IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

MULTI QUERY OPTIMIZATION AND ITS STRENGTH

Bhanu Shanker Prasad

P G Deptt of Statistics and Computer Application,TMBU,Bhagalpur,Bihar,India

Designation:- M.D.N H/S SCHOOL,DUMRAMA ,AMARPUR,BANKA,BIHAR.INDIA

(Computer Science +2 Teacher)

Abstract :-

Data Stream Management System usually have to process many long running queries that are active at the same time .Multi query can be evaluated more efficiency together than independently .Because it is often possible to share state and computation . By this observation , various multi query optimization (MQO) technique have been proposed, but having some limitations like they focus on very specialized workloads and integrating MQO technique for stream engine.

The purpose of this paper is to demonstrate a rule based MQO framework that incorporate a set of new abstractions , physical operators, extending their counterparts , transformation rule and stream processing system.

Keywords:- stream , query , observation , optimization , abstractions , transformation

INTRODUCTION

Query optimizers have been instrumental for the success of relational database technology. The cost difference between a good and a bad query plan can be several orders of magnitude. For data stream systems the stakes are even higher. Instead of one- shot queries in a relational DBMS, a stream system is processing *many continuous queries simultaneously.* These queries are active for long periods of time and they process massive streams in real time. A poor query implementation choice can negatively affect system performance for the lifetime of the query.

The key to achieving good stream processing performance is to optimize multiple queries together, rather than individually. In a stream query workload, it is often the case that multiple concurrently active queries can share state and computation. Query evaluation techniques that exploit this property are referred to as Multi- Query Optimization (MQO) techniques. The importance of MQO for stream processing is widely accepted and various stream MQO techniques have been proposed [10, 16, 12, 22, 14, 15, 7].

Unfortunately, existing MQO techniques apply only to very specific queries or workload properties. For example, predicate indexing [10, 16] is tailored for a set of selection operators that all read the same input stream. In addition, work on MQO techniques so far has happened in parallel for CQL-style stream engines [2, 5], referred to as *Relational Engines (RE)*, and event pattern detection engines [8, 21], referred to as *Event Engines (EE)*. The former use an operator model similar to relational databases, while the latter implement queries with automata. This has led to an unsatisfactory state of MQO, characterized by a confusing variety of individual techniques that apply to specific workloads or implementation models only. This prevents effective MQO for complex queries and leads to a situation where similar approaches might be re-invented by the different communities for REs and EEs.

To address these problems, we propose a Rule-based MQO framework, called *RUMOR*. It is inspired by the classical Query Graph Model (QGM) of RDBMSes [17], where query optimization techniques for *single* queries can be naturally modeled as transformation rules on query plans. RUMOR provides a modular and extensible framework, enabling new optimization techniques to be developed and incorporated incrementally into the system.

To support rule-based MQO, we have to extend the key abstractions that are used in a traditional RDBMS or stream system: physical operators, transformation rules, and streams. We introduce a small number of carefully designed abstractions that together create a powerful MQO framework. In fact, RUMOR incorporates all previously proposed MQO techniques for stream processing. In particular, it successfully incorporates MQO techniques from both relational stream engines and automata-based event processing engines. Hence an additional benefit of RUMOR is that it enables the unification of these diverse camps of stream processing systems. Experimental results for our prototype implementation indicate that we can efficiently process a large number of

CQL-style relational stream queries, event processing queries, as well as *hybrid queries* involving query features from both types of query workloads.

RUMOR lays the foundation for multi-query optimizers (MQ Optimizers) for data stream processing. It opens up opportunities for exciting future research on finding new rewrite rules and on extending the approach to cost-based MQ Optimizers, incorporating ideas from the classical dynamic programming approach to cost-based *single* query optimization in RDBM Sec [18].

Contributions and roadmap. Our contributions can be summarized as follows.

- We propose RUMOR, a rule-based MQO framework, which naturally extends the rule-based query optimization and query-plan-based processing model used by current RDBM- Ses and stream systems.
- We show how new and existing MQO techniques for relational stream engines and for event engines can be integrated into RUMOR. This is done by defining a small number of carefully designed abstractions.
- We demonstrate the efficacy of our approach by presenting experimental results using a prototype implementation of RUMOR.

RUMOR integrates MQO techniques for REs and EEs. For ease of exposition, we interleave the description of RUMOR and integration of MQO techniques for REs into RUMOR. We then describe the integration of MQO techniques for EEs .

Transformation Rules on m-ops

We now extend the traditional transformation rules, which operate on query plans composed of physical operators, to *multi- query transformation rules,* or *m-rules* for short. M-rules operate on query plans composed of m-ops. Similar to a traditional transformation rule, an m-rule consists of a pair of *condition* and *action* functions [17]. The condition function is a Boolean side-effect-free function on the query plan to identify opportunities for sharing. Once a sharing opportunity is identified among a set of operators in a query plan, the action function modifies the query plan by replacing that set of operators with a single m-op. We say the m-rule *maps* a set of m-ops to a single m-op, or it *merges* these operators.

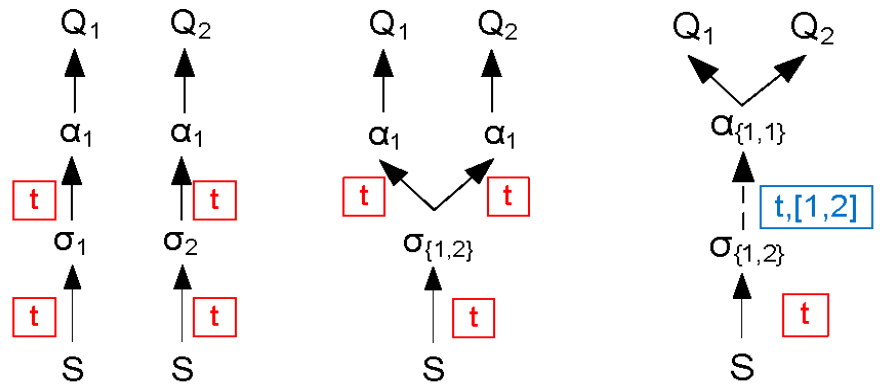

Figure 1:

Query Plans in RUMOR (Red Rectangles Represent Stream Tuples; the Blue Rectangle is a Channel Tuple)

More precisely, the condition of an m-rule is a function from the power set of the set of all possible m-ops to {true, false}. For a given set of m-ops, the rule can only be applied if the condition function evaluates to true.

The action of an m-rule is another function. This function maps a set of m-ops (for which the condition function evaluated to true) to a single m-op, referred to as the *target m-op*, which implements the input m-ops more efficiently with an MQO technique. In the query plan, we simply replace all edges that previously connected other operators with the to-be merged operators by edges to the corresponding input and output streams of the target m-op.

Expressing MQO Techniques with m-ops and m-rules

Most of the existing specialized MQO techniques share work among operators reading the *same stream(s).* These can be implemented in RUMOR through m-ops and m-rules. For example, we can model predicate indexing for equality predicates on a single attribute as an m-rule as follows. The condition of the m-rule evaluates to true only for a set of selection operators that all read the same stream and whose selection predicate is an equality predicate on the same attribute A.

The action rule then replaces them with a target m-op that uses a hash index on attribute A for a more efficient evaluation of the selection predicates of these operators.

It is not hard to see that all these previously proposed MQO approaches for multiple selection [10, 16], aggregation [22], and join [12] operators can be expressed similarly through corresponding m-ops and m-rules. The first three rows in Table 1 summarize these rules. Notice that in data stream processing systems, join and aggregation operators usually contain window specifications to prevent unbounded memory consumption. Also, aggregation operators may contain optional group-by specifications. For each operator type τ, we name the corresponding m-rule s_τ, indicating that it is an m-rule for instances of operator τ that all process the same input stream(s). The remaining rows in Table 1 will be discussed later in this paper. The current set of m-rules is not intended to be complete—the extensible nature of rule-based query optimization allows for adding new rules.

Extending Streams to Channels

Logically, a channel is equivalent to the union of a set of streams; the streams that are combined to form a channel are required to have compatible schemas. This can always be achieved by "padding" the schemas of individual streams with the attributes from other streams after appropriate attribute renaming.

Unlike a union of streams, a channel keeps track of which original stream a tuple belongs to. We say the channel *encodes* these streams. More formally, a channel encodes a set of data streams with union-compatible schemas as follows. The channel is defined as the union of its streams, but each stream tuple has an additional attribute called *membership component.* The membership component specifies the set of streams to which this tuple belongs. For efficiency, the membership component is implemented by a bit vector.

Through the use of a channel we can share work in two ways. First, when identical tuples from different streams are encoded as a single channel tuple, their *space* is shared. Second, when multiple streams are encoded into the same channel, the *computation* of their consumer operators may be shared.

Clearly, channels generalize streams. In RUMOR, they take the place of streams as the input and output of m-ops. For each m-op, the input (resp. output) channels together partition the set of input (resp. output) streams of this m-op. When an m-op *o* processes an input channel tuple t, a *decoding* and an *encoding* step are involved as follows. *o* first determines to which set of input

streams *t* belongs, so that it conceptually only evaluates those physical operators implemented by *o* that take this tuple as input. This is the decoding step. Similarly, when *o* is about to produce a set of output tuples, it needs to encode it into a set of channel tuples with the appropriate stream membership component, and then write them to the appropriate output channels. This is the encoding step.

Note that the decoding and encoding steps can often be implemented very efficiently, or might actually not be necessary at all. For example, consider an m-op $\pi_{\{1, \ldots, n\}}$ implementing *n* projections with the same projection specification, but with different input streams S_i through S_n. Suppose these *n* input streams are encoded by channel *C*, and the n output streams are encoded by channel *D*. In this case, for each input channel tuple *t* from C,$\pi_{\{1, \ldots, n\}}$ needs to perform projection only once and to produce only one output channel tuple in *D*, keeping the membership component of *t* intact in the output *D* tuple.

m-rule name	Set of input operators to which the m-rule is applicable	Target m-op
S_C	A set of selection operators which read the same stream	Predicate indexing [10, 16]
S_α	A set of aggregation operators which read the same stream with the same aggregate function but potentially different group by specifications	Shared aggregate evaluation [22]
$S_.$	A set of join operators which read the same two streams, with the same join predicate but potentially different window lengths	Shared join evaluation [12]
c_α	A set of aggregation operators reading sharable streams, with the same definition	Shared fragment aggregation [15]
$c_.$	A set of join operators which read sharable streams, with the same definition	Precision sharing join [14]

$s_i (or\ s_\mu)$	A set of; (or μ) operators reading the same two streams, with the same definition	Common Subexpression Elimination (Section 4.3)
$c_i (or\ c_\mu)$	A set of; (or μ) operators which a) have the same definition b) read sharable input streams for the first input stream parameter, where these input stream are produced by the same m-op c) read the same input stream for the second input stream parameter	Channel Based MQO (Section 4.4)

Table 1: Representative m-rules to Express Existing and New MQO Techniques

we can use a channel to encode the two output streams of $\sigma\{_1,2\}$ in Figure 1(b), resulting in the query plan shown in Figure 1(c). Here the dashed arrow represents the channel, and $\sigma\{_{1,1}\}$ represents the aggregation m-op, implemented by the shared fragment aggregation technique described in [15]. Suppose an input tuple t from stream S satisfies both predicates in $\sigma\{_{1>2}\}$. $\sigma\{_{1,2}\}$ then produces a single output channel tuple, represented by the blue rectangle in Figure 1(c). That channel tuple has the same content as the input tuple t, but is associated with a membership component denoted as [1,2], indicating that it belongs to both output streams of $\sigma\{_{1>2}\}$.

Note that ideas similar to channels were used for specific MQO algorithms for joins and aggregates in relational engines [14, 15]. Our contribution is to propose the addition of the channel concept to an MQO framework as a general abstraction for sharing work. As we will show the combination of m-ops, m-rules, and channels also leads to powerful new MQO techniques for event processing queries.

Conclusions:-

In this paper, we propose RUMOR , a rule-based MQO framework to express and evaluate query plans that share work among multiple stream quries. RUMOR integrates existing and new MQO technique for both REs and EEs. As a result , we are able to unify REs and EEs ,and efficiently process a large number of RE queries , and hybrid queries in a single engine.

REFERENCES

[1] *J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman. Efficient pattern matching over event streams. In Proc. SIGMOD, pages 147-160, 2008.*

[2] *Arasu, S. Babu, and J. Widom. The CQL continuous query language: Semantic foundations and query execution. Technical report, Stanford University, 2003.*

[3] *D. Carney, U. Qetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik. Monitoring streams — a new class of data management applications. In Proc. VLDB, 2002.*

[4] *S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite events for active databases: Semantics, contexts and detection. In Proc. VLDB, pages 606-617, 1994.*

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

[6] *J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for internet databases. In Proc. SIGMOD, pages 379-390, 2000.*

[7] *Demers, J. Gehrke, M. Hong, M. Riedewald, and W. White. Towards expressive publish/subscribe systems. In Proc. EDBT, 2006.*

[8] *Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, and W. White. Cayuga: A general purpose event monitoring system. In Proc. CIDR, 2007.*

[9] M. T. Edmead and P. Hinsberg. *Windows NT Performance Monitoring, Benchmarking and Tuning.* Pearson Education, 1998.

[10] *F. Fabret, H.-A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe. In Proc. SIGMOD, pages 115-126, 2001.*

[11] *N. H. Gehani, H. V. Jagadish, and O. Shmueli. Composite event specification in active databases: Model and implementation. In Proc. VLDB, pages 327-338, 1992.*

[12] *M. A. Hammad, M. J. Franklin, W. G. Aref, and A. K. Elmagarmid. Scheduling for shared window joins over data streams. In Proc. VLDB, pages 297-308, 2003.*

[13] *Q. Jiang, R. Adaikkalavan, and S. Chakravarthy. Towards an integrated model for event and stream processing. Technical Report CSE-2004-10, University of Texas at Arlington, 2004.*

[14] *S. Krishnamurthy, M. J. Franklin, J. M. Hellerstein, and G. Jacobson. The case for precision sharing. In Proc. VLDB, pages 972-986, 2004.*

[15] *S. Krishnamurthy, C. Wu, and M. Franklin. On-the-fly sharing for streamed aggregation. In Proc. SIGMOD, 2006.*

[16] *S. R. Madden, M. A. Shah, J. M. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In Proc. SIGMOD, 2002.*

[17] *H. Pirahesh, J. M. Hellerstein, and W. Hasan. Extensible/rule based query rewrite optimization in starburst. In Proc. SIGMOD, pages 39-48, 1992.*