POSTER ABSTRACT

Data Stream Management System for MavHome

Qingchun Jiang
Computer Science & Engineering
University of Texas at Arlington
jiang@cse.uta.edu

Sharma Chakravarthy
Computer Science & Engineering
University of Texas at Arlington
sharma@cse.uta.edu

ABSTRACT

MavHome project provides rich applications for addressing various issues associated with stream data processing. In this paper, we present our approach for building a data stream management system (DSMS) for the above smart home project. We further summarize our primitive solutions for continuous query processing, Quality of Service(QoS) management, and mapping a trigger mechanism to a stream processing system.

1. INTRODUCTION

The MavHome project at UT Arlington is a multi-disciplinary research project that focuses on the creation of an intelligent and versatile home environment. We view the smart home as an intelligent agent that perceives its environment through the use of sensors, and that can act upon the environment through the use of actuators. In order to learn and predict the inhabitants behavior through the use of sensors, we need to be able to process and analyze large amounts of steam data from various sensors in a timely manner.

The main issues that we are face in the design of a DSMS for a sensor environment such as Mavhome include: (*1*) provide efficient ways to process both continuous queries and one-time queries over unbounded streaming data using limited resources. (*2*) guarantee pre-defined QoS requirements although the system load sometimes exceeds the system capacity due to bursty inputs. (*3*) provide efficient and effective mechanisms to trigger actions associated with QoS constraints once pre-defined or abnormal events are detected. For more details about DSMS issues, refer to [1]. In the rest of the paper, we summarize our approach to designing a DSMS to address the above issues.

2. QUERY PROCESSING MODEL

Our query processing model illustrated in Figure 1 consists of a set of operator trees corresponding to query plans. The operator trees are merged (to facilitate computation sharing across multiple queries) to form a data-flow diagram. The basic job of our query processing system is to push each tuple from incoming data streams through a set of operators,

SAC'04, March 14–17, 2004, Nicosia, Cyprus.

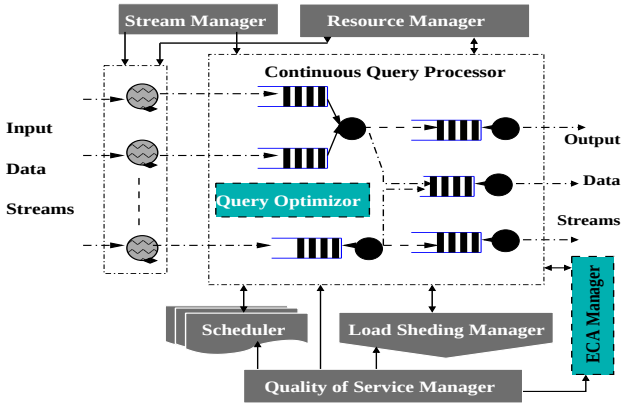

Figure 1: Architecture of the proposed DSMS

and ultimately to output streams that are consumed by applications. Furthermore, various mechanisms are employed to deliver QoS requirements of a query. Furthermore, an Event-condition-action (ECA) paradigm over data streams is employed to handle large number of triggers in the system.

The `continuous query engine` compiles a continuous query into a query plan and registers it with the scheduler. It is capable of processing select-project-join queries (window-based) and simple aggregate queries. The operators supported in the current system include `project, select, join, group-by`, and some aggregate operators, such as `max, min, average, count`, and special operators for stream data processing, such as `duplicate, split`, and `drop`. The `resource manager` is responsible for: allocating memory from its underlying operating system, allocating initial buffer for each operator, and for some routine work, such as recycling expired tuples in the system. The `stream manager` maintains catalog information for all streams, creates a thread for each incoming stream, and places the tuples from a stream into their corresponding input queues. It is also responsible for monitoring the input and stream characteristics of a data stream, such as sorted order of tuples, clustering, etc.

The `run-time scheduler` determines which operator, operator path, or query plan is to be executed at any time instant. It also takes the responsibility of delivering predefined QoS requirements of an active query by allocating appropriate resources. The `load shedding manager` determines when and where to shed load, and also how many tuples to shed by continuously predicting the system load. The `QoS manager` is designed to manage all aspects of QoS requirements obtained from applications. It is used to maintain all QoS contracts, and to guarantee the satisfaction

of QoS requirements by employing various means, such as scheduling, load shedding, and so on. The `ECA manager` is used to manage various ECA rules, to provide an effective means to continuously detect events, evaluate conditions, and to trigger per-defined actions.

3. QOS CONTROL AND MANAGEMENT

Quality of Service is one of the main challenges in our system and is critical to the success of the Mavhome project. Each application can specify its own QoS metrics such as tuple latency, inaccuracy, response time and others. Currently, we are exploiting various mechanisms, such as scheduling strategy and load shedding. We briefly discuss our results here. For additional details, refer to [3, 2]

Scheduling Strategy: It allocates more resources to those queries that have a more critical QoS requirements. However, it cannot decrease system load. Considering that most stream-based applications have a critical response time requirement, we use the `path capacity strategy` to achieve the best tuple latency [3].

Path Capacity Strategy: *At any time instant, consider all the operator paths that have input tuples waiting in their queues in the system, schedule a single time unit for the operator path with a maximal processing capacity to serve until its input queue is empty or there exists an operator path which has non-null input queue and a bigger processing capacity than the currently scheduled one. If there are multiple such paths, select the one which has the oldest tuple in its input queue.*

The `segment scheduling strategy` employs a similar strategy as the Path Capacity strategy except that it schedules the operator segment which has the biggest processing capacity instead of the operator path. Therefore it decreases the memory requirement of the stream query processing system. Currently, we are developing a QoS guaranteed scheduling strategy. However, scheduling strategies by themselves do not decrease system load. When system load exceeds system capacity, we have to resort to load shedding.

Load Shedding: It dynamically inserts a drop operator, which drops unprocessed tuples or partially processed tuples in a random manner or based on some semantics, to an existing query plan during heavy load periods and deletes the drop operator when system load decreases to a certain level. However, we have to decide when to start load shedding, where to shed, and how much to shed. The detailed solutions are presented in [2].

We define the `place weight` of a shedder as the ratio of load it saved to the error it introduced by dropping a tuple at a specified position along an operator path. The most effective position of a shedder along each operator path is the place where it has the biggest place weight. In order to determine when to start load shedding and how much to shed, we have to predict QoS metrics such as tuple latency of a query plan. We start load shedding as long as predicted QoS metrics are not consistent with pre-defined QoS requirements.

QoS metrics Prediction: To predict QoS metrics, we model each operator in the system as a queueing system, in which the functionality of the operator is modeled as service facility of a queueing system, and its input and output buffers are modeled as the input and output queues respectively. Then we analyze its tuple latency and memory requirements as a stand-alone system theoretically [5]. However, in a multiple query processing system, once the operator gains the processor, which is determined by a particular scheduling algorithm, if its input queue is empty, the operator goes for vacation immediately. Otherwise, the processor requires a setup time and then serves a certain number of tuples determined by a specific service discipline, i.e. gate service or exhaustive service, using a first-come-first-served order, and the service of a tuple is non-preemptive. After that, the processor becomes unavailable (to serve other operators or to handle other tasks) for a vacation time and then returns for further service. Therefore, an operator is modeled as a queueing system with vacation time and setup time. As a result, the whole query processing system is modeled as a network of such queueing systems.

In this model of queueing network, the queueing systems can be categorized into three classes based on their inputs: (*1*) external input(s) queueing system, which has only external input(s) from a continuous data stream. (*2*) internal input(s) queueing system. The arrival time of a tuple from an internal input is the departure time of the output of another operator. Therefore, this class only has inputs during its vacation period, and no input during its setup and service times. (*3*) external input and internal input queueing system. This class has both internal input and external input, and its inputs are a combination of the above two classes. By analyzing the queueing characteristics and input characteristics of each of the above queueing systems, we find a closed form solution to estimate the tuple latency and memory requirement. Given a scheduling strategy and the input rate of an input stream, we can predict the tuple latency of a query plan through monitoring input rate(s) of its input stream(s). Those predicted QoS metrics are further used to determine when to do load shedding, and how much to shed. For detailed information please refer to [4].

4. CONCLUSION

Currently, we have a prototype implementation, which is capable of processing various select-project-join window-based queries and simple aggregate operators. A run-time scheduler has been implemented to support the proposed scheduling strategies. The proposed load shedding techniques have been implemented as well. We are currently implementing load shedding and ECA rules. In this paper, we analyze the issues related to continuous query processing in a sensor environment such as MavHome, and present our solutions for these issues.

5. REFERENCES

[1] B. Babcock and et al. Models and issues in data stream systems. In *Proc. of the 2002 ACM PODS*, June 2002.

[2] Q. Jiang and S. Chakravarthy. Load shedding in a data stream management system. *http://itlab.uta.edu /sharma/Projects/MavHome/files/loadShedding.pdf.*

[3] Q. Jiang and S. Chakravarthy. Scheduling strategies in a data stream management system. *Technical Report CSE-2003-30, UT Arlington.*

[4] Q. Jiang and S. Chakravarthy. Analysis and validation of continuous queries over data streams. *Technical Report CSE-2003-7, UT Arlington*, July 2003.

[5] Q. Jiang and S. Chakravarthy. Queueing analysis of relational operators for continuous data streams. In *Proceedings of the 12th ACM CIKM*, Nov 2003.