Query Optimization for Distributed Data Streams

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

Abstract

With the recent explosive growth of sensors and instruments, data-driven stream applications are emerging as a new field. Query optimization for such high performance stream applications has not been extensively studied, especially its core component, the *cost model*. We observe that the cost model for stream query processing should consider two aspects: *output rate* and *computation cost*. However, most existing work on evaluating stream queries uses only one or the other. In this paper, we propose a multi-model based optimization framework by leveraging both aspects. Further, we extend it from the centralized environment to the distributed environment by introducing distributed metrics and an algorithm for query plan decomposition.

keywords: stream data, query optimization, distributed stream processing

1 Introduction

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, 11, 7, 15], 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, we develop a new cost model that factors in both output completeness and computation cost for stream query processing. We observe that these two metrics are not dependent variables, although they are relevant. As Section 2.1 will discuss, an optimal plan satisfying the maximum output completeness may not have the minimum computation cost. In most applications, output completeness is more important. Therefore, our new cost model has the optimization goal as: *"Among a set of plans with the maximum output completeness, finding a plan with the minimum cost."*

Further, because streams are distributed, a fact that follows from wide-spread distribution of sensors, a centralized stream data management system is likely to result in limit performance. It is thus reasonable to build stream processing systems over distributed sites where the stream data are carrying out. Such distributed stream processing involves a new distributed query cost model and a new search algorithm.

Specifically, we extend a centralized cost model to include network bandwidth cost and computation workload of a distributed site. The new search space includes all possible centralized query plans and the corresponding distributed plans. Our search algorithm is accomplished in two steps: first searching for an optimal centralized plan with the centralized cost model; then evaluating all the distributed plans generated from the optimal centralized plan and choosing the one satisfying the optimization goal of the distributed cost model. The algorithms and heuristics developed as part of this research will be tested under a load of thousands to tens of thousands of streams using a scalable stream processing system [16] developed in our lab.

The remainder of this paper is organized as follows: Section 2 introduces the new rate-based cost model and the corresponding centralized optimization framework. Section 3 extends the centralized framework to distributed environment. Section 4 experimentally justifies our optimization framework. Section 5 discusses related work and Section 6 future research plans.

2 Centralized Multi-model Based Optimization

As the key in query optimization, cost model consists of an *optimization goal* and *operation cost estimations* for var-

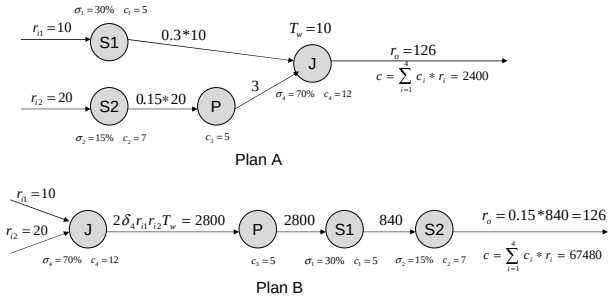

Figure 1: Observation1: Plans with same output rate but different costs.

ious query operators and their implementations. The optimization goal defines what the optimal plan is and cost estimation explains how to evaluate each query plan [8]. As Section 1 discussed, the cost model for stream query processing needs to consider both computation cost and output completeness. Towards this goal, we develop a multi-model based optimization framework. In this section, we give detailed description about this work in a centralized computing environment, covering both optimization goal and operation cost estimation.

2.1 Optimization Goal

The optimization goal of a cost model defines an optimal query plan. In traditional cost model, the optimal plan has minimum computation cost [18]. As a plan with less computation cost can save computation resources and make a system more scalable, minimum *computation cost* is still a necessary quality for an optimal query plan in stream query processing. Besides computation cost, as we discussed in Section 1, *output completeness* should also be counted to evaluate a stream query plan. We call the output based on partial input data set as incomplete output. The output completeness can be measured by *output rate*, the number of output events per time unit.

Therefore, an optimal query plan for stream query processing should have *maximum output rate* and *minimum computation cost*. Can we find a plan which satisfies both metrics at the same time? To answer this question, we study the relationship between these two metrics and have following observations. We use cost estimations defined in Section 2.2 to clearly explain our observations.

Observation1: Plans with the same output rate can have different computation costs. As shown in Figure 1, plan A and plan B are equivalent plans where nodes $S1(S2)$, P and J represent operators of selection, projection and join respectively. The individual operator's selectivity (σ) and cost (c) to process a unit event or event pair are marked beside the operator. The arrow lines denote intermediate streams whose rates are marked above the lines. Although these two plans have the same output rate r_o, planB's computation cost 67480 is 28 times higher than planA's cost 2400. Therefore, using only the maximum output rate as the optimization goal, we can not distinguish these two plans and may choose plan B which consumes much more computation resources. Hence, purely using the *maximum output rate* is inadequate to find an optimal stream query plan.

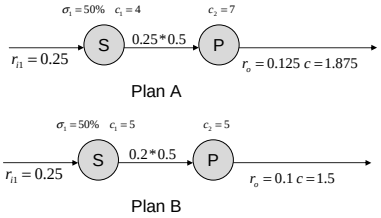

Figure 2: Observation2: Computation cost and output rate are not absolutely consistent.

Terminology	Symbol
Input Rate	r_i
Output Rate	r_o
Unit Cost for Selection	c_{su}
True value probability	p
Unit Cost for Projection	c_{pu}
Number of projection attributes	n
Selection Computation Cost	c_s
Projection Computation Cost	c_p
Join Computation Cost	c_j

Table 1: Symbol definition.

Observation2: Selecting a plan with reduced computation cost does not guarantee higher output rate. As shown in Figure 2, both S and P have different implementations in planA and planB, which cause their different costs. In planA, S is fast enough to let all input data pass smoothly. But planB has congestion in with slower S. This causes planA has higher output rate. But planB has lower cost than planA. Hence, a plan with lower computation cost does not necessarily have higher output rate. Therefore, having minimum computation cost as the only metric is not enough. A combined optimization goal is needed.

In summary, although computation cost and output rate are relevant, they are not dependent variables. Considering both of them, our new cost model has a goal as: *"Among a set of plans with the maximum output completeness, finding a plan with the minimum cost."* Formally, we assume that $cost()$ and $rate()$ are functions to compute a query plan's computation cost and output rate respectively. An optimal query plan P_o can thus be defined as follows:

$$P_o = \arg\min_{p \in \mathcal{P}}(cost(p))$$

where $\mathcal{P} = \arg\max_{s \in \mathcal{S}}(rate(s))$ and $\mathcal{S}$ is the search space of all equivalent plans. Note that $\mathcal{P}$ is a set of plans satisfying the maximum output rate.

2.2 Operation Cost Estimation

The evaluation of a stream query plan thus includes two aspects: output completeness and computation cost. In this section, we discuss how both are measured and estimated. In this paper, our focus is SPJ queries, the most common queries used in databases [8].

First, output completeness is measured by output rate: the number of output data events per time unit. Estimation of the output rate of a query plan is based on the output rate estimation of the basic query operators: selection, projection and join. We refine the rate estimation of Viglas

Operator	Output Rate	Computation Cost
Selection(r_i,σ,$\wedge$)	$r_o = \sigma * min(r_i, 1/c_s)$	$c_s = (1+p) * c_{su}$
Selection(r_i,σ,$\vee$)	$r_o = \sigma * min(r_i, 1/c_s)$	$c_s = (2-p) * c_{su}$
Projection(r_i)	$r_o = min(r_i, 1/c_p)$	$c_p = n * c_{pu}$
Join($r_{i1}, r_{i2}, T_w, \sigma$)	$r_o = min(2 * r_1 * r_2 * T_w, 1/c_j) * \sigma$	$c_j = c_{su} + \sigma * n * c_{pu}$

Table 2: Estimations for three basic operators.

and Naughton [19]. Especially, we give a new rate estimation for the sliding-window based join operation. Generally, for any operator, its output rate r_o is decided by minimum value of stream input rate r_i and query processing rate r_p. That is, $r_o = min(r_i, r_p)$.

In a traditional cost evaluation model, cost is measured in terms of the number of disk I/Os, because disk access causes principle cost there. In a stream query processing system, however, input data events arrive on a socket interface and from there pass through the query processor. All data is resident in main memory with swapspace incurring the only disk overhead. Therefore, *CPU computation cost* is the principle cost for a stream query. When queries are distributed over a wide area or over slow network links, communication time is also important. In Section 3, we discuss network bandwidth cost as well. We currently focus on the centralized case and consider the CPU load as the cost metric.

We estimate a query plan's computation cost analytically, similar to the approach taken in [18, 8]. Specifically, we first estimate the cost c_i for each operator i to process a unit event or event pair and the rate r_i of the data passing through this operator, then the whole query plan's cost is $C_q = \sum(c_i * r_i)$. Due to space limitations, we give only the result of output rate and computation cost estimation for three basic operators, shown in Table 2. The symbols are defined in Table 1. For more details, please refer to the extended report [14].

The cost model is the core of a centralized stream query optimization framework. Then, the remaining two parts of optimization framework are a plan search space and a plan search algorithm. Following the approaches used in traditional cost model [18, 12, 6], the search space includes all the logical query plans and equivalent physical plans. The search algorithm has two stages: first select the optimal logical query plan using heuristics [15], then evaluate all the equivalent physical plans derived from the optimal logical query plan. As this has been studied, we will not focus on this further.

3 Distributed Multi-model Based Optimization

The centralized optimization framework for stream query processing forms the basis of optimization techniques in a distributed environment. Streams are often produced at distributed sites. So, using queries to reduce data volume at the source is desirable. To deal with distributed stream query processing with our proposed query optimization framework, we must extend the cost model.

Terminology	Symbol
Weight for computation load	W_d
Weight for network bandwidth	W_b
Site computation load	d
Query's network bandwidth needs	b
Query's maximum bandwidth needs	max_b

Table 3: Symbols for distributed site selection.

3.1 Cost Model Extension

In a highly distributed environment where streams are available on distributed sites, processing data locally can achieve significant gains in performance by reducing expensive data transfer cost. Therefore, we propose to disassemble 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 can consume significant network bandwidth. Hence, both *network bandwidth* and *stream availability* need to be considered while evaluating a distributed query plan. At the same time, putting too many queries on one site can cause it overloaded. Hence, *balance of computation load* is also an important metric for distributed query evaluations.

So, for distributed stream query processing, we consider 3 extended metrics: *stream availability*, *network bandwidth* and *balance of computation load*. Stream availability and network bandwidth can be converted with each other: we can transfer a stream to a computation site by using extra network bandwidth. Thus, only network bandwidth and computation load are considered here. Intuitively, we would like to put a query fragment on the site with lighter computation load and larger available network bandwidth. By using the weighted sum, as shown in Equation 1, we select the site with highest score. The corresponding symbols are defined in Table 3.

$$score = (1-d) * W_d + \frac{b}{max\_b} * W_b \qquad (1)$$

3.2 Search Space Expansion

In traditional database query processing, for a given query, multiple different query plans are generated, which defines a search space. When data resources are highly distributed as they are in a stream application, the search space is expanded, because multiple distributed query plans can be derived from a centralized physical query plan. This reality extends the query searching space exponentially. That is, given a query tree $T =< V, E >$, there are 2^E decomposition plans. For assigning query fragments, given M fragments and N distributed sites, N^M possible assignment plans can be generated.

3.3 Search Algorithm

To address the search for an optimal query plan in an exponentially expanding search space, we take a *two-stage*

search algorithm. In the first stage, we search for the optimal centralized plan under the centralized cost model; then we evaluate all the distributed plans derived from this centralized plan and select the one satisfying the goal of the distributed cost model. The intuition behind such a two-stage algorithm is: an optimal centralized plan will filter data as early as possible to save unnecessary computations [15]; thus, the optimal centralized plan generally has less intermediate data stream transferring. Therefore, the distributed plans derived from an optimal centralized plan should have less network bandwidth cost.

Although our search space is significantly decreased by only focusing on distributed plans generated from an optimal centralized plan, it is still an exponential space as discussed before. Searching through such a space is not realistic. Meanwhile, we observe the heuristics: to save network bandwidth, we need to divide a query into as few fragments as possible and put a join operator on a site where both input streams are available. Based on such observations, we propose a heuristic-based greedy Algorithm 1 to find a distributed query plan given an optimal centralized query plan. Although distributed query search space is exponential, our greedy algorithm can find an optimal distributed plan in polynomial time, $O(logV) + O(N)$.

Algorithm 1 Distributed plan searching.

```
DPS(Tree root, SiteNumber n)
1: DPS(root.left, n)
2: DPS(root.right, n)
3: if root==JOIN then
4:     decompose tree at root to (F, RT)
5:     /*F: Fragment; RT: Remaining Tree */
6:     max_score=0; site=-1
7:     for i=1 to n do
8:         score = (1 − d) ∗ W_d + b/max_b ∗ W_b
9:         if score ≥ max_score then
10:            max_score=score; site=i
11:        end if
12:    end for
13:    return assignment pair (F,site)
14: end if
```

In Algorithm 1, we take recursive post-order traversal of a query tree. Whenever we meet a JOIN operator, we decompose a query fragment from there. Then, we study all the possible assignment for this query fragment based on Equation 1 and choose a site which has the minimum score . After we complete the traversal, an optimal distributed query plan is generated. In Section 4.3, we will show how the algorithm works through a case study.

Our work focuses on choosing the query plan which satisfies our optimization goal. Once we select the plan and deploy it into the system, in some cases, due to the variation of input streams' rates, we may need to dynamically adjust the plan to balance computation loads of different sites. For dynamic balancing of workloads, we can adopt existing approaches, e.g., [22, 23].

4 Experiments

We experimentally evaluate how the *Rate Model* selects query plans to demonstrate the need for a multi-model based optimization framework. We follow this with a case study where we show how the optimization framework works in a distributed environment.

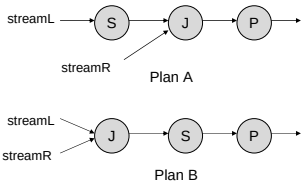

Figure 3: Equivalent plans for experiment1

Operator	Processing Time	Selectivity
S(Selection)	$0.5ms$	0.5
P(Projection)	$0.7ms$	1
J(Join)	$0.5ms$ (Steady Mode)	0.7
	$10ms$ (Overload Mode)	

Table 4: Operator parameters for experiment1.

4.1 Experiment Setup

To validate our cost model, we test continuous stream queries using dQUOB [16], a publicly available stream query processing system. The dQUOB system, implemented in C++, consists of a query parser, a query optimizer and a query execution engine. From a single query, we manually generate various relational algebra trees, which are converted to query graphs encoded as scripts that are deployed at a query processing engine. We generate input streams automatically using a publish/subscribe system dQUOBEC [20], in which each event has its unique *Timestamp* and *EventId*. The hardware setup includes a dual processor 2.8GHz workstation with 2GB memory running RedHat Enterprise Linux(RHEL).

4.2 Insufficiency of the Rate Model: An Empirical Justification

In Section 2.1, we analytically determined the insufficiency of the Rate Model: that is, it can only distinguish equivalent plans in *Overload Mode* where the processor does not have sufficient capacity to handle all the incoming data events. In this experiment, we experimentally justify the inefficiency and demonstrate that output rate can characterize the output completeness. We study two logically equivalent plans shown in Figure 3 and measure the output completeness of each as well as the output rate, both in *Overload Mode* and *Steady Mode* where the processor is at capacity. To simulate the Overload Mode, we delay "JOIN" service time to $10ms$ as shown in Figure 4. Other parameters in Figure 4 are measured from the dQUOB system [16]. By keeping the same stream input rates for two plans, we expect they yield the same number of output events, which makes our comparison more clear. The equivalent plans are shown in Figure 3. The input streams, streamL and streamR, are automatically generated. Since we are simulating scientific streams, streamL is generated with 13 attributes and streamR 8 attributes. Both streams have rate of $1\ event/ms$ and event size of $0.5MB$.

In the first part of this experiment, we compare the output rates of two query plans and measure their output

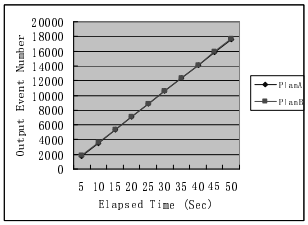

Figure 4: Output rate comparison (steady)

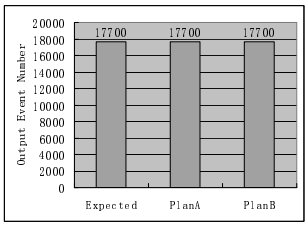

Figure 5: Output completeness comparison (steady)

completeness in the Steady Mode. To measure the output rates, we record the cumulative number of output data events at specific points in time during the experiment. The result in Figure 4, where the slope represents the output rate, shows that two plans have the same output rate under the Steady Mode. Note that the experiments are carried out on real streams and the experiments are begun after the system enters the stable stage. Therefore, there is no warm up stage in our result. Further, in Figure 5, we compare the number of output events for each plan with the expected output numbers based on 50000 sequential input events. This experiment shows that both plans reach 100% output completeness and have the same output rate in Steady Mode.

In the second part of the experiment, we carried out the same tests on the same two query plans in Overload Mode. Figure 6 shows that planA has higher output rate than planB. Further, from Figure 7, we can see that both plans did not reach full completeness due to the congestion in the Overload Mode. However, planA has more output events than planB, which is expected since planA has higher output rate than planB.

From these experiments, we draw two conclusions. First, output rate is a valid metric to measure a query plan's output completeness; second, the Rate Model can only distinguish equivalent plans in Overload Mode and all the equivalent plans should have the same output rate in Steady Mode, as long as they have same input rates. Hence, we can not only use Rate Model to evaluate query plans for stream query processing. Therefore, a multi-model optimization framework is needed.

4.3 Case Study: Distributed Query Plan Generation

In this section, we evaluate our distributed query plan search algorithm by means of a case study. Given an optimized centralized query plan, we show that a query can be decomposed into query fragments, which are assigned to distributed computation sites.

A centralized query plan is shown in Figure 8. As

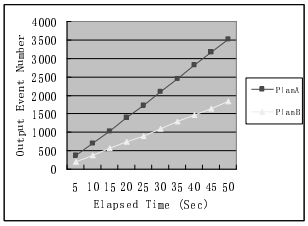

Figure 6: Output rate comparison (overload)

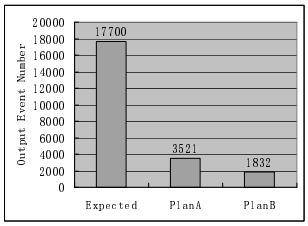

Figure 7: Output completeness comparison (overload)

given in Algorithm 1, the centralized plan is decomposed at the join operator. From this we obtain a set of query fragments shown in Figure 9. Nstr1 and Nstr2 are special nodes generated to connect these fragments. Figure 10 shows a mesh of computation nodes with distance weight on the edge. The state of each site is given in Figure 5. For each query fragment, we go through all the possible site assignments, compute the score for each site assignment based on the Equation 1, and choose the one having the highest score. Due to the space limitation, we skip the computation detail here. Among all possible assignments for *Fragment1*, site *S3* has the highest score. Hence, we distribute Fragment1 to site S3. Similarly, both *Fragment2* and *Fragment3* are assigned to site *S2*.

5 Related Work

Significant effort has been spent on the query optimization problem [18, 12, 6] in traditional databases by using the well-recognized *Cardinality-Based Cost Model*. However, we cannot simply adopt this optimization model for stream query processing because of the challenges brought by stream data, as we discussed in Section 1.

In recent years, several groups have studied query optimization for stream processing systems. STREAM [2] applies Synopsis Sharing within a single query or among multi-queries. NiagaraCQ [7] exploits incremental group optimization with expression signature. These techniques are brought out as a performance boosting approach and both systems are under a centralized environment. Borealis [1] has the QoS based multi-level optimization framework for a distributed stream query processing system. However, similar to [2, 1], it only considers the computation cost while evaluating a query plan. In a word, all above three systems, plus [13, 5, 10], evaluate a query plan only based on the computation cost. On the contrary, Viglas [19] just considers the output rate in his rate-based query optimization model. Ayad [4] recognizes resources constraints while maximizing the output rate. However, it does not give systematic model to include those two metrics in evaluating a stream query plan. Plus, it just focuses

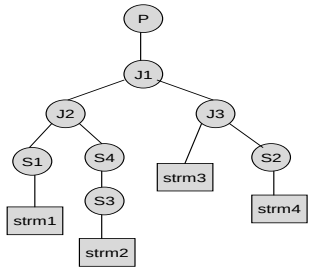

Figure 8: Optimal or sub-optimal centralized plan.

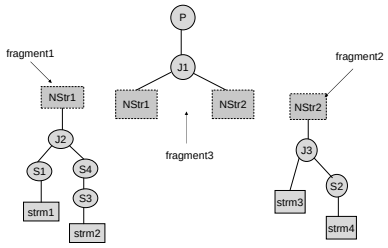

Figure 9: Query fragment set.

on a centralized query plan, while our framework can handle distributed data streams.

6 Conclusion and Future Work

In this paper, we proposed a multi-model based stream query optimization framework for a distributed stream query environment. There are a number of interesting directions to further study in the future.

First, we can model memory usage as another evaluation metric in our multi-model optimization framework. Some streams, especially scientific ones, include large-size data events. For example, an event from NexRad leveII streams from the WSR-88D Doppler radars can have an event of 1.7MB in size. Meteorology model data, called Eta data, can be 41.5MB [17]. Queries involving such streams easily face very tight memory resources, therefore finding a plan that minimizes memory usage is needed.

Second, this paper focuses on the join operator implemented with *time-based sliding window* when we estimate the output rate of a join operator. However, there are many other implementations and algorithms [3, 9, 21], e.g. count-based sliding window, pipelined hash join and etc, for the join operation in stream query processing. Although they share similar selectivity estimation and counting methods with the operators discussed in this paper, they indeed have their own unique characters. We plan to conduct more complete study on the rate/cost estimation of the operators with various implementations and algorithms.

Third, this paper focuses on static optimization by assuming that streams have steady rates. We plan to investigate adaptive optimization with our multi-model optimization framework under non-uniform stream rates.

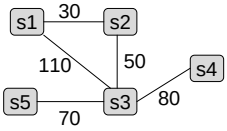

Figure 10: Graph of distributed sites.

Site	Avail-Workload	Avail-Bandwidth	Avail-Stream
S1	85%	$80Mbps$	strm2,3
S2	95%	$100Mbps$	strm1,3,4
S3	70%	$70Mbps$	strm1,2,3
S4	33%	$40Mbps$	strm1,2,3,4
S5	90%	$80Mbps$	strm1,2

Table 5: State of distributed sites.

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 Conference*, 2005.

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

[3] A. Arasu and J. Widom. Resource sharing in continuous sliding-window aggregates. In *30th VLDB Conference*, 2004.

[4] A. M. Ayad and J. F. Naughton. Static optimization of conjunctive queries with sliding windows over infinite streams. In *ACM SIGMOD international conference on Management of data*, 2004.

[5] S. Babu, R. Motwani, K. Munagala, I. Nishizawa, and J. Widom. Adaptive ordering of pipelined stream filters. In *ACM SIGMOD international conference on Management of data*, 2004.

[6] S. Chaudhuri. An overview of query optimization in relational systems. In *PODS Conference*, 1998.

[7] 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.

[8] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database System Implementation*. Prentice-Hall, 2000.

[9] L. Golab and M. T. Özsu. Issues in data stream management. *SIGMOD Rec.*, 32(2):5–14, 2003.

[10] L. Golab and M. T. Özsu. Update-pattern-aware modeling and processing of continuous queries. In *ACM SIGMOD international conference on Management of data*, 2005.

[11] M. A. Hammad, M. F. Mokbel, M. H. Ali, W. G. Aref, and et al. Nile: A query processing engine for data streams. In *ICDE conference*, 2004.

[12] Y. E. Ioannidis. Query optimization. *ACM Comput. Surv.*, 1996.

[13] J. Kang, J. Naughton, and S. Viglas. Evaluating window joins over unbounded streams. In *Int. Conf. on Data Engineering (ICDE)*, 2003.

[14] Y. Liu and B. Plale. Multi-model based optimization for stream query processing. In *KSI Eighteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'06)*, 2006.

[15] B. Plale and K. Schwan. Optimizations enabled by a relational data model view to querying data streams. In *IPDPS Conference*, 2001.

[16] B. Plale and K. Schwan. Dynamic querying of streaming data with the dquob system. *IEEE Transactions on Parallel and Distributed Systems*, 14(4), April, 2003.

[17] B. Plale and N. Vijayakumar. Evaluation of rate-based adaptivity in joining asynchronous data streams. In *19th IEEE International Parallel and Distributed Processing Symposium*, April 2005.

[18] 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. In *SIGMOD Conference*, 1979.

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

[20] N. Vijayakumar and B. Plale. dQUOBEC event channel communication system. Technical Report TR614, Indiana University, Computer Science Department, 2005.

[21] A. N. Wilschut and P. M. G. Apers. Dataflow query execution in a parallel main-memory environment. *Distributed and Parallel Databases*, 1(1):103–128, 1993.

[22] Y. Xing, S. Zdonik, and J.-H. Hwang. Dynamic load distribution in the borealis stream processor. In *ICDE Conference*, 2005.

[23] Y. Zhou, B. C. Ooi, and K.-L. Tan. Dynamic load management for distributed continuous query systems. In *ICDE Conference*, 2005.