Aurora: A Data Stream Management System

D. Abadi[†], D. Carney[§], U. Çetintemel[§], M. Cherniack[†], C. Convey[§], C. Erwin[§], E. Galvez[†], M. Hatoun[§], A. Maskey[†], A. Rasin[§], A. Singer[§], M. Stonebraker[±], N. Tatbul[§], Y. Xing[§], R. Yan[§], S. Zdonik[§]

[†] *Brandeis University* [§] *Brown University* [±] *M.I.T.*

1. INTRODUCTION

Streams are continuous data feeds generated by such sources as sensors, satellites, and stock feeds. Monitoring applications track data from numerous streams, filtering them for signs of abnormal activity, and processing them for purposes of filtering, aggregation, reduction, and correlation. *Aurora* [1,2,3] is a general-purpose data stream manager that is being designed and implemented (at Brandeis University, Brown University, and M.I.T.) to efficiently support a variety of real-time monitoring applications.

2. OVERVIEW OF AURORA

Aurora is being designed to deal with large numbers of asynchronous, push-based data streams. An Aurora processing network represents a set of continuous queries as a loop-free, directed graph of stream-oriented operators. The tuples get processed as they flow through the network and then delivered to the corresponding applications.

Aurora users build continuous queries out of a small set of well-defined operators that implement standard filtering, mapping, and windowed aggregate and join operations. The windowed operations have optional *timeout* and *slack* parameters that enable them to deal with slow and out-of-order arrivals, respectively.

Each Aurora application defines one or more *Quality of Service* (QoS) functions/graphs, each defining the utility of query results in terms of a performance or quality metric. Currently, QoS is captured by three graphs: (1) a *latency* graph, (2) a *value-based* graph, and (3) a *loss-tolerance* graph. The latency graph indicates how QoS drops as the results are delayed. The value-based graph defines the relative importance of the output values. The loss-tolerance graph is a simple way to describe how averse the application is to approximate or incomplete answers. The operational goal of the run-time system is to maximize the total QoS delivered to the applications.

The key components of the Aurora run-time system are the *scheduler*, the *storage manager*, and the *load shedder*. The scheduler decides which operators to execute and in which order to execute them. The scheduler pays special attention to reducing operator scheduling and invocation overheads. In particular, the scheduler batches (i.e., groups) multiple tuples and operators and executes each batch at once.

The storage manager is designed for storing ordered queues of tuples instead of sets of tuples (relations). It also combines the storage of push-based queues with pull-based access to historical data stored at connection points.

The load shedder is responsible for detecting and handling overload situations. The latter is accomplished by shedding tuples by temporarily adding "drop" operators to the Aurora processing network. The goal of a drop is to filter messages, either based on the value of the tuple or in a randomized fashion, in order to rectify the overload situation and provide better overall QoS at the expense of reduced answer quality.

Aurora has a GUI that allows the construction of arbitrary Aurora networks, specification of QoS graphs, stream-type inferencing, and zooming. Users construct an Aurora network by simply dragging and dropping operators from the operator palette and connecting them to each other, as well as to the input sources and output applications.

The current Aurora prototype is implemented on top of Debian GNU/Linux and consists of about 75K lines of C++ code augmented by another 50K lines of Java code that implements the GUI.

3. SYSTEM DEMONSTRATION

The system demonstration will include the illustration of query specification using the GUI, the Aurora performance/system monitoring tools that display in real-time the current state and the performance of the run-time system, and the execution of sample monitoring applications.

One application we will demonstrate is a tactical command-and-control application that was developed with the MITRE Corporation. The application is fed simulated data streams giving the locations of various classes of friendly and enemy units. Aurora's QoS specifications are exercised to show that in resource-limited times, the most critical information is delivered to the user.

We will also present an environmental monitoring application. Ongoing research uses live fish to detect the presence of toxins in water supplies. Sensors measure and report water quality and the fishes' breathing rates. In the demonstration, Aurora correlates sensor data from potentially noxious water, with baseline sensor data from clean water, to alert the user about potential contamination.

4. REFERENCES

[1] Carney, Cetintemel, Cherniak, Convey, Lee, Seidman, Stonebraker, Tatbul and Zdonik. Monitoring Streams – A New Class of Data Management Applications, Proceedings of Very Large Databases (VLDB), Hong Kong, August, 2002.

[2] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, S. Zdonik. Scalable Distributed Stream Processing. In proceedings of the First Biennial Conference on Innovative Database Systems (CIDR'03), Asilomar, CA, January 2003.

[3] The Aurora Project. `http://www.cs.brown.edu/research/aurora`

SIGMOD 2003, June 9-12, 2003, San Diego, CA.