Nile: A Query Processing Engine for Data Streams

W. G. Aref, A. K. Elmagarmid, M. H. Ali, A. C. Catlin, M. G. Elfeky, M. Eltabak, T. Ghanem, R. Gwadera, M. A. Hammad, I. F. Ilyas, M. Lu, M. Marzouk, M. F. Mokbel, X. Xiong
Department of Computer Sciences
Purdue University
West Lafayette, In., USA

1. Introduction

This demonstration presents the design of "*STEAM*", Purdue Boiler Makers' stream database system that allows for the processing of continuous and snap-shot queries over data streams. Specifically, the demonstration will focus on the query processing part, "*Nile*". Nile extends the query processor engine of an object-relational database management system to support data streams. Our approach is motivated by the fact that industrial strength DBMS provides greater flexibility in defining and executing queries over static data. Furthermore, many emerging applications, particularly in pervasive computing, sensor-based environments, retail transactions, and video processing continuously report up-to-the-minute readings of sensor values, locations, status updates, etc. Therefore, extending DBMS functionality to support data streams could play a central role for these emerging applications.

We start our prototype implementation based on Predator [3], an object-relational DBMS. We added data stream as a special data type and implemented a stream-query interface through stream-scan and stream manager components. We use traditional SQL operators and consider window execution as an approach to restrict the size of stored state in operators such as join. We adopt the notion of long-running continuous queries that are also used by many emerging stream processing systems. When continuous queries are interested only on the recent portions (window) of the data streams, continuous queries are referred to as sliding window queries (SWQs). We focus on our demo on the pipeline execution of multiple SWQs over multiple data streams. We exploit sharing (query clustering) whenever possible to increase the scalability of our system.

The Nile stream query processing engine supports the following features:

- Scalability in terms of the number of queries and the number of data streams.
- Access control to accept/register new continuous queries and new streams.
- Providing guarantees for Quality of Service and Quality of Answers and how QoS is integrated into the Nile stream query processing engine.
- Online stream summary manager.
- Integrating online data mining tools in query processing over data streams.
- Approximate window join processing and joining in a network of data streams

We experimented our prototype query processing system using real data streams that represent retail transactions (from Wal*Mart department store) and medical video data streams.

2. Sliding Window Queries

Our current implementation of the Nile stream query processing engine supports sliding window queries over data streams. We illustrate such queries by the following example queries, where the input data stream is the retail transactions from multiple Wal*Mart stores. The schema of the transaction stream has the form (StoreId, ItemID, price, quantity, TimeStamp), where StoreId indicates the retail store, ItemID is the sold item identifier, price and quantity are information about the sold item, and TimeStamp is the time this transaction occurs. Query Q_1, continuously reports the distinct items that are commonly sold by two stores A and B within a two-hour interval as follows:

```
SELECT DISTINCT S1.ItemId
FROM SalesStream S1, SalesStream S2
WHERE S1.ItemID = S2.ItemID and
S1.StoreID = ``A" and S2.StoreID = ``B"
WINDOW 2 hours;
```

A second sliding window query, Q_2, that continuously reports the SUM of sales for each Store in the last hour is specified as follows:

```
SELECT S.StoreID, SUM(S.price)
FROM SalesStream S
GROUP BY S.StoreID
WINDOW 1 hour;
```

The WINDOW clause in the query syntax indicates that the user is interested in executing the queries over the sales transactions that occur during the time period beginning at a specified time in the past and ending at the current time.

3. Stream Query Processing

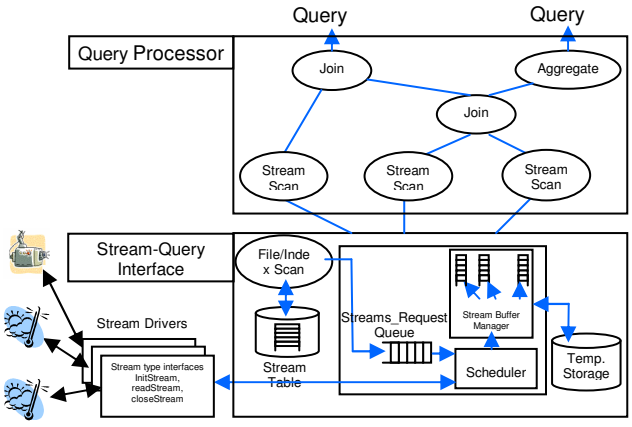

Figure 1: The Nile stream query processor

Figure 1 shows the main components of the Nile stream query processor. We describe in the following the stream-query interface, the window specification and the window-join algorithms.

3.1. Stream Query Interface

To enable query processing over data streams, we introduce a special data type, "StaremType", to define new input streams to the system. The StreamType is added to the set of supported types in Predator and can be used while specifying the schema in the "CREATE TABLE" command. Since the actual values of the data stream are collected at query execution time, the StreamType interface provides special "iterator-like" functions to communicate with input stream sources (e.g., remote locations over the network or sensor sources). Any StreamType must provide the following interfaces, InitStream, ReadStream, and CloseStream. In order to collect data from the streams and supply them to the query execution engine, we developed a *stream manager* as a new component of the stream database system. The main functionality of the stream manager is to register new stream-access requests, retrieve data from the registered streams into its local buffers, and supply data to be processed by the query execution engine. The stream manager runs as a separate thread and schedules the retrieval of tuples in a round robin fashion (other scheduling options are still being studied). To interface the query execution plan to the stream manager, we introduce a *StreamScan* operator to communicate with the stream manager and receive new tuples as they are collected by the stream manager.

3.2. Window Query Processing

We have developed two new algorithms to process window-join queries over multiple data streams [1]. The first algorithm which we call backward evaluation of window join (BEW-join) provides low output response time, whereas our second algorithm (which we refer to as forward evaluation of window join - FEW-join) improves output throughput. Both algorithms do not require synchronization between inputs from different data streams and therefore are applicable for streams where delays are likely to occur.

We also consider sharing between multiple SWQs with different windows and propose three approaches [2] to schedule the execution of the window join in this case.

We extend the query optimizer in Predator to integrate our window join operators while constructing the query execution plans. In addition, we modify the query optimizer and the run-time environment to consider sharing among new and currently executing queries. Furthermore, we implement each operator as a separate thread, which are scheduled by the system. The operators communicate with each other through a network of FIFO queues.

4. Description of the Demo

In this demonstration we will show the flexibility of our system to define new streaming sources and introduce continuous queries. We will present applications on real data sets that includes retail transactions from Wal*Mart stores and video data streams. We will also demonstrate the usage of Nile in supporting spatio-temporal data streams that represent the locations of moving objects. We will demonstrate how the window join operator can be integrated in a query plan to reconstruct in real-time the motion path of objects in a sensor network [1].

10. References

[1] Moustafa A. Hammad, Walid G. Aref and Ahmed K. Elmagarmid. Stream Window Join: Tracking Moving Objects in Sensor-Network Databases. *In Proc. of 15th International Conference on Scientific and Statistical Database Management, SSDBM 2003.*

[2] Moustafa A. Hammad, Michael J. Franklin, Walid G. Aref and Ahmed K. Elmagarmid. Scheduling for shared window joins over data streams. *In Proc. of the 29th International Conference on Very Large Data Bases, VLDB 2003*.

[3] Seshadri, P. Predator: A resource for database research. SIGMOD Record. 27(1). pp. 16-20. 1998.