Reducing Execution Overhead in a Data Stream Manager

Don Carney[†], Uğur Çetintemel[†], Alex Rasin[†], Stan Zdonik[†]
Mitch Cherniack[§], Mike Stonebraker[±]

[†]Department of Computer Science, Brown University
[§]Department of Computer Science, Brandeis University
[±]Laboratory for Computer Science & Department of EECS, Massachusetts Institute of Technology

1 Introduction

Applications that deal with potentially unbounded, continuous streams of data are becoming increasingly popular due to a confluence of advances in real-time, wide-area data dissemination technologies and the emergence of small-scale computing devices (such as GPSs and micro-sensors) that continually emit data obtained from their physical environment. Example applications include sensor networks, position tracking, fabrication line management, network management, and financial portfolio management. All these applications require timely processing of large volumes of continuous, potentially rapid and asynchronous data streams.

We have designed a system called Aurora [2], a data stream manager that addresses the performance and processing requirements of stream-based applications. Aurora supports multiple concurrent continuous queries, each of which produces results to one or more stream-based applications. Each continuous query consists of a directed acyclic graph of a well-defined set of operators (or *boxes* in Aurora terminology). Applications define their service expectations using Quality-of-Service (QoS) specifications, which guide Aurora's resource allocation decisions.

A key component of Aurora, or any data stream management architecture for that matter, is the operator scheduler that decides which operators to execute and how long to execute them. A naïve approach to scheduling would pick a tuple to run through an operator, schedule that operator, then loop back to pick another tuple. For operators with very low execution costs (e.g. filter or union), this tuple-at-a-time approach would experience serious performance problems because it does not take various execution and scheduling overheads into account. When operator costs are low, such overheads become dominant and scheduling without considering overhead leads to system degradation.

In this paper, we present a glimpse into our scheduling model which is designed to take advantage of our overhead reducing techniques. We also discuss the specific overheads encountered during initial development of our Aurora prototype. We have not found other stream management systems that take these overheads into account for scheduling.

2 Operator Execution Model

The traditional model for structuring database servers is *thread-based execution*, which is supported widely by traditional programming languages and environments. The basic approach is to assign a thread to each query or operator. The operating system (OS) is responsible for providing a virtual machine for each thread and overlapping computation and I/O by switching among the threads. The primary advantage of this model is that it is very easy to program, as the OS does most of the job. On the other hand, especially when the number of threads is large, the thread-based execution model incurs significant overhead due to cache misses, lock contention, and switching. More importantly for our purposes, the OS handles the scheduling and does not allow the overlaying software to have fine-grained control over resource management.

We suggest using a *state-based execution* model. In this model, there is a single scheduler thread that tracks system state and maintains the execution queue. The execution queue is shared among a small number of worker threads responsible for executing the queue entries, where each entry is a sequence of operators belonging to a (sub-)query. This state-based model avoids the mentioned limitations of the thread-based model, enabling fine-grained allocation of resources according to application-specific targets (such as QoS). Furthermore, this model also enables effective *batching* of operators and tuples, which we expect to have a significant effect on system performance as it cuts down the scheduling and box execution overheads.

An important challenge with the state-based model is that of designing an intelligent but low-overhead scheduler. In this model, the scheduler becomes solely responsible for keeping track of system context and deciding when and for how long to execute each operator. In order to meet application-specific QoS requirements, the scheduler should carefully multiplex the processing of multiple continuous queries. At the same time, the scheduler should try to minimize the system overheads, time not spent doing "useful work" (i.e., processing), with no or acceptable degradation in its effectiveness.

3 Reducing Overhead with Scheduling

We have built a prototype of Aurora based on the state-based model. Building the prototype rather than a simulator has enabled us to understand the actual system overheads involved in scheduling continuous queries (and their respective operators) for streaming data. Overheads fall into two categories: (1) the amount of processing overhead per tuple required to execute an operator (ideally, we want only to execute the operator, however, there is overhead associated with each operator execution including scheduling, box invocation, and queuing operations); (2) the number of times the stream management system goes to disk for each tuple processed. This second overhead is only relevant for memory-constrained systems requiring disk storage.

Our approach to overhead reduction takes advantage of *batching*. We use batching in two ways: The first approach, which we refer to as *train-scheduling*, schedules batches (or *trains*) of tuples within a single box execution. This reduces the total number of box executions required to process the tuples, thereby reducing overhead associated with scheduling, calls to box code, and context switches. Also, train-scheduling improves memory utilization by reducing the number of times tuples are moved between memory and disk.

The second approach to batching that we suggest is *superbox-scheduling* where a sequence of boxes is scheduled as a group. This can reduce overhead in two significant ways. First, scheduling overhead is reduced since multiple boxes are scheduled as a single unit. Second, memory utilization can be improved by allocating memory for the entire superbox at once rather than for each operator execution.

We suggest using a multi-level scheduling approach to address the execution of multiple simultaneous queries while also accommodating superboxes and trains. A first-level decision involves inter-query scheduling; i.e., determining *which* (sub-)query, or superbox, to process. This entails dynamically assigning priorities to queries, according to tuple train opportunities and QoS specifications. Sharing of operators (sub-queries) makes this decision much more interesting. A second-level scheduling decision involves intra-query scheduling (deciding how the selected query should be processed). This decision entails choosing the order in which tuple trains will be pushed through component operators. The outcome of these decisions is a sequence of operators, referred to as a *scheduling plan*, to be executed one after another. The scheduling plan is inserted into the execution queue to be later picked up and executed by one of the worker threads.

Our initial implementation of this two-level scheduling model has revealed significant improvements in overhead reduction. This is seen in terms of tolerance to increasing loads. We observed that using train-scheduling over a simple tuple-at-a-time approach yields significant performance improvements in terms of both memory usage and processing overhead. Also, pushing tuple-trains through superboxes allows our prototype to handle higher loads by further reducing scheduling overhead and improving upon memory utilization.

4 Ongoing work

An interesting tension exists between user expectations (latency-based QoS goals) and system throughput. Very high throughput can be achieved by allowing tuples to queue up at the inputs to a stream management system, however, this causes higher latencies which, in turn, lowers result utilities. This could create an interesting tradeoff between our batching techniques and user expectations. We plan to study our approaches in a QoS framework. The challenge is to find the correct degree of batching to meet QoS goals while achieving high throughput.

5 Related Work

We have begun to study the various scheduling techniques in more detail and provide initial results in [3]. We refer the reader to that paper for a detailed discussion of related work.

Of particular note is Eddies [1] tuple-at-a-time scheduling which provides extreme adaptability but has limited scalability. Also, work on continuous queries by Viglas and Naughton [5] discusses rate-based query optimization for streaming wide-area information sources in the context of NiagaraCQ. The primary scheduling goal of STREAM involves the minimization of the intermediate queue sizes [4]. None of these projects consider the overheads discussed here.

References

[1] R. Avnur and J. Hellerstein. Eddies: Continuously Adaptive Query Processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, Dallas, TX, 2000.

[2] D. Carney, U. Cetintemel, 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 *proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02)*, Hong Kong, China, 2002.

[3] D. Carney, U. Cetintemel, A. Rasin, S. Zdonik, M. Cherniack, M. Stonebraker. Operator Scheduling in a Data Stream Manager. In *proceedings of the 29th International Conference on Very Large Data Bases (VLDB'03)*, September 2003. To Appear.

[4] B. Babcock, S. Babu, M. Datar, and R. Motwani. Chain: Operator Scheduling for Memory Minimization in Data Stream Systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2003.

[5] S. Viglas and J. F. Naughton. Rate-Based Query Optimization for Streaming Information Sources. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Madison, Wisconsin, 2002.