executing simultaneously in distributed data stream systems. Research to develop top-down, bottom-up, and hybrid algorithms for exploiting operator-level reuse through hierarchical network partitions.

Keywords: Query Optimization, hierarchical network, distributed queries.

I. INTRODUCTION

All Recently, technological advancements that have driven down the price of handhelds, cameras, phones, sensors, and other mobile devices, have benefited not only consumers but the computational science community. As a result, a new field called data-driven computing is emerging, where computationally intensive applications often need real-time responses to data streams from distributed locations. These stream sources can have vastly varying generation rates and event sizes. Responsiveness, i.e., the ability of a data driven application to respond in a timely manner, is critical.

Stream query processing has been an active research area in recent years [1, 2] yet limited work has been done on query optimization for such high performance stream applications. Especially, to our knowledge, the core part of stream query optimization, i.e., the cost model, has not been systematically studied in this context.

As infinite event sequences, data streams introduce new challenges to query plan selection. First, since cardinality is not available for streams, the cardinality-based cost model loses its usefulness under the stream processing scenario. Second, data are not guaranteed to be fully processed. If a query processor does not process stream data in time, the data will be lost forever once they are removed from a buffer. Hence, unlike traditional query processing where all input data are processed, stream query processing may yield output based on a subset of input data events. Therefore, besides computation cost, output completeness, which is represented by output rate, is another important aspect for evaluating stream query plans.

In response to these challenges, develop a new cost model that factors in both output completeness and computation cost for stream query processing. Observe that these two metrics are not dependent variables, although they are relevant.

For the past twenty years, query optimization has been an intensively studied area of database system research. Most modern optimizers are cost-based in that they decide between execution plans by minimizing the estimated cost of evaluating the query. A fundamental technique used in cost estimation is cardinality estimation – optimizers take as input the cardinalities of tables at the leaves of a query tree, and then use selectivities of operators in the tree to estimate the cardinality of the input to operators further up in the tree. To convert cardinalities to costs, optimizers use functions that estimate the cost per tuple of each operator. While this approach is not perfect, it is very effective in most traditional DBMS applications. However, as we move to the Internet domain, this approach, in its current form, may not even apply. The reason for this is that if the leaves of the query tree correspond to incoming network streams, not only is their cardinality often not known, in some cases it may not even be well defined (e.g., in the case of infinite streams.) To allow the optimization of queries in the presence of streaming data, a new approach is needed. In this paper propose rate-based optimization for such applications.

The conventional approach to stream query processing used in many existing distributed data stream management systems [6], [7] consists of three consecutive phases: query planning, query deployment, and 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.

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. This limits the scope of the adaptation which aims at exploiting runtime environment properties to further optimize the efficiency of distributed stream query deliveries. One of the key ideas in our framework is to use hierarchical network partitions to scalable exploit various opportunities for operator-level reuse in the processing of multiple stream queries.

II. LITERATURE SURVEY

The seminal paper on cost-based query optimization was [8]. Other optimization models have been proposed, especially in the areas of parallel query optimization, using cost models that are not cardinality-based but instead deal with resource scheduling and allocation. The optimizer could optimize for the first result [9], while the optimization criterion was a combination of execution time and resource utilization. Modeling streaming behavior through input rates and modeling network traffic as Poisson random processes have appeared in many contexts, including [11], although to our knowledge it has not been applied in the context of query optimization.

A lot of work has been carried out in the areas of non blocking symmetric join algorithms [12], which aim at producing plans that do not block their execution because of slow input streams. Framework indicates that with variable rate sources it is beneficial to employ such algorithms. In the same context, methodologies aiming at avoiding blocked parts of an execution plan at runtime [13] can benefit from our framework of rate optimization by starting with and/or switching to plans for which the predicted output rate is maximized.

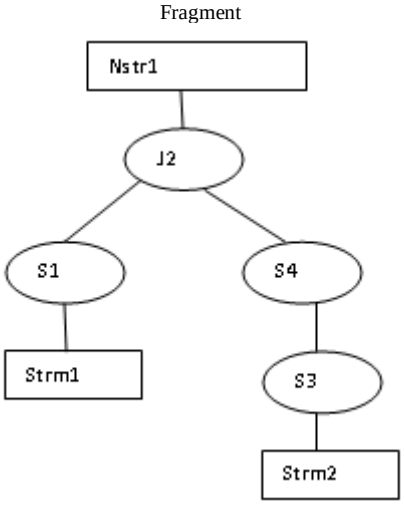

Figure 2.1 Query Fragment Set

The overload management problem in distributed stream processing systems has close relevance to the congestion control problem in computer networks [8].

Congestion in computer networks mainly arises when routers run out of buffer space, either because their processors cannot keep up with the incoming input data or because the outgoing link has a smaller bandwidth capacity than the incoming link [10].

The most closely related areas of work come from the adaptive query execution and dynamic re-optimization frameworks of [14] and [15]. In these frameworks, the main concern is to dynamically monitor an execution plan and identify points of sub-optimal performance. Once such points are identified, the system can choose to reorganize the plan in a way that is expected to yield better performance.

III. SYSTEM METHODOLOGY

Multiple continuous queries may be executing simultaneously and hundreds of nodes, distributed across multiple geographic locations are available for processing. In order to answer these queries, data streams from multiple sources need to be joined based on the attribute, perhaps using something like a symmetric hash join. The modern enterprise applications [16], scientific collaborations across wide-area networks and large-scale distributed sensor systems are placing growing demands on distributed streaming systems to provide capabilities beyond basic data transport such as wide-area data storage and continuous and opportunistic processing.

A. Problem definition:

Stream joins are performed using standard techniques 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. It assume that all such plans produce equivalent results.

The 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 research assume stream joins are performed using standard techniques that potentially, any operator can be deployed at any node in the system.

B. Stream Query Optimization Algorithms and Infrastructure:

Distributed query optimization, dynamic programming does not result in any pruning of the search space without loss of optimality since the query optimization problem in distributed data stream systems does not exhibit the property of optimal substructure [4].An optimal execution plan, traditional query optimizers typically perform an exhaustive search of the solution space using dynamic programming, estimating the cost of each plan using precomputed statistics.

The search space increases exponentially with an increase in the query size. Certainly, in a system with thousands of nodes, such an exhaustive search even with dynamic programming would be infeasible. In the case of distributed query optimization, dynamic programming does not result in any pruning of the search space without loss of optimality since the query optimization problem in distributed data stream systems does not exhibit the property of optimal substructure.

C. Hierarchical Network Clusters:

It organize physical network nodes into a virtual clustering hierarchy, by clustering nodes based on link costs which represents the cost of transmitting a unit amount of data across the link. Refer to this clustering parameter as internode or cluster traversal cost. Nodes that are close to each other in the sense of this clustering parameter are allocated to the same cluster. Allow no more than max_{cs} nodes per cluster

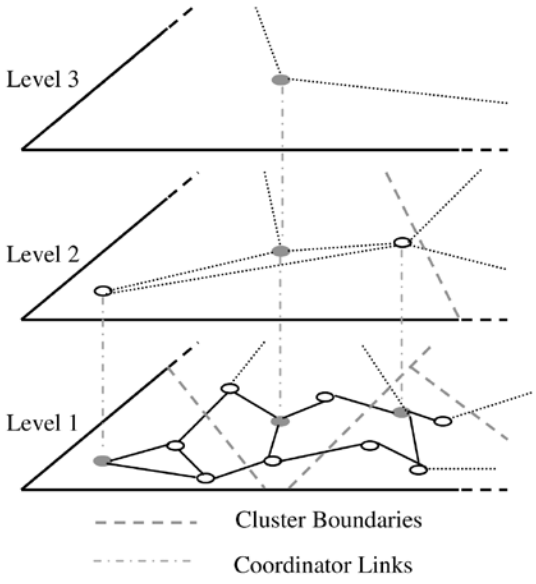

Figure 3.1. Hierarchical Network Clusters

At the lowest level, Level 1, the physical nodes are organized into clusters of max_{cs} or fewer nodes. Each node within a cluster is aware of the internode traversal cost between every pair of nodes in the cluster. A single node from each cluster is then selected as the coordinator node for that cluster and promoted to the next level, Level 2. There may be a set of nodes in a cluster, each of which qualifies to be a representative coordinator node as long as they do not modify the ordering of euclidean distances between the clusters. Nodes in Level 2 are again clustered according to average internode traversal cost, with the cluster size again limited by max_{cs}. This process of clustering and coordinator selection continues until Level N.

The request is propagated up the hierarchy and the top-level coordinator assigns it to the top-level node that is closest to the new node. This top-level node passes the request down to its child that is closest to the new node.

The virtual hierarchy is robust enough to adapt as necessary. It can handle both node joins and departures at runtime. Failure of coordinator nodes can be handled by maintaining active backups of the coordinator node within each cluster.

Situations where nodes in the entire system either are all widely distributed or are all close to one another in terms of network cost, may result in loosely defined clusters, which further impact the quality of coordinator nodes selected. Such situations are relatively rare. In the worst case, it is possible to choose appropriate values for max_{cs} in order to improve accuracy of the planning process. The node distribution in the network might possibly result in loosely defined clusters, it may be beneficial to compare planning decisions across multiple hierarchical structures with different values of max_{cs}.

D. Advertisements of Stream:

The stream advertisements are aggregated by the coordinator nodes and propagated up the hierarchy, as a result of the advertisement of derived stream sources, nodes are now aware of operators that are readily available at multiple locations in the network and can be reused with no additional cost involved for transporting input data. The advertisements are one-time messages exchanged only at the initial time of operator instantiation and deployment. The coordinator node at each level is aware of all the stream sources available in its underlying cluster. Advertisements of derived stream sources are key to operator reuse in our algorithms.

E. Top-Down Algorithm:

The Top-Down algorithm, the query starts at the top of the hierarchy, and is recursively planned by progressively partitioning the query and assigning subqueries to progressively smaller portions of the network. The Top-Down algorithm bounds suboptimality by making deployment decisions using bounded approximations of the underlying network, specifically, each coordinator's estimate of the distance between its cluster and other clusters. The algorithm works as follows, The query Q is submitted as input to the top-level (say level t) coordinator. The coordinator exhaustively constructs the possible query trees for the query, and then for each such tree constructs a set of all possible node assignments within its current cluster. The cost for each assignment is calculated and the assignment with least cost is chosen. An assignment of operators to nodes partitions the query into a number of views, each allocated to a single node at level t. Each node is then responsible for instantiating such a view using sources (base or derived) available within its underlying cluster.

F. The Bottom-Up Algorithm:

Describe the Bottom-Up algorithm which propagates queries up the hierarchy, progressively constructing complete query execution plans. Unlike the Top-Down approach, the Bottom-Up algorithm does not provide a good bound on the suboptimality of the solution. However, in return, the Bottom-Up approach is usually able to further reduce the search space compared to the Top-Down algorithm. Thus, in situations where quick planning is needed, the Bottom-Up algorithm may be appropriate, perhaps to be replaced later with a Top-Down deployment. Queries are registered at their sink. When a new query Q over base stream sources arrives at a sink at Level 1, the sink informs its coordinator at Level 2.

G. The NPC Algorithm:

Develop a heuristic-based hybrid algorithm that combines the strengths of both the Top-Down and Bottom-Up algorithms the Net Present Cost (NPC) algorithm. The NPC algorithm is a probabilistic algorithm that guides the planning process based on cost estimates of choosing a join order locally or delaying the decision. That combines the advantages

of reduced search space from the Bottom-Up algorithm and improved query planning from the Top-Down algorithm.

Then, the estimated NPC is computed as follows

$$\Omega l = \sum_{i=l+1} p_i \times h, \quad \lambda(Q) \times d_i$$

In order to compute Ω_l, expected future costs of delaying the query partitioning decision to the next level. The NPC algorithm then performs query partitioning at the current level l if $\Omega \geq T_l$. Unlike the other algorithms, the NPC algorithm requires knowledge of the hierarchical structure in terms of height, number of nodes in a cluster, and maximum intracluster traversal costs at each level. It also requires knowledge of join selectivities. Since the NPC algorithm attempts to avoid poor join orders, it is expected to perform better than the Bottom-Up algorithm. Since it continues to make query partitioning decisions based only on efficiency of join orders, oblivious to the availability of reuse opportunities, it is expected to produce less efficient deployments as compared to the Top-Down algorithm.

IV. EXPERIMENTAL RESULTS

Experiments focus on the effect of the max_{cs} clustering parameter on the trade-off between suboptimality and search space, the effectiveness of our algorithms as compared to existing approaches, and the efficiency of our algorithms compared to an optimal solution computed through an exhaustive search. Our experiments show that our algorithms result in acceptable suboptimality, the Top- Down algorithm is suboptimal by only 10 percent and the Bottom-Up algorithm by 34 percent while exploring less than 1 percent of the total search space. At the same time, our algorithms clearly outperform existing approaches. For example, the Bottom-Up algorithm reduces cost by nearly 25 percent when compared to the In-network [5] algorithm while exploring only a small fraction of the search space. Also, the NPC algorithm allows us to further fine tune the trade-off between search space and suboptimality and help us achieve plans that were close to the Top-Down algorithm in optimality and Bottom-Up algorithm in search space.

Stream query adopt the "network usage" metric [9] to compute costs of query deployments. Recall that, the network usage of a query q represents the total amount of data that is in-transfor a query at any given instant. As described in later , network is organized, into a virtual clustering hierarchy based on link costs which represent the cost of transmitting a unamount of data across the link. Used the hierarchical [8] clustering in order to create the clustering hierarchy.

A. Search space in Cluster Size:

An exhaustive search of all possible query plans and all possible placement of operators may not be feasible as network size increases. For example, an exhaustive search on a 128-node network for the deployment of a single query over five stream queries required enumeration of approximately 4:83* 1010 plans that took nearly 3 hs to complete. In this section, It demonstrate how the max_{cs} parameter can be used to tune the trade-off between the suboptimality of the heuristic and minimizing the search space. The experiments were conducted using the synthetic workload described in later.

B. Network Size in Scalablity:

This experiment, study the scalability of the algorithms with respect to the number of deployments considered as network size increases. It generated a workload of 100 queries using 10 streams with each query performing joins over n streams. Research measured the average number of deployments considered over n different transit-stub topologies of different sizes generated using sinks were placed at random nodes in the network.

Figure shows the deployments considered for a single query with Bottom-Up and Top-Down algorithms with maxcs 32 and exhaustive search. The figure also shows how the average case compares with the worst case analytical bounds. Again, the value of maxcs was set to 32 to produce the largest feasible search space.

Query processing that increase in Exhaustive is offset by the decrease in such that the worst case bounds are nearly identical across the different networks. Note that the y-axis has a log scale.

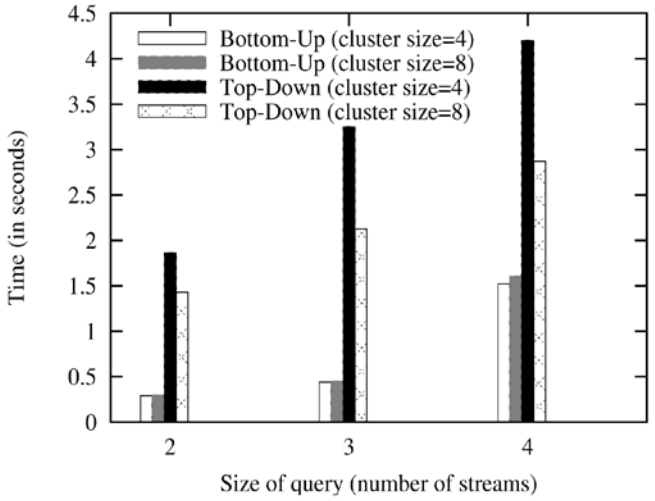

Figure 4.1. Network scalability

V. CONCLUSION

In Hierarchical network partitions that integrates query planning and distributed stream query optimization framework. The framework consists of two key components, a hierarchical clustering of network nodes that allows network approximations and stream advertisements that enable operator reuse. The network partitions algorithms are Top-Down, Bottom- Up, and Hybrid use the search space reduction. Which exploit different ways of using hierarchical network partitions for operator-level reuse and search space reduction. Show that although Top-Down and Bottom-Up algorithms can both choose efficient deployments while exploring only a small fraction of the search space, the Top Down algorithm is more effective in limiting the suboptimality of the solutions, while the Bottom-Up approach is more effective in reducing the search space and the time-to-deployment.

Stream advertisement and hierarchical clusters enable to use the operator reuse. The hybrid algorithm NPC find efficient execution plans while examining a very small search space, allowing us to further tune the trade-off between search space and algorithm suboptimality. Show through both experimental and analytical results that our algorithms are efficient and scalable at costs comparable to optimal while exploring much fewer plans.

VI. REFERENCES

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

[2] A. Arasu, B. Babcock, and et al. Stream: The stanford data stream management system. In Data Stream Management, 2004.

[3] R. Avnur and J. M. Hellerstein. Eddies: Continuously Adaptive Query Processing, Proceedings of the 2000 ACM SIGMOD Conference.

[4] D. Bertsekas and R. Gallager. Data Networks, Prentice Hall, 2nd edition, 1991.

[5] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. Niagaracq: A scalable continuous query system for internet databases. In SIGMOD Conference, pages 379–390, 2000.

[6] Z. G. Ives, D. Florescu, M. Friedman, A. Levy and D. S. Weld. An Adaptive Query Execution System for Data Integration, Proceedings of the 1999 ACM SIGMOD Conference.

[7] V. Oleson, K. Schwan, G. Eisenhauer, B. Plale, C. Pu, and D. Amin, "Operational Information Systems—An Example from the Airline Industry," Proc. First Workshop Industrial Experiences with Systems Software.

[8] G. Schumacher. GEI's Experience with Britton-Lee's IDM, IWDM, 1983, pp. 233-241.

[9] 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, Proceedings of the 1979 ACM SIGMOD Conference.

[10] M.A. Shah, J.M. Hellerstein, S. Chandrasekaran, and M.J. Franklin, "Flux: An Adaptive Partitioning Operator for Continuous Query Systems," Proc. 19th Int'l Conf. Data Eng. (ICDE), 2003.

[11] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin and Andrew Yu. Mariposa: A Wide-Area Distributed Database System, VLDB Journal, 1996, (5) 1:48-63.

[12] Stratis D. Viglas and Jeffrey F. Naughton, "Rate-Based Query Optimization for Streaming Information Sources", ACM SIGMOD'2002.

[13] T. Urhan and M. J. Franklin. Xjoin: A Reactively-Scheduled Pipelined Join Operator, IEEE Data Engineering Bulletin, June 2000, (23) 2:27-33.

[14] Y. Yao and J. Gehrke, "The Cougar Approach to In-Network Query Processing in Sensor Networks," SIGMOD Record, 2002.

[15] Yongluan Zhou, Karl Aberer, and Kian-Lee Tan, "Toward Massive Query Optimization in Large-Scale Distributed Stream Systems", Middleware 2008, LNCS 5346, pp. 326–345, 2008.

[16] D.J. Abadi et al., "The Design of the Borealis Stream Processing Engine," Proc. Second Biennial Conf. Innovative Data Systems Research (CIDR), 2005.