

Optimizing Away Joins on Data Streams

Nick Koudas¹, Divesh Srivastava², and David Toman³

¹ University of Toronto

koudas@cs.toronto.edu

² AT&T Labs–Research

divesh@research.att.com

³ University of Waterloo

david@uwaterloo.ca

Abstract. Monitoring aggregates on IP traffic data streams is a compelling application for data stream management systems. Often, such streaming aggregation queries involve joining multiple streams (e.g., streams of SYN and ACK packets) using temporal join conditions (e.g., within 5 seconds), followed by computation of aggregates (e.g., COUNT) over temporal tumbling windows (e.g., every 5 minutes). While such a query expression is natural, its evaluation over high speed IP traffic data streams is infeasible in practice. In this paper, we develop rewriting techniques for streaming aggregation queries that identify conditions under which such joins can be optimized away, while providing error bounds for results of the rewritten queries. The basis of the optimization is a powerful but decidable theory in which constraints over data streams can be formulated. The result error bounds are specified as functions of the boundary effects incurred during query rewriting.

1 Introduction

The phenomenon of data streams is real. In data stream applications, data arrives very fast and the rate is so high that one may not wish to (or be able to) store all the data. Yet, the need exists to analyze this data.

The quintessential application seems to be the processing of IP traffic data in the network (see, e.g., [4, 19]). Routers forward IP packets at great speed, spending typically a few hundred nanoseconds per packet. Processing the IP packet data for a variety of monitoring tasks at the speed at which packets are forwarded is an illustrative example of data stream processing. One can see the need for aggregation queries in this scenario: to provide simple statistical summaries of the traffic carried by a link, to identify normal activity vs activity under denial of service attack, etc. Often, such streaming aggregation queries involve joining multiple streams (or, substreams of the same packet stream). For example, a common IP network analysis query (to help detect a type of denial of service attack) is “for every 5 minute interval, report the total number of SYN packets in that interval that do not have a matching ACK packet within 5 seconds”. While such a query expression is natural, its exact evaluation would require the data stream management system to (a) store every SYN packet seen in the stream until it finds a subsequent matching ACK packet or 5 seconds have elapsed (whichever

is smaller), and (b) match every ACK packet seen in the stream with a previous SYN packet (within the past 5 seconds). This is infeasible, in practice, both from storage and computational perspectives, especially over high speed IP traffic data streams.

In this paper, we develop rewriting techniques for streaming aggregation queries that identify conditions under which such expensive joins can be *optimized away*. On the previous query, the rewritten optimized query is “for every 5 minute interval, report the difference between the total number of SYN packets and the total number of ACK packets in that interval, provided this difference is positive”. Notice that the rewritten query does not join (i.e., correlate) individual SYN packets with matching ACK packets. Only the total number of SYN packets and the total number of ACK packets in each 5 minute interval need to be computed and maintained. This can be done much more efficiently, by maintaining running totals.

Of course, this optimized query is not necessarily equivalent to the original query. The reported differences may be lower or higher than the number of SYN packets in that interval without matching ACK packets. Intuitively, however, this error cannot be large under reasonable constraints on the arrival pattern of SYN and ACK packets in intervals. We generalize these intuitions and make the following contributions in this paper:

1. We define a powerful but decidable theory in which constraints over data streams can be formulated. The proposed theory is more powerful than temporal integrity constraints studied so far; for a more detailed discussion see Sections 3 and 6.
2. We develop approximate rewriting techniques for streaming aggregation queries that allow elimination of joins based on the above constraints. We show error bounds for results of the rewritten queries.
3. We present experimental evidence that complements our analytical results, using real IP data. The experiments show that the query rewrites proposed in the paper improve query performance by more than an order of magnitude, even in an RDBMS.

The rest of the paper is organized as follows. Section 2 gives a motivating example and outlines the applications of the technical development in Sections 3 and 4. Section 5 provides experimental results. Related work is presented in Section 6.

2 Motivating Example

We model data streams as relations with a fixed schema in which tuples are timestamped by the time of arrival in the stream. The timestamp is represented by a `time` attribute in the schema of the relations.

In our motivating example we consider a stream of TCP packets, collected on the fly from, e.g., a network router. Building upon the intuitive example described in the introduction, we consider three of its sub-streams, the sub-stream of the TCP SYN packets (that originate a TCP connection from a client), the sub-stream of TCP SYN-ACK packets (that acknowledge the original SYN packets), and the sub-stream of the corresponding ACK packets that complete the 3-way TCP handshake for establishing a TCP connection. We represent these three sub-streams using three relational schemes, SYN,

SYN-ACK, and ACK, with attributes *ip* standing for the *IP addresses* (for simplicity of exposition we encapsulate the parts representing the source and destination addresses, ports, etc., in a single attribute), and *time* standing for the *timestamp* (the time of arrival of the tuple, in seconds, in the respective stream). We would like to answer the query

For each 5 minute interval, how many SYN packets in that interval have a matching SYN-ACK packet, but do not have a matching ACK packet within 5 seconds?

This query can be used, e.g., to identify a SYN-flood based DOS attack and can be formulated (in SQL syntax) over the above schemes as follows:

```
SELECT  tb, count(*) as cnt
FROM    SYN s, SYN-ACK sa
WHERE   s.ip = sa.ip
        AND sa.time >= s.time AND sa.time - s.time <= 5
        AND NOT EXISTS
          ( SELECT *
            FROM   ACK a
              WHERE sa.ip = a.ip AND a.time >= sa.time
                AND a.time - sa.time <= 5 )
GROUP BY s.time/300 as tb
```

However, this query requires executing a rather expensive join and anti-join operations between the three streams. While many efficient algorithms for implementing joins on data streams have been proposed [12, 17, 24], at high streaming speeds, such joins became less and less feasible [10].

In this paper, we pursue a more indirect approach to evaluating the above query: we observe the high degree of collocation of packets along the temporal dimension and use this observation to replace the above query by the query

```
SELECT  s.tb, min(s.cnt,sa.cnt) - a.cnt as cnt
FROM    ( SELECT  tb, count(*) as cnt
          FROM    SYN
          GROUP BY time/300 as tb ) s,
        ( SELECT  tb, count(*) as cnt
          FROM    SYN-ACK
          GROUP BY time/300 as tb ) sa
        ( SELECT  tb, count(*) as cnt
          FROM    ACK
          GROUP BY time/300 as tb ) a
WHERE   s.tb = sa.tb AND sa.tb = a.tb
        AND min(s.cnt,sa.cnt) - a.cnt > 0
```

Intuitively, the second query simply computes an arithmetic expression over independent counts of SYN, SYN-ACK and ACK packets in a 5 minute interval, provided this difference is positive. This query is vastly preferable from the point of view of efficiency as it completely eliminates the need for the join operation between individual SYN and

SYN-ACK packets, and the (anti-)join operation between the result of the previous join operation and ACK packets: the three counts can be computed on the fly and the WHERE condition that equates the `tb` attributes merely states that every 5 minutes the arithmetic expression over the counters is computed and the counters are reset.

However, in general, the two queries are not necessarily equivalent (or even close). Thus, we study *integrity constraints* that are satisfied by data streams, that make the above transformation possible. We observe (and use) the following constraints, which can be obtained from TCP specifications [14] or using mining techniques [18].

- In the SYN, SYN-ACK and ACK streams, the `ip` attribute can serve as an identifier of a TCP connection (a key) for the duration of a connection (but it is not a key in general as there may be multiple consecutive connections between a particular source-destination pair of IP addresses).
- For every normal TCP connection, there is a single SYN, a single SYN-ACK, and a single ACK packet. However, in abnormal circumstances (e.g., in the DOS attack case) some of these packets (e.g., the ACK packet in the DOS attack case) may be missing.
- The SYN, SYN-ACK and ACK packets belonging to the same TCP flow are temporally collocated in the three streams and appear, say, at most 1 sec apart.

These constraints must have been known (at least intuitively) to the user when formulating the *original* query, in particular, when specifying the *5 second window* for the streaming join. Note also, that all the above constraints hold in *perfect* TCP flows. In practice, the constraints hold only approximately, i.e., for most parts of the streams. In the example of TCP flows, this is mainly due to network latencies. There are two ways to solve this problem:

- use a more complex but precise specification that accounts for the latencies, or
- use more intuitive constraints that are satisfied by most of the stream.

While the first solution may seem preferable from the theoretical point of view, the complexity of developing comprehensive descriptions that account for all possible problems in, e.g., a network is prohibitive and the computational properties of such theories are often quite poor.

Continuing with our example, the errors induced by the rewritten query can be traced to two main sources:

1. **Boundary effects:** In our example incurred by dividing the time line into 5 minute buckets and counting independently: the (SYN, SYN-ACK) pairs and (SYN-ACK, ACK) pairs that *cross* this boundary are not accounted for. Note that this error cannot be completely avoided by, e.g., shifting or expanding the 5 minute window for SYN-ACK and ACK packets by a few seconds, as we would incur the same error for the matching packets that arrive closer together.
2. **Approximate satisfaction of integrity constraints:** In our setting incurred, e.g., by asserting that all ACK packets arrive no later than 1 sec after the corresponding SYN-ACK packet. In practice, due to network delays, this constraint only holds for the majority of the packets, but not all packets. If, e.g., ACK packets arrive more

than 5 sec after the corresponding SYN-ACK packets, but are in the same 5 minute bucket, the and ACK packets that arrive too late will not be accounted for in the original query, but would in the rewritten query.

Note also that these errors cannot be eliminated altogether as the `ip` attribute serves as a TCP flow identifier for a limited period of time (and thus, on noisy networks, the exact accounting on lost/superfluous packets is not possible, save reproducing the whole TCP state machine [14] in the query).

While our main motivation is monitoring networks, the technique is applicable in many other situations in which properties of a single (conceptual) entity are monitored at different time instants. For example:

- counting the number of passengers in a subway system: here the streams of events are the *entry* and *exit* times for each passenger. Indeed, the commonly employed solution that counts independently the entries and exits is justified by the appropriate integrity constraints (e.g., passengers aren't born in the subway) and our query rewrites.
- monitoring complex protocols: the technique is not restricted to pairing together events in network streams. For example, the integrity constraints and query rewrites can be applied to monitor the *two-phase commit* (2PC) protocol for irregularities, e.g., for attempts to commit transactions for which one of the participants did not vote *yes*.

We next discuss our theory of integrity constraints, before describing how these are used in our query rewrites in Section 4.

3 Stream Integrity Constraints

For a fixed theory of linearly ordered time we define stream constraints as follows:

Definition 1 (Constraints) Let $R_1, \dots, R_k, S_1, \dots, S_l$ be predicate symbols (not necessarily distinct), ψ and ψ' conjunctions of atomic equalities, and φ and φ' formulas in the theory of time. We define stream integrity constraints to be formulas of the form:

$$\forall \mathbf{t} \forall \mathbf{x}. R_1(t_1, \mathbf{x}_1) \wedge \dots \wedge R_k(t_k, \mathbf{x}_k) \wedge \varphi_{\mathbf{t}} \wedge \psi_{\mathbf{x}} \\ \rightarrow \begin{cases} \exists \mathbf{t}'. \psi'_{\mathbf{t}, \mathbf{t}'} \wedge S_1(t'_1, \mathbf{y}_1) \wedge \dots \wedge S_l(t'_l, \mathbf{y}_l) \\ \psi'_{\mathbf{x}} \\ \varphi'_{\mathbf{t}} \end{cases}$$

for $\mathbf{y}_1 \cup \dots \cup \mathbf{y}_l \subseteq \mathbf{x} = \mathbf{x}_1 \dots \mathbf{x}_k$ vectors of data variables, $\mathbf{t} = \{t_1, \dots, t_k\}$, and $\mathbf{t}' = \{t'_1, \dots, t'_l\}$ to be time variables. The subscripts of the φ and ψ formulas in the constraints indicate the sets of allowable free variables in these formulas.

Given a finite set of constraints Σ and a constraint C , an implication problem $\Sigma \models C$ is a question whether C is true in all models of Σ .

The above constraints can be thought of as *temporal* variants of functional dependencies and embedded inclusion dependencies. Note however, that the temporal part of the constraints can use arbitrarily complex formulae in the theory of time—this arrangement makes this approach much more expressive than virtually any temporal integrity constraints proposed in the literature so far [5, 9, 15]. In the rest of the paper we allow combining right hand sides of constraints with the same antecedent by conjunction; this is mere syntactic convenience. We also omit the external universal quantifiers when presenting integrity constraints.

Example 2 Continuing with our running example, we formally specify the stream constraints outlined in Section 2. However, to be able to specify that the `ip` attribute is an identifier of a TCP flow within a limited time window, we need to extend the schema of the three streams with an additional *conceptual attribute* that stands for the identifier of TCP flows. Note that the *connection identifier* attribute in each of the schemes does not really exist in TCP flows; it is solely used for specifying integrity constraints that have to hold in a stream or between streams. The modified schema (fixing the positions of the attributes) is as follows:

$$\text{SYN}(\text{time}, \text{id}, \text{ip}), \text{SYN-ACK}(\text{time}, \text{id}, \text{ip}), \text{ and } \text{ACK}(\text{time}, \text{id}, \text{ip}).$$

For our running example, the constraints needed are as follows:

- (i) $\text{SYN}(t_1, i, a_1) \wedge \text{SYN}(t_2, i, a_2) \rightarrow a_1 = a_2 \wedge t_1 = t_2$
- (ii) $\text{SYN-ACK}(t_1, i, a_1) \wedge \text{SYN-ACK}(t_2, i, a_2) \rightarrow a_1 = a_2 \wedge t_1 = t_2$
- (iii) $\text{ACK}(t_1, i, a_1) \wedge \text{ACK}(t_2, i, a_2) \rightarrow a_1 = a_2 \wedge t_1 = t_2$
- (iv) $\text{SYN}(t_1, i, a_1) \wedge \text{SYN-ACK}(t_2, i, a_2) \rightarrow a_1 = a_2 \wedge t_2 \geq t_1$
- (v) $\text{SYN-ACK}(t_1, i, a_1) \wedge \text{ACK}(t_2, i, a_2) \rightarrow a_1 = a_2 \wedge t_2 \geq t_1$
- (vi) $\text{SYN}(t_1, i_1, a) \wedge \text{SYN-ACK}(t_2, i_2, a) \wedge (t_2 \geq t_1) \wedge (t_2 - t_1 \leq 10) \rightarrow i_1 = i_2$
- (vii) $\text{SYN-ACK}(t_1, i_1, a) \wedge \text{ACK}(t_2, i_2, a) \wedge (t_2 \geq t_1) \wedge (t_2 - t_1 \leq 10) \rightarrow i_1 = i_2$
- (viii) $\text{ACK}(t_1, i, a) \rightarrow \exists t_2. \text{SYN-ACK}(t_2, i, a) \wedge (t_1 \geq t_2) \wedge (t_1 - t_2 \leq 1)$
- (ix) $\text{SYN-ACK}(t_1, i, a) \rightarrow \exists t_2. \text{SYN}(t_2, i, a) \wedge (t_1 \geq t_2) \wedge (t_1 - t_2 \leq 1)$

stating that (i-iii) `id` is a *true* key, (iv-v) `id` is also a *foreign* key, (vi-vii) `ip` is a key within 10 second windows, and (viii-ix) `ACK` packets arrive within 1 second after the corresponding `SYN-ACK` packet, and `SYN-ACK` packets arrive within 1 second after the corresponding `SYN` packet (note the use of the `id` attribute here).

In general, the integrity constraints that enable the query rewriting may not be given explicitly but rather are logical consequences of other constraints given for a stream (or streams). Thus we need to be able to decide whether a constraint is a logical consequence of a given constraint theory.

3.1 Correspondence Theorem

First we show how to convert an implication problem to a decision problem in the associated theory of time. We need several (technical) definitions. These definitions are used to simulate the effects of the data values in the constraints. In particular, for every

predicate symbol $R(t, \mathbf{x})$ and every substitution $[\mathbf{a}/\mathbf{x}]$ of constants drawn from a finite set A for the variables \mathbf{x} we define a unary predicate symbol $R^{\mathbf{a}}(t)$; the collection of these unary predicates represents $R(t, \mathbf{x})$ in the result of the transformation.

Definition 3 Let σ be a schema and A a finite set of constants. We define

$$\sigma(A) = \{R^{\mathbf{a}}(t) : R(t, \mathbf{x}) \in \sigma, \mathbf{a} \in A^{|\mathbf{x}|}\}$$

$$\text{AUX}_A^\sigma = \{E^{a,a}, E^{a,b} \leftrightarrow E^{b,a}, E^{a,b} \wedge E^{b,c} \rightarrow E^{a,c}, \\ \forall t. E^{a,b} \rightarrow (R^{\mathbf{a}\mathbf{a}\mathbf{b}}(t) \leftrightarrow R^{\mathbf{a}\mathbf{b}\mathbf{b}}(t)) : a, b, c \in A, R^{\mathbf{a}\mathbf{a}\mathbf{b}}, R^{\mathbf{a}\mathbf{b}\mathbf{b}} \in \sigma(A)\}.$$

The additional $E^{a,b}$ propositions simulate the effects of equality in the original formulae: whenever constants a and b are forced to be equal in a model of the original constraints, e.g., as a consequence of a functional dependency, the proposition $E^{a,b}$ is true in the corresponding model on the transformed theory. The set AUX_A^σ captures the interactions between the $E^{a,b}$ propositions and the remainder of the translation.

The symbols defined above are used to transform constraints in the constraint theory Σ as follows:

Definition 4 Let C be a constraint, A a finite set of constants, and θ a substitution for variables \mathbf{x} with values from A . We define

$$C\theta = \forall \mathbf{t}. R_1^{\mathbf{x}_1\theta}(t_1) \wedge \dots \wedge R_k^{\mathbf{x}_k\theta}(t_k) \wedge \varphi_{\mathbf{t}} \wedge (\psi_{\mathbf{x}}\theta) \\ \rightarrow \begin{cases} \exists \mathbf{t}'. \psi'_{\mathbf{t},\mathbf{t}'} \wedge S_1^{\mathbf{y}_1}(t'_1) \wedge \dots \wedge S_l^{\mathbf{y}_l}(t'_l) \\ (\psi'_{\mathbf{x}}\theta) \\ \varphi'_{\mathbf{t}} \end{cases}$$

where $(\psi_{\mathbf{x}}\theta)$ is the formula ψ in which each atomic subformula $x_i = x_j$ is replaced by a proposition $E^{x_i\theta, x_j\theta}$ and $R_i^{\mathbf{x}_i\theta}, S_j^{\mathbf{y}_j\theta} \in \sigma(A)$ are unary predicates in the theory of time.

Given a theory Σ for the schema σ , we define a set of formulas

$$\text{SAT}_A(\Sigma) = \{C\theta : C \in \Sigma, \theta \text{ a substitution for } \mathbf{x} \text{ with values from } A\}.$$

For an implication question $\Sigma \models C$, the consequent C is transformed into a contrapositive form by Skolemizing all quantifiers over data variables as follows:

Definition 5 Let C be a constraint and $\{x_1, \dots, x_k\}$ the set of (universally quantified) data variables in C . We define A_C to be the set $\{a_1, \dots, a_k\}$ of distinct constants and $\text{NSAT}(C) = \neg(C[a_1/x_1, \dots, a_k/x_k])$.

The implication question $\Sigma \models C$ is then transformed into a satisfiability problem.

Theorem 6 Let Σ be a set of constraints and C a constraint over the schema σ . Then $\Sigma \models C$ if and only if $\text{AUX}_{A_C}^\sigma \cup \text{SAT}_{A_C}(\Sigma) \cup \{\text{NSAT}(C)\}$ is not satisfiable.

Proof. Consider first that $\text{AUX}_{A_C}^\sigma \cup \text{SAT}_{A_C}(\Sigma) \cup \{\text{NSAT}(C)\}$ is satisfiable and thus has a model T with a domain dom_T . In T , the propositions $E^{a,b}$ define an equivalence relation on A_C . We designate a canonical value for each of the equivalence classes. We say that a substitution θ is canonical if it only uses canonical values in A_C .

Now we construct a structure M as follows:

$$R_i(t, \mathbf{x}\theta) \text{ is true in } M \iff R_i^{\mathbf{x}\theta}(t) \text{ is true in } T \text{ for } \theta \text{ canonical and } t \in \text{dom}_T.$$

It is easy to verify that $M \models \Sigma$, assuming otherwise leads to a contradiction with $T \models \text{AUX}_{A_C}^\sigma \cup \text{SAT}_{A_C}(\Sigma)$. However, $M \not\models C$ as otherwise we would have $T \models \text{NSAT}(C)$.

For the converse, consider a structure M such that $M \models \Sigma$ and $M \not\models C$. Then, for C to be falsified, there must be a substitution $[a_1/x_1, \dots, a_k/x_k]$ that makes the precondition of C true and the consequent false in M . Let $A = \{a_1, \dots, a_k\}$ and substitutions θ range over A . We construct a structure T setting

$$R_i^{\mathbf{x}\theta}(t) \text{ is true in } T \iff M \models R(t, \mathbf{x}\theta) \text{ is true in } M$$

for θ a substitution and $t \in \text{dom}_M$. We also make $E^{a,a}$ true and $E^{a,b}$, $a \neq b$, false in T for $a, b \in A$. Then $T \models \text{AUX}_{A_C}^\sigma$ (trivially) and $T \models \text{SAT}_{A_C}(\Sigma)$ since falsifying $C_i\theta \in \text{SAT}_{A_C}(\Sigma)$ would also falsify $M \models C_i$. $T \models \text{NSAT}(C)$ follows immediately from the construction.

$\text{AUX}_{A_C}^\sigma \cup \text{SAT}_{A_C}(\Sigma) \cup \{\text{NSAT}(C)\}$ is a monadic formula in the theory of time (with one quantifier alternation in addition to alternations present in the temporal parts of the constraints). Thus, in combination with the results from [6], we have:

Corollary 7 *Let the theory of time be the theory of one successor function (S1S). Then the logical implication problem is decidable.*

Proof. Immediate by observing that the second-order existential closure of the conjunction of formulas in $\text{AUX}_{A_C}^\sigma \cup \text{SAT}_{A_C}(\Sigma) \cup \{\text{NSAT}(C)\}$ is an S1S sentence.

The choice of a very powerful logic to serve as the basis for reasoning about time in the proposed stream constraints allows the user to specify (non first-order) properties of data streams, e.g., periodic events [16], time granularities [5], etc., in addition to the more common temporal keys and functional dependencies [26].

3.2 Complexity of Reasoning

In general, the constraints imposed on the temporal dimension can be arbitrary complex S1S formulas, yielding a (tight) non-elementary complexity bound even for deciding satisfiability of a single constraint. However, the size of individual constraints is commonly rather small⁴ and therefore we are mainly concerned with the *number of constraints* resulting from the *instantiation* by Skolem constants:

⁴ For commonly studied theories, e.g., FDs, MVDs, or IDs in the relational setting, the size of the individual constraints is often bounded by a function depending only on the size of the relational schemes (signatures) and is generally considered to be fixed.

Lemma 8 *The size of $\text{AUX}_{AC}^\sigma \cup \text{SAT}_{AC}(\Sigma) \cup \{\text{NSAT}(C)\}$ is exponential in the (maximal) number of variables in a constraint C and polynomial in the number of constraints.*

Thus, following the standard construction of a Büchi automaton [6] for SIS and assuming essentially *constant* size of the automata for the individual constraints, we end with an automaton roughly exponential in the size of the constraint theory. This is no worse than other schema languages proposed for database systems. Note also that logical implication for the theory of full dependencies combined with functional dependencies alone, a non-temporal sub-theory of our constraint theory, is already EXPTIME-complete [7] and the temporal sub-theory, restricted to universally-quantified Horn clauses (Datalog_{1S}) is PSPACE-complete [8].

4 Query Transformation

We consider queries based on applying an aggregate operator (in particular, the count operator) on a result of join and/or anti-join operations. Satisfaction of stream integrity constraints is the prerequisite for each of the rewrites. We assume that we have been given a stream constraint theory, Σ , that describes the data streams involved in the queries.

The rewrites are formulated for a pair of streams, S_1 and S_2 , with a common schema. For sake of simplicity, the schema uses three generic attributes, $(\text{time}, \text{id}, \text{a})$, standing for the time instant (time of arrival in the stream), a (possibly virtual) identifier of corresponding tuples, and *other* attributes in the tuples arriving in the streams (denoted here by a single attribute a). This arrangement simplifies the exposition of the rules without limiting their applicability (cf. Example 2).

4.1 Window Predicate Elimination

The first rewrite proposed is used to eliminate window predicates (in the WHERE clause) by rediscovering the true identities (e.g., the actual TCP flows) to which individual items arriving on the stream(s) belong to. Note that here we introduce a conceptual attribute id to represent this identifier. The attribute is used solely for conceptual reasoning, in particular it allows us to formulate the necessary integrity constraints, and is eliminated later using the join/anti-join rules.

The window predicate elimination rule is defined as follows. The selection condition

$$\text{WHERE } S_1.\text{a} = S_2.\text{a} \text{ AND } S_1.\text{time} - S_2.\text{time} \leq \delta$$

that relates two data streams simplifies to

$$\text{WHERE } S_1.\text{id} = S_2.\text{id}$$

if

$$\Sigma \models \left\{ \begin{array}{l} S_1(t_1, x_1, y) \wedge S_2(t_2, x_2, y) \wedge t_1 - t_2 \leq \epsilon \rightarrow x_1 = x_2 \\ S_1(t_1, x, y_1) \wedge S_2(t_2, x, y_2) \rightarrow y_1 = y_2 \wedge t_1 - t_2 \leq \epsilon' \end{array} \right\}$$

for $\epsilon \geq \delta \geq \epsilon'$. Note that the second required constraint is not given explicitly in Example 2 but is implied in our running example by stating, e.g., that there is only one ACK packet (*id* is a key for the ACK stream) and that the ACK packet arrives at most one second after the matching SYN-ACK packet.

Proof. (sketch) Having two tuples that satisfy the first selection condition, their *id* attribute values must be the same due to the first constraint as $\epsilon \geq \delta$. On the other hand, two packets that agree on the *id* attribute and thus satisfy the second selection condition, must, by the second constraint, also satisfy the first selection condition, in particular the window condition as $\delta \geq \epsilon'$.

4.2 Join Elimination

The join/anti-join elimination rules are used to completely remove the join/anti-join from the query and replace it by an arithmetic expression that involves independent counts over the involved streams. Note the crucial use of the conceptual *id* attribute in the integrity constraints enabling this rewrite. The join elimination rule reads as follows. The query

```
SELECT  tb, count(*) as cnt
FROM    S1, S2
WHERE   S1.id=S2.id
GROUP BY S1.time/k as tb
```

rewrites to

```
SELECT  tb, min(s1.cnt,s2.cnt) as cnt
FROM    ( SELECT  tb, count(*) as cnt
          FROM    S1
          GROUP BY time/k as tb ) s1,
        ( SELECT  tb, count(*) as cnt
          FROM    S1
          GROUP BY time/k as tb ) s2
WHERE   s1.tb=s2.tb AND
        min(s1.cnt,s2.cnt)>0
```

if

$$\Sigma \models \left\{ \begin{array}{l} S_1(t_1, x, y_1) \wedge S_1(t_2, x, y_2) \rightarrow y_1 = y_2 \wedge t_1 = t_2 \\ S_2(t_1, x, y_1) \wedge S_2(t_2, x, y_2) \rightarrow y_1 = y_2 \wedge t_1 = t_2 \\ S_1(t_1, x, y) \rightarrow \exists t_2. S_2(t_2, x, y) \wedge 0 \leq t_2 - t_1 \leq \epsilon \end{array} \right\}$$

with a boundary error bounded by ϵ/k (assuming uniformity of packet arrival over k time units).

Proof. (sketch) The *id* attribute is the key for both streams S_1 and S_2 , there can be at most one pair of tuples that agree on *id*. The inclusion dependency constraint postulates that there indeed must be exactly one such pair for each tuple in S_1 (or S_2). Thus the minimal value of the independent counts is indeed equal to the count of tuples in the result of the join.

4.3 Anti-Join Elimination

Similarly to the *join case* we can treat anti-joins:

```

SELECT  tb, count(*) as cnt
FROM    S1
WHERE   NOT EXISTS
        ( SELECT *
          FROM  S2
          WHERE S1.id=S2.id )
GROUP BY S1.time/k as tb

```

rewrites to

```

SELECT  tb, s1.cnt-s2.cnt as cnt
FROM    ( SELECT  tb, count(*) as cnt
          FROM    S1
          GROUP BY time/k as tb ) s1,
        ( SELECT  tb, count(*) as cnt
          FROM    S1
          GROUP BY time/k as tb ) s2
WHERE   s1.tb=s2.tb AND
        s1.cnt-s2.cnt>0

```

if

$$\Sigma \models \left\{ \begin{array}{l} S_1(t_1, x, y_1) \wedge S_1(t_2, x, y_2) \rightarrow y_1 = y_2 \wedge t_1 = t_2 \\ S_2(t_1, x, y_1) \wedge S_2(t_2, x, y_2) \rightarrow y_1 = y_2 \wedge t_1 = t_2 \\ S_2(t_2, x, y) \rightarrow \exists t_1. S_1(t_1, x, y) \wedge 0 \leq t_2 - t_1 \leq \epsilon \end{array} \right\}$$

with a boundary error bounded by ϵ/k (again, assuming uniformity over k time units).

Proof. (sketch) Since the *id* attribute is a key for both streams, for each S_1 tuple there is at most one S_2 and for each S_2 there is exactly one S_1 tuple in the streams and thus the difference of the independent counts is equal to the count of tuples in the set difference (note that here we use the *conceptual* *id* attribute in a crucial way as the *real* attributes of tuples arriving on the streams do not have this property). The first aggregate subquery gives the exact count of the S_1 tuples and the second one the count of matching S_2 tuples not accounting for tuples within the last ϵ time units.

Note that the symmetric variant in the anti-join case is vacuous as it implies that the result is empty (due to the constraint stating that for every S_2 tuple there must be at least one corresponding S_1 tuple).

Note also that it is relatively easy to see that instances of streams that violate the integrity constraints make the above rewrites not valid.

4.4 Multiple Streams and Complex Queries

While the rules presented so far have been specified in terms of two data streams, it is easy to extend the technique to multiple streams whose items refer to the same conceptual entities.⁵ Using a natural inductive argument, the rewrites can be extended to all queries of the form

⁵ Otherwise, there is little point of asking queries that correlate the streams.

```

SELECT  tb, count(*) as cnt
FROM    Q'
GROUP BY S1.time/k as tb

```

where Q' is an arbitrary combination of joins and anti-joins over a finite set of input streams, such that the integrity constraints required by the atomic rewrites in Sections 4.1–4.3 are satisfied for each operator in Q' ; this allows applying the rules top-down ultimately eliminating all joins and anti-joins. Note that the integrity constraints are preserved under the common join reordering and thus we are not restricted to searching for a particular plan for Q' to perform the rewriting.

Example 9 The rewrite used in our example in Section 2 uses the composition of *window elimination*, *join elimination* and *anti-join elimination* transformations utilizing the constraints given in Example 2.

In practice, however, the rules should be integrated in a stream query optimizer where their application is determined by the optimizer’s search strategy and cost model.

5 Experiments

In this section, we report the results of experiments we conducted in order to evaluate the performance of our optimization technique. Using a prototype packet sniffer we collected 300K SYN and ACK packets along with timestamp information and (source and destination) IP addresses. In this data set, pairs of SYN and ACK packets are in one-to-one correspondence, i.e., there are no missing ACK packets. We imported the data set into MySQL version 4.1 using the default configuration parameters the software ships with. The original and optimized queries are simpler versions of the queries in the motivating example, involving only the SYN and ACK schemas.

Figure 1 presents the response time of the original (unoptimized) query versus the optimized version. Both relations are fully indexed (the index construction is included in the total response time). It is evident that response time increases with increasing data set size. The performance benefits of the optimized query are substantial (more than an order of magnitude). These benefits are observed even in an RDBMS. In a data stream management system monitoring IP network traffic, our optimization would enable the efficient evaluation of a query which could otherwise not be evaluated at all.

6 Related Work

Integrity constraints for time-dependent data have been studied in the area of temporal databases. For example *temporal functional dependencies* have been studied by Jensen *et al.* [15] and extended to accommodate granularities of time by Wang *et al.* [25] and Wijzen [26]. The main goal of these approaches was to develop tools for modeling of temporal databases, e.g., the ERVT data model [1] that supports time-stamping and evolution constraints, IS-A links, and disjointness and covering constraints. However, none of these approaches provides constraints expressive enough to capture the properties of

Fig. 1. Performance of Optimized and Un-optimized Queries

data streams enabling the rewrites proposed in this paper. The proposed constraint theory is a combination of a powerful logic for linear time, SIS [6], with generalized full dependencies, both tuple- and equality-generating [7]. The technique used to combine these theories is based, in part, on Datalog_{1S} [8].

The (Anti-)Join-Count query rewriting rule proposed in this paper rule is fundamentally different from aggregate-join pushdown rules proposed, e.g., by Paulley&Larson [20], Gupta *et al.* [13], Srivastava *et al.* [21], or DeHaan *et al.* [11]: all the above use functional dependencies (under constraints with varying expressiveness) to identify whether a subquery of a join functionally determines all grouping attributes. This allows commuting the grouping-aggregation with the join. In our approach, the join operation is completely eliminated (replaced with a scalar operation) based on an inclusion dependency. Note also that the groups in our case are, in general, not functionally determined by either of the join subqueries.

Efficient techniques for evaluating joins over streaming data have been studied in the past [12, 17, 24]. Our technique, however, is not trying to improve on the algorithms for executing joins but rather on eliminating the joins altogether. As pointed out in the introduction, at gigabit speeds, this may be the only option.

Also, the proposed technique is complementary to techniques proposed for optimization of streaming queries, based, e.g., on rate of delivery [2], utilization of resources [3, 22], or quality of service [23], as the application of the proposed optimization rules can be integrated with an appropriate search strategy and a cost function in a query optimizer.

7 Conclusions

Streaming aggregation queries often involve joining multiple streams using temporal join conditions, followed by computation of aggregates over temporal tumbling windows. While such a query expression is natural, its evaluation over high speed IP traffic data streams is infeasible in practice. In this paper, we develop rewriting techniques

for streaming aggregation queries that allow optimizing away such joins, while providing error bounds for results of the rewritten queries. The basis of the optimization is a powerful but decidable theory in which constraints over data streams can be formulated.

The errors we have studied in this paper are due to boundary effects, assuming perfect satisfaction of integrity constraints. An interesting direction of future work is to analyze errors due to approximate satisfaction of constraints, and their propagation during query transformation, based on logical inference.

References

1. Alessandro Artale, Enrico Franconi, and Federica Mandreoli. Description Logics for Modelling Dynamic Information. In Jan Chomicki, Ron van der Meyden, and Gunter Saake, editors, *Logics for Emerging Applications of Databases*. Lecture Notes in Computer Science, Springer-Verlag, 2003.
2. Ahmed Ayada and Jeffrey F. Naughton. Static Optimization of Conjunctive Queries with Sliding Windows Over Infinite Streams. In *ACM SIGMOD International Conference on Management of Data*, pages 419–430, 2004.
3. Brian Babcock, Shivnath Babu, Mayur Datar, and Rajeev Motwani. Chain: Operator Scheduling for Memory Minimization in Data Stream Systems. In *ACM SIGMOD International Conference on Management of Data*, pages 253–264, 2003.
4. Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Symposium on Principles of Database Systems (PODS)*, pages 1–16, 2002.
5. Claudio Bettini, Sushil Jajodia, and Xiaoyang Sean Wang. *Time Granularities in Databases, Data Mining, and Temporal Reasoning*. Springer, 2000.
6. J. Richard Büchi. On a Decision Method in Restricted Second Order Arithmetic. In *International Congress on Logic, Methodology, and Philosophy of Science*, pages 1–11, 1962.
7. Ashok K. Chandra, Harry R. Lewis, and Johann A. Makowsky. Embedded Implicational Dependencies and their Inference Problem. In *ACM Symposium on Theory of Computing (STOC)*, pages 342–354, 1981.
8. Jan Chomicki. Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding. *ACM Transactions on Database Systems*, 20(2):149–186, 1995.
9. Jan Chomicki and David Toman. Temporal Databases. In Michael Fisher, Dov Gabbay, and Lluís Vila, editors, *Handbook of Temporal Reasoning in Artificial Intelligence*. Elsevier, 2005.
10. Charles D. Cranor, Theodore Johnson, Oliver Spatscheck, and Vladislav Shkapenyuk. Gigascope: A Stream Database for Network Applications. In *ACM SIGMOD International Conference on Management of Data*, pages 647–651, 2003.
11. David DeHaan, David Toman, and Grant E. Weddell. Rewriting Aggregate Queries using Description Logics. In *Description Logics 2003*, pages 103–112. CEUR-WS vol.81, 2003.
12. Lukasz Golab and M. Tamer Özsu. Processing sliding window multi-joins in continuous queries over data streams. In *International Conference on Very Large Data Bases (VLDB)*, pages 500–511, 2003.
13. Ashish Gupta, Venky Harinarayan, and Dallen Quass. Aggregate-query processing in data warehousing environments. In *International Conference on Very Large Data Bases (VLDB)*, pages 358–369, 1995.
14. Information Sciences Institute. RFC 793, 1981. Edited by Jon Postel.

15. Christian S. Jensen, Richard T. Snodgrass, and Michael D. Soo. Extending Existing Dependency Theory to Temporal Databases. *IEEE Transactions on Data and Knowledge Engineering*, 8(4), 1996.
16. Froduald Kabanza, J-M. Stevenne, and Piere Wolper. Handling Infinite Temporal Data. *Journal of Computer and System Sciences*, 51(1):3–17, 1995.
17. Jaewoo Kang, Jeffrey F. Naughton, and Stratis Viglas. Evaluating Window Joins over Unbounded Streams. In *International Conference on Data Engineering (ICDE)*, pages 341–352, 2003.
18. Flip Korn, S. Muthukrishnan, and Yunyue Zhu. Checks and Balances: Monitoring Data Quality Problems in Network Traffic Databases. In *International Conference on Very Large Data Bases (VLDB)*, pages 536–547, 2003.
19. Nick Koudas and Divesh Srivastava. Data Stream Query Processing: A Tutorial. In *International Conference on Very Large Data Bases (VLDB)*, page 1149, 2003.
20. Glenn N. Paulley and Per-Åke Larson. Exploiting Uniqueness in Query Optimization. In *International Conference on Data Engineering (ICDE)*, pages 68–79, 1994.
21. Divesh Srivastava, Shaul Dar, H. V. Jagadish, and Alon Y. Levy. Answering queries with aggregation using views. In *International Conference on Very Large Data Bases (VLDB)*, pages 318–329, 1996.
22. Utkarsh Srivastava and Jennifer Widom. Memory-Limited Execution of Windowed Stream Joins. In *International Conference on Very Large Data Bases (VLDB)*, pages 324–335, 2004.
23. Nesime Tatbul, Ugur Cetintemel, Stanley B. Zdonik, Mitch Cherniack, and Michael Stonebraker. Load Shedding in a Data Stream Manager. In *International Conference on Very Large Data Bases (VLDB)*, pages 309–320, 2003.
24. Stratis Viglas, Jeffrey F. Naughton, and Josef Burger. Maximizing the Output Rate of Multi-Way Join Queries over Streaming Information Sources. In *International Conference on Very Large Data Bases (VLDB)*, pages 285–296, 2003.
25. Xiaoyang Sean Wang, Claudio Bettini, Alexander Brodsky, and Sushil Jajodia. Logical Design for Temporal Databases with Multiple Granularities. *ACM Transactions on Database Systems*, 22(2):115–170, 1997.
26. Jef Wijsen. Temporal FDs on Complex Objects. *ACM Transactions on Database Systems*, 24(1):127–176, 1999.