Scheduling Continuous Queries in Data Stream Management Systems

Mohamed A. Sharaf[1] Alexandros Labrinidis[2] Panos K. Chrysanthis[2]
[1] ECE Department, University of Toronto
[2] CS Department, University of Pittsburgh
msharaf@eecg.toronto.edu {panos, labrinid}@cs.pitt.edu

ABSTRACT

Recently, several policies have been proposed for scheduling multiple Continuous Queries (CQs) in a Data Stream Management System (DSMS). The decision on which policy to use plays an important role in shaping the percieved online performance provided by the DSMS. In this tutorial, we provide an overview of different policies employed by current CQ schedulers and the performance goals optimized by these policies. Further, we discuss the salient properties of CQs conisdered by current policies as well as the efficent implementation of such policies into CQ schedulers. Finally, we present future research directions and open problems in CQ scheduling.

1. INTRODUCTION

The growing need for *monitoring applications* has caused a shift in data processing paradigm, moving from Database Management Systems (DBMSs) to Data Stream Management Systems (DSMSs). Traditional DBMSs employ a store-and-then-query data processing paradigm, where data is stored in the database and queries are submitted by the users to be answered in full, based on the current snapshot of the database. In contrast, in DSMSs, monitoring applications register Continuous Queries (CQs) which continuously process unbounded data streams looking for data items that represent events of interest to the end-user.

One main component in all DSMS is the CQ *Scheduler* which is the part of the system responsible for ordering the execution of registered CQs. Specifically, in a DSMS, the arrival of new data triggers the execution of multiple corresponding CQs, where the CQ scheduler is the component which decides the execution order of the operators of these CQs. As such, CQ scheduling is basically the resource allocation mechanism used to allocate the DSMS's processing power to different registered CQs with the objective of optimizing a certain performance goal. This is especially important when different CQs exhibit different characteristics and have different requirements. For instance, some CQs might be more important than others, or some CQs might be more costly to process than others. Hence, one of the main objectives in the design of a DSMS is the development of scheduling policies that leverage CQs characteristics to optimize the DSMS online performance.

PVLDB '08, August 23-28, 2008, Auckland, New Zealand

Challenges: Scheduling is a well know technique for optimizing the performance of computer systems (e.g., Web servers and O.S.). However, the scheduling problem in DSMSs is more challenging due to the salient characteristics of CQs which are significantly different from those of simple jobs in traditional computer systems. Among these features are complex query structures, dependency between queries, and the probabilistic nature of query processing.

To address the scheduling problem in DSMS, several research efforts have been devoted to different dimensions of the problem. These dimensions are as follows:

1. Performance goals: defining the DSMS performance goals and device policies for meeting these goals (e.g., minimize latency, or optimize memory usage). In addition, multi-objective scheduling tries to balance the trade-off between multiple performance goals.

2. Window-based CQs: scheduling of the more complicated window-based CQs like Joins and Aggregates. In such queries, buffers are used to store input data and the execution of a query involves processing this buffered data in addition to the newly arriving data.

3. Shared CQs: scheduling multiple optimized CQs with shared common subexpressions. This is another unique scheduling problem in DSMS where the CQ scheduler should exploit the sharing for further optimization.

4. Scheduler implementation: designing low-overhead CQ schedulers that are practical to use in such online system. This often involves a trade-off between the complexity of the scheduler and the quality of the scheduling decision.

Most of the above dimensions have been studied in isolation though they are interdependent by nature. The goal of this tutorial is to highlight the synergy between these different efforts. The expected outcome is a ready recipe of ingredients that are essential for designing efficient CQ schedulers in DSMSs.

Tutorial Overview: In this tutorial, we discuss all the previously mentioned aspects of CQ scheduling in a DSMS. The discussion partially builds on our own experience in this

area in building the NSF-funded *AQSIOS* DSMS prototype [1]. Specifically, we present the following: (1) Introduction to CQ processing in DSMSs, (2) DSMS performance goals, (3) Basic scheduling strategies, (4) Scheduling strategies for complex CQs, (5) Scheduler implementation issues, and (6) Future directions and open problems.

2. SCOPE OF TUTORIAL

In the following, we describe the main points addressed in this tutorial on scheduling CQs in a DSMS.

2.1 Continuous Query Plans

Scheduling is a well know technique for optimizing the performance of traditional computer systems. However, the scheduling problem in a DSMS is significantly different due to the special characteristics of CQs. These characteristics are easily illustrated by considering the following types of CQ plans:

1. Single-stream CQs,
2. Multi-stream CQs,
3. Multiple CQs with shared operators, and
4. Window-based CQ plans.

Each of the above CQ plans exhibit special features which are different from traditional jobs in an O.S. or Web server. This in turn makes the CQ scheduling problem more challenging. Moreover, CQs with different kinds of plans are expected to coexist at the same time at the DSMS. Hence, an efficient CQ scheduler should exploit the characteristics of those plans for further performance improvements.

2.2 Basic Scheduling Strategies

The MCQ scheduler decides the execution order of CQs, or more precisely, the execution order of operators within CQs. The execution order is decided with the objective of optimizing the DSMS performance under certain metrics. We classify performance metrics into traditional metrics and DSMS specific metrics. We also, orthogonally, classify scheduling policies into:

Single-Objective Scheduling: In a CQ, a tuple that fails some predicate is discarded, whereas a tuple that satisfies all predicates produces an event which is of interest to the user. This *probabilistic* nature of query processing has lead to a new family of *Rate-based* scheduling strategies for optimizing the traditional metrics in a DSMS, namely, *response time, throughput* and *slowdown*. In this tutorial, we explain the basic concepts underlying these policies as well as the specific variants used for:

1. optimizing average-case performance, and
2. balancing the trade-off between average- and worst-case performance.

While traditional metrics are suitable for measuring performance in any on-line system, in DSMS, more metrics are needed to measure performance from the data-perspective. Thus, we also discuss such metrics and scheduling policies for optimizing them. Our discussion includes scheduling policies for improving the QoD of data streams, when QoD is defined in terms of *freshness* or through application-specified graphs which define the utility of *stale* output. Finally, since a DSMS is basically an *in-memory* database system, we discuss scheduling policies that have been proposed to optimize the memory usage.

Multiple-Objective Scheduling: In DSMSs, and in computer systems in general, it is often desirable to optimize for multiple metrics at the same time. However, those metrics might be in conflict most of the time. Towards this, we present the current attempts for designing schedulers that are able to balance the trade-off between certain conflicting metrics (e.g., memory and response time).

2.3 Scheduling of Complex Queries

We will discuss in details the scheduling challenges due to two unique features of certain CQs, namely:

1. the presence of window-based operators, and
2. the inter-dependency between operators.

These features are manifested in several types of CQs where aggregate CQs use window-based operators, while optimized multiple CQ plans will exhibit operator inter-dependency, and finally join CQs will exhibit both features. In this tutorial, we will discuss several scheduling policies that handle these queries under different performance measures.

2.4 Scheduler Implementation

To ensure the applicability of scheduling policies in DSMSs, a low-overhead implementation is needed in order to reduce the amount of computation involved in computing priorities. In this tutorial, we make the distinction between:

1. Static CQ scheduling policies, and
2. Dynamic CQ scheduling policies.

For static policies, the schedule is computed only once which naturally leads to a low-overhead implementation. On the other hand, for dynamic policies, the schedule is a function in time which might render that class of policies very impractical. In this tutorial, we discuss several approximation methods for efficient implementation of dynamic policies to balance the trade-off between scheduling overhead and accuracy.

2.5 Future Directions and Open Problems

We discuss the requirements for a comprehensive framework of a CQ scheduler which accommodates different scheduling policies and considers all types of CQ operators and structures. That framework should also provide various knobs for balancing the trade-offs between different performance goals as well as for balancing the trade-offs between the scheduling complexity and accuracy. We also discuss the possibility of collaborative techniques that utilize the synergy between the CQ scheduler and other components in the DSMS, namely, the load shedder, query optimizer, and memory manager.

3. REFERENCES

[1] Mohamed A. Sharaf, Panos K. Chrysanthis, Alexandros Labrinidis, and Kirk Pruhs. Algorithms and metrics for processing multiple heterogeneous continuous queries. *ACM Trans. Database Syst.*, 33(1), 2008.