Multi-Model Based Optimization for Stream Query Processing

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

Abstract

With recent explosive growth of sensors and instruments, large scale data-intensive and computation-intensive applications are emerging, especially in scientific fields. Helping scientists to efficiently, even in real time, process queries over those large scale scientific streams thus has great demand. However, query optimization for high volume stream applications– in particular its core component, the evaluation model– *has not been systematically studied. We observe that evaluating stream query plans should consider three aspects:* output rate, computation cost *and* memory consumption*. However, to our knowledge, no existing research on evaluating stream query plans consider all three metrics. In this paper, we propose a new combined optimization goal which leverages all these aspects and develop a multi-model based optimization framework to accomplish this goal. Specifically, we build three models to evaluate a plan's output rate, computation cost and memory consumption respectively. Based on such three models, we search for an optimal plan while considering systems's computation resource and memory constraints. We also experimentally evaluate our optimization framework.*

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, data-driven computing is emerging, where computationally intensive applications often need real-time responses to data streams from distributed locations. These streaming 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. For instance, out-of-date analysis of beam luminosity data can be useless or worse yet may mislead particle physicists. Therefore, in practice, performance plays an important role in stream query processing.

Stream query processing has been an active research area in recent years [1, 2, 8], yet limited work has been done on query optimization for such high volume stream processing. Especially, to our knowledge, the core part of stream query optimization, i.e., the *evaluation 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 here. Second, queries may not complete within designated service time because bursty streams or limited system resources.

In response to these challenges, we propose a multi-model based stream query optimization framework that considers three metrics: *output rate*, *computation cost* and *memory consumption*. Such a framework includes evaluation models to estimate three metrics respectively. As we will discuss in Section 3, our framework also considers a system's constraints on computation and memory resources. When a system's resource, either computation resource or memory resource, is insufficient to process a query within a designated service time, we reduce arrival rates by selectively dropping events. The selected plan is an optimal or sub-optimal one at the new input rates.

We observed that three metrics are relevant, but they are not dependent variables. For example, two equivalent plans with the same output rate may have different computation cost. As shown in Figure 1, planA and planB 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 an unit event or event pairs 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 of 67480 is more than 28 times higher than planA's cost of 2400. The symbols used here are formally defined in Section 2, and the computation formulas are discussed in Section 3.2. Similar relationship exists in any pair of metrics. Due to the space limitation, we will not enumerate all of them.

Therefore, stream query optimization should consider

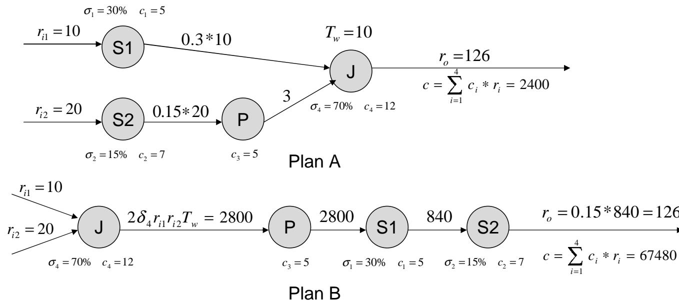

Figure 1. Plans with same output rate but different costs.

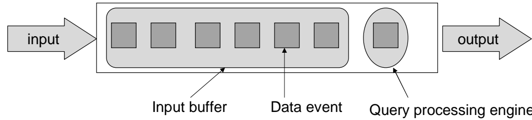

Figure 2. Stream query processing model.

the three metrics and have a combined optimization goal. In response to this, we propose a new optimization goal which considers all three metrics: output rate, computation cost and memory consumption. To accomplish such a goal, we conduct a multi-model based optimization framework, which has three evaluation models to estimate each individual metric. Further, we implement our optimization framework in Calder [20], a grid enabled stream processing system.

The remainder of this paper is organized as follows: In Section 2 we give preliminary definitions used in this paper. Section 3 introduces our multi-model based optimization framework which includes an optimization goal, a search algorithm and evaluation models. Section 4 reports experimental justification of the multi-model based stream query optimization. Section 5 discusses related work and Section 6, future research plans.

2 Preliminary Definition

In this section, we lay the groundwork with definitions that will be used in this paper.

2.1 Stream Query Processing Model

A stream S is defined as a sequence of events, $S = \{e_i\}$ where i is a monotonically increasing integer and $0 < i < \infty$. An *Event* is an atomic unit of data in the stream and the counterpart to a tuple of a traditional database table. In the stream processing system, queries are *continuous query* [6] which will remain in the engine for a period of time specified by the owner of the query and continuously evaluate incoming events. Similar to [10], our *Stream Query Processing Model*, shown in Figure 2, takes streams as input and produces streams. A *processing engine* holds continuous queries and processes stream data events sequentially.

Terminology	Symbol
Input Rate	r_i
Output Rate	r_o
Processing Rate	r_p
Cost for Unit Selection	c_{su}
Cost for Unit Projection	c_{pu}
Selection Cost per event	c_s
Projection Cost per event	c_p
Join Cost per event pair	c_j
Join Memory Consumption	m_j
Event Size	es

Table 1. Terminology definition.

2.2 Evaluation Model Terminology

Given two adjacent events e_j and e_{j+1} in a stream which arrive at time t_j and t_{j+1} separately, we define *Stream Rate* r as $r = 1/(t_{j+1} - t_j)$. Accordingly, r_i and r_o are the input or output *Stream Rate* of the query respectively. A query operator has a $ProcessingRate$ r_p which defines the number of events that can be processed by this operator per time unit.

As stream data pass through networks and most processing occurs within main memory, *computation cost* and *memory consumption* become two major consumed resources. In our evaluation model, computation cost, c, is defined as the amount of *CPU time* spent processing events in a *Unit time*. We use m_j to denote memory consumption for a $JOIN$ operation. To compute the memory consumption, we also define an event's size as es.

We further use c_s, c_p and c_j to represent the computation cost to process a data event by selection, projection and join operators respectively. As we will discuss in Section 3.2, these costs are based on the unit cost c_{su} and c_{pu}, where c_{su} is the unit cost to compare a condition and c_{pu} is the unit cost to copy an attribute.

Table 1 lists all the symbols defined in this section.

3 Multi-Model Based Optimization

3.1 Optimization Goal and Search Algorithm

As discussed in Section 1, our optimization framework has a combined goal as *Finding a plan with maximum output rate, minimum computation cost and minimum memory consumption, while considering a system's resources constraints.* In the following, we discuss how we select a plan satisfying such goal.

Resources consumed by stream processing include CPU cycles and memory resource. Considering such two resource constraints, there are four different scenarios shown in Table 2. In Scenario I, the system has sufficient resources to process all the input data. Therefore, as discussed in [14], the input events are fully processed and equivalent plans under such scenario have the same output rate. Scenarios II,

	Memory sufficient	Memory insufficient
CPU sufficient	Scenario I	Scenario II
CPU insufficient	Scenario III	Scenario IV

Table 2. Four scenarios with constraints.

III and IV all face resource constraints of one form or another. Under these scenarios, we reduce streams' input rates by selectively dropping input events to fit the system's resource constraints. This techniques can be referred as *sampling* [15], with which the output is not based on complete input set. However, as we drop events probabilistically, results are not biased.

Given a query, we can have many equivalent plans with different resource requirements. Hence, these equivalent plans may belong to different scenarios for a specific system. Plans under scenario II, III and IV are facing resource shortages. As discussed above, we reduce input streams' rates by sampling input events to move these plans to scenario I, where the system can provide adequate resource to the plan. Then, based on our optimization goal. We first search for plans with maximum output rate, as output rate indicates output completeness [14].

After we get a set of plans with the same maximum output rate, we intuitively want a plan with minimum computation cost and minimum memory consumption from the candidate set. However, in Section 1, we have pointed out that computation cost and memory consumption are independent variables. The ideal plan with minimum values on both metrics may not exist. For such dual-metric selection, we rely on users' preferences, which are specified when users submit queries.

Based on above analysis, our search strategy is summarized below. We first search for plans with maximum output rates. Then we do further selection based on users' preference. If users believe CPU resource is tighter than memory resource for their applications, we first find plans with minimum computation cost, then within the selected plans find the one with minimum memory consumption. Vice versa. Note, if sampling is needed due to resource constraints, all computation is based on decreased input rates caused by events dropping.

3.2 Evaluation Models

Besides the optimization goal and search algorithms, evaluation models are also indispensable for an optimization framework. An evaluation model defines a set of formula to estimate a plan's metrics. As we discussed in Section 1, our framework considers three metrics: output rate, computation cost and memory consumption, therefore we have an evaluation model for each of them.

3.2.1 Evaluation Model for Computation Cost

According to Table 1, c_s, c_p and c_j denote the cost of selection, projection and join operators respectively. In this section, we first show how we compute them based on unit selection and projection cost c_{su} and c_{pu}. We further discuss how we compute a query plan's cost C_q.

We know that there are mainly two ways to combine the basic select conditions: AND($\wedge$) or OR($\vee$). Therefore, we only study the following two cases: ($con1 \wedge con2$) and ($con1 \vee con2$). More complex selection cases can be derived based on these two basic cases. The cost are shown below, where p is the *True Value Probability*, which defines the probability of $con1$ being *true*. We adopt the heuristic rules discussed in [9] to estimate p.

$$\begin{aligned} c_{s1} &= p * (c_{su} + c_{su}) + (1 - p) * c_{su} \\ &= (1 + p) * c_{su} \\ c_{s2} &= p * c_{su} + (1 - p) * (c_{su} + c_{su}) \\ &= (2 - p) * c_{su} \end{aligned}$$

As for the projection's implementation, each event comes into the processing engine and the engine copies the selected attributes into the output event. We defines *Unit Projection Cost*, c_{pu}, as the cost to copy one attribute. This cost can be varied for the attributes with different data types. c_{pu} is assumed as the average value. With this assumption, a projection operator with the attribute list $(a_1, a_2 ... a_n)$ has the cost c_p as:

$$c_p = n * c_{pu}$$

Unlike the selection and projection operators, join operator is a binary full-relation operator. That is, the join operation needs the presence of the full relations. However, unlimited stream size makes traditional join algorithms not suitable for stream query processing. Consequently, many stream query processing engines implement join operation over a sliding window, instead of the full stream. Here, we only discuss the join cost c_j of a pair of input events. The complete cost for a join operation implemented on sliding windows can be computed based on c_j and event passing rate discussed in Section 3.2.2.

Given the selectivity of a join operation is σ, $R \bowtie S$ is actually equal to $_\pi(_\sigma(R \times S))$. That is, join operation is implemented based on selection and projection. Therefore, the join operator's *Computation Cost* to process an event pair, c_j, as defined in Section 2 should be:

$$c_j = c_{su} + \sigma * n * c_{pu}$$

Once we have each operator's computation cost for an event or event pair, we can get each operator's cost per time unit by multiply the operator i's cost c_i with its input rate r_i. Further, we can sum these cost together to get a query plan's cost C_q, which reflects how much CPU time consumed by the whole plan in a time unit.

$$C_q = \sum_{i=1}^{k}(c_i * r_i)$$

3.2.2 Evaluation Model for Output Rate

Viglas and Naughton [19] give a series of formulas to estimate the output rates for three basic query operators: selection, projection and join. However, their model requires computing the solution to an integral which is not efficient to evaluate a large number of query plans. Although they use rough heuristics to approximate integrals, this causes inaccurate estimations. Hence, we refine their models with a new estimation techniques on the join operation.

The join operator is a *Binary Full-Relation* operation. Traditional join algorithm does not work under streaming query processing scenario, where different *Sliding Window Based* join algorithms are applied. The *Output Rate* varies with different algorithms. In this paper, we study the join operator that executes natural join over a time-based sliding window [3], where the window size T_w is defined by the time interval. Other join implementations have similar analysis. We plan to do more complete study for various operator implementations in the future, as we will discuss in Section 6.

As shown in [17, 16], a time-based sliding window can be implemented at low cost, and when tied to input stream rate, can be more intuitive for users than a count-based sliding window, where the window size is defined as the number of interested events.

Suppose two streams of events R and S. R is composed of events e_i: $R = \{e_i\}$ and $0 < i < \infty$. i is a monotonically increasing number. Similarly, $S = \{e_j\}$ and $0 < j < \infty$. Further, suppose stream R has an input rate of r_1 and S an input rate of r_2.

There are $r_1 * T_w$ events at any time residing in the sliding window of stream R and $r_2 * T_w$ events residing in the sliding window of stream S. Without loss of generality, we take as the observation period the time interval $[t_0, t_0 + t]$. During this period, $r_1 * t$ events will arrive on stream R, each of which will be joined by Cartesian product to all events in stream S's sliding window. Given the selectivity σ, those joins generate $r_1 * t * r_2 * T_w * \sigma$ output events during the observation interval. An event output from a join is a tuple $< e_i, e_j >$ where $e_i \in S$, $e_i \in R$. Examining this from the side of stream S instead, with similar analysis we obtain $r_2 * t * r_1 * T_w * \sigma$ events output during the observation interval t.

Putting together, there are

$$\begin{aligned} r_1 * t * r_2 * T_w * \sigma + r_2 * t * r_1 * T_w * \sigma \\ = 2 * (r_1 * r_2 * t * T_w * \sigma) \end{aligned}$$

events generated during such an observation period.

Operator	Output Rate
Selection(r_i,σ)	$r_o = \sigma * min(r_i, r_p)$
Projection(r_i)	$r_o = min(r_i, r_p)$
Join($r_{i1}, r_{i2}, T_w, \sigma$)	$r_o = min(2 * r_1 * r_2 * T_w, r_p) * \sigma$

Table 3. Output rate estimations.

Therefore, the output rate is the number of events divided by the observation time t, that is:

$$\begin{aligned} r_o &= 2 * (r_1 * r_2 * t * T_w * \sigma)/t \\ &= 2 * r_1 * r_2 * T_w * \sigma \end{aligned}$$

Table 3 lists output rate estimations for three kinds of basic operators.

3.2.3 Evaluation Model for Memory Consumption

As for memory consumption, we only consider memory used to keep state information. That is, we do not consider the memory consumed for queueing events between operators. Within an SPJ query which is the focus of this paper, only the $JOIN$ operator needs to maintain states to join two input streams. Therefore, we only discuss memory consumption of join operators.

As for join implementation, we still follow the time-based sliding algorithm discussed in Section 3.2.2. Given sliding window size as T_w on both sides, we will need memory $r_1 * es_1 * T_w$ and $r_2 * es_2 * T_w$ on either side respectively, where r_1, r_2 are input rates and es_1, es_2 are event size of two input streams. Put them together, we will get the total memory needs of this join operator:

$$m_j = (r_1 * es_1 * T_w) + (r_2 * es_2 * T_w)$$

4 Experiments

In this section, we first experimentally evaluate our cost model. Then, we show how our multi-model optimization framework differentiate three logically equivalent plans.

4.1 Experiment Setup

All the testing programs are written in C++. To simulate the system's memory constraints, we use a global variable to count the consumed memory. When it reaches the limit, we start dropping input events to simulate real memory constraints. As for the computation cost constraints, we simulate that by adding the delay in the operators. The hardware setup includes a dual processor 2.8GHz workstation with 2GB memory running RedHat Enterprise Linux(RHEL).

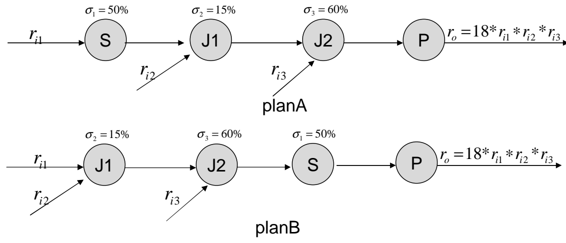

Figure 3. Equivalent SPJ plans.

Plan	Estimated Cost	Real Cost
planA	$9.458ms$	$5.654ms$
planB	$61.4ms$	$56ms$

Table 4. Cost model evaluation.

4.2 Evaluation of Cost Model

As discussed in Section 3, evaluation models are the basis of our optimization framework. Without accurate estimation of plans' metrics, we can not correctly choose an optimal plan. Hence, we first experimentally evaluate our cost model by studying two equivalent SPJ plans, which are shown in Figure 3. Three input streams, stream1, stream2 and stream3, are involved in this query. Their event formats contain 13, 8 and 12 name-value pairs respectively, plus a vector of data. All three input streams are generated as the rate of $1event/sec$ and each input event has the size of $0.5MB$.

We estimate the computation cost of two plans using the cost model introduced in Section 3.2.1. Meanwhile, we measure two plans' computation costs by recording all of their operators' processing time during an observation period of $T = 60sec$. In this experiment, we use a sliding window size of $T_w = 10sec$ for both join operators. All three input streams arrive at the rate of $1event/sec$. The estimated costs and measured costs are compared in Table 4. From the results, we can see that the plan yielding the lowest estimated cost, which is the one that would have been chosen, has the lowest actual cost, too. These results generalize to other SPJ queries which are not presented here.

Our cost model not only correctly orders two plans, but also accurately shows the difference between two plans.

4.3 Multi-Model Framework

In this experiment, we experimentally demonstrate that the three metrics used in the multi-model framework are necessary. Three equivalent multi-join plans are given in Figure 4. The parameters of join operators and input streams are listed in Table 5 and Table 6. These plans run on a system which has $256MB$ memory limitation.

Based on our evaluation models, we estimate each plan's needs of computation cost and memory. The result shows that only planB, which requires $234MB$ memory and $0.42sec$ CPU time per second, is under the scenarioI: the system can provide it adequate computation and memory resources. Both planA and planC, which have resource requirements as ($23MB$, $1.06sec$) and ($210MB$, $4.1sec$) respectively, fail to satisfy the computation cost constraints, therefore, our framework will choose planB based on the search algorithm discussed in Section 3.

We run all three plans in the system with the resource constraints as specified and measure output rates. Figure 5 illustrates the results, from which we can see that planB has higher output rate. Therefore, planB is an optimal plan as our framework expects.

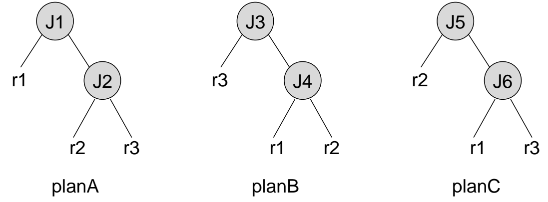

Figure 4. Equivalent multi-join plans.

Operator	Selectivity	Computation Cost
J1	0.3	$0.04ms$
J2	0.3	$0.025ms$
J3	0.45	$0.025ms$
J4	0.2	$0.05ms$
J5	0.9	$0.5ms$
J6	0.1	$0.025ms$

Table 5. Parameters of join operations.

5 Related Work

Literature is rich with work on query optimization in traditional databases [18, 12, 7]. However, traditional optimization models are not appropriate for stream query processing because of the challenges raised by stream processing, as discussed in Section 1.

Several groups have contributed to the literature for query optimization in stream systems. STREAM [2] applies Synopsis Sharing within a single query or among multi-queries. NiagaraCQ [8] exploits incremental group optimization with expression signature. Borealis [1] has designed the QoS based multi-level optimization framework for a distributed stream query processing system. However, all three systems, plus [13, 5, 11], consider the computation cost while evaluating a query plan. Viglas and

Stream	Input Rate	Event Size
r1	$1event/sec$	$1MB$
r2	$1event/sec$	$20KB$
r3	$10event/sec$	$2KB$

Table 6. Parameters of input streams.

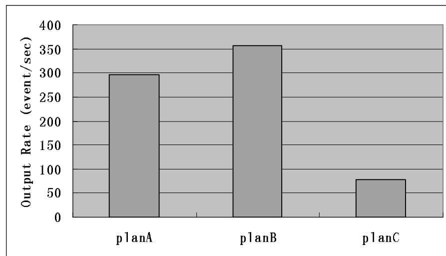

Figure 5. Output rate comparisons.

Naughton [19] consider the output rate in their rate-based optimization model. Ayad and Naughton [4] recognized resource constraints while maximizing the output rate, but do not consider memory consumption which is an important metric for stream processing in the scientific applications. Our multi-model based optimization framework factors in three metrics: output rate, computation cost and memory consumption.

6 Conclusion and Future Work

In this paper, we present our multi-model based optimization framework for stream query processing, which has a combined optimization goal and three evaluation models. With our framework, we can find an optimal plan not only for the the system with adequate resources, but also for the system with resource constraints.

In our study for this multi-model based optimization framework, we also observed some open issues that warrant further research. First, all above work is on a centralized system. However, 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 limited performance. Therefore, extending such a multi-model query optimization framework to a distributed environment is ongoing work. Second, this paper focuses on the *time-based sliding window* algorithm for join operation. However, there are many other implementations and algorithms [3, 10, 21], e.g. count-based sliding window, pipelined hash join and etc. We plan to conduct more complete study on the rate/cost estimation of the operators with various implementations and algorithms.

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, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom. 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. Babu and J. Widom. Continuous queries over data streams. *SIGMOD Record*, 30(3):109–120, 2001.

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

[8] 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.

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

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

[11] 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.

[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. Query optimization for distributed data streams. In *Manuscript submitted*, 2006.

[15] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, resource management, and approximation in a data stream management system. In *CIDR Conference*, 2003.

[16] B. Plale. Leveraging run time knowledge about event rates to improve memory utilization in wide area data stream filtering. In *IEEE HPDC Conference*, 2002.

[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 (IPDPS)*, 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, Y. Liu, and B. Plale. Calder query grid service: Insights and experimental evaluations. In *CCGrid Conference*, 2006.

[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.