Performance Evaluation of Rate-Based Join Window Sizing for Asynchronous Data Streams

Nithya Vijayakumar and Beth Plale
Computer Science Department
Indiana University, Bloomington
nvijayak@cs.indiana.edu, plale@cs.indiana.edu

Abstract

Our work is motivated by the large number of data stream sources that define mesoscale meteorology where asynchronous streams are commonplace. Techniques for performing filtering, aggregation, and transformation on multiple streams must be effective for the case of asynchronous streams. Rate Sizing algorithm (RS-Algo) links the number of events waiting to participate in a join to the rate of the streams responsible for their delivery. In this poster, we show the results of performance evaluation of the RS-Algo. The gains in memory utilization are largest under asynchronous streams.

1. Introduction

In grid computing data streams can be found in examples such as large scale scientific instruments that continuously generate data used by a community and in sensor networks. Experience we have gained with continuous query systems, particularly in the context of dQUOB [2], has made it clear that in scientific computing, the events constituting the data streams are often in the megabyte range in size, and that the rates of these streams can vary significantly relative to one another.

The dQUOB system model conceptualizes a data stream as a relational table over which a restricted set of SQL can operate. The data streams are implemented as channels in a publish subscribe system. Query execution is accomplished by means of inserting queries into a channel. This can be understood as inserting an intermediate node into a publish-subscribe graph $V = \{E, G\}$ where E are edges between nodes, and G are providers, consumers, or both. The query can serve such functions as filtering the stream, aggregating values over a stream, or combining streams.

In [1] we introduced an algorithm to improve memory utilization that takes advantage of the widely asynchronous stream rates observed in our motivating applications and uses the rate information to selectively adjust the size of the sliding windows used in database query joins. In this poster, we show the results of performance evaluation of the rate sizing algorithm.

2. Rate Sizing Algorithm

```
rate_sizing(window_interval) {
  for all i concurrently {
    sampleevent stream[i] for durationof window_interval;
    barrier();
    max_timestamp_interval = last_event[i].timestamp –
                             first_event[i].timestamp;
    join_window_size[i] = (event_received[i] * window_interval)
                             / max_timestamp_interval;
  }
  change_window_size[i]();
}
```

Figure 1. Pseudocode for RS-Algo dynamic rate sizing algorithm.

Sliding windows used for join processing can be sized either based on integer counts or a time interval. An integer count strategy would fix the join window size at say, 1000 events, then this size would then be applied to all join windows in the system. Our rate sizing algorithm (RS-Algo) maintains join windows at a size that can be expressed as a timestamp interval and adapts the sliding window size at runtime in response to detected changes in stream rate. By doing so, we could obtain two gains. First, a timestamp interval positions the user to better reason about the trade-off between performance and increased incident of false negatives. Second, we achieve window sizes that mirror differences in stream rates almost as a byproduct.

RS-Algo executes over the two streams participating in a particular join operation. The two input streams are sampled for an interval to determine respective stream rates.

0-7803-2175-4/04/$20.00 ©2004 IEEE

Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC'04)
1082-8907/04 $ 20.00 IEEE

The new join window size is calculated based on a time interval. Thus the join window sizes for "slow" streams are allowed to grow and shrink in response to real data rates.

3. Performance Evaluation

The purpose of the performance evaluation is to quantify the effect the algorithm has on memory utilization as compared to the default integer count based approach where a join window is of fixed size throughout execution.The experimental evaluation of the rate sizing algorithm as shown in Figure 1 is done in the context of the dQUOB system [2]. The query used in the experiment, shown in Figure 2, is a simple query consisting of one join operator, one project operator, and a user defined function (*i.e.,* Act). The query joins two streams and projects the results to a new stream that is transformed by applying a simple function on the result.

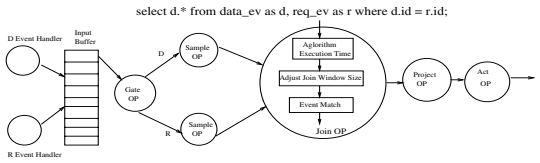

Figure 2. Q1: Simple query with one join, one select and one project operator.

The workload used to evaluate the rate sizing algorithm consists two typed event streams, that is, streams carrying events of a single type. Events are serviced by a depth-first traversal of the query tree, as shown in Figure 2. Each operator pushes the event onto the next operator in the path of traversal. Query execution is triggered by the arrival of either D or R events. D events carry virtual memory and CPU statistics of a client. To achieve a realistic workload on the quoblet, the D events are set to be 50 kilobytes in size. R events are small events of a few bytes carrying a user request.

4. Results

The join window size for streams D and R are plotted in the top two graphs of Figure 3. D events arrive at a rate of 10 events/sec and R alternates between 10 events/sec and 1 event/sec every 10 seconds. The rate sizing algorithm responds almost immediately to the change in stream rate. The algorithm overhead is in the order of a few microseconds and hence frequent triggering of the algorithm is possible.

The improvements in memory utilization can be seen by comparing the maximum join window size for D to the join window size for R in Figure 3. If an integer count were used, plotted lines for both D and R would remain at 100. Instead, R's plotted line changes between 10 and 1. This difference is the savings. We quantify the memory savings as a % measure to make it independent of the event size. For input rates used in our workload, we observed 57% savings in memory for the R stream. In another run, the integer count for input rates 50 events/sec (D) and 1 event/sec (R) was 500 and we obtained 97% savings in memory utilization for the slower R stream using RS-Algo.

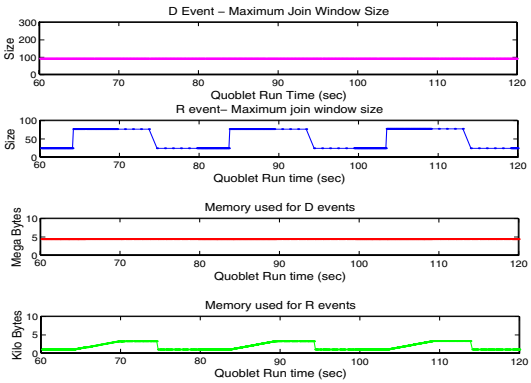

Figure 3. Join window size and memory utilization for streams with slow varying rates.

5. Conclusion

Under the workload used, we observed gains in memory utilization that support the viability of the algorithm. Our current work is focused on developing heuristics for the key parameters in the rate-based window sizing algorithm. These include *sample duration* - the time interval over which the stream is sampled to determine stream rate; *Sampling frequency* - the frequency at which re-sampling is triggered; and *Rate Sizing algorithm trigger* - the frequency with which the algorithm is triggered. More frequent triggering results in a more responsive join window size.

References

[1] B. Plale. Leveraging run time knowledge about event rates to improve memory utilization in wide area data stream filtering. In *Proc. 11th IEEE Intl. High Performance Distributed Computing*, 2002.

[2] B. Plale and K. Schwan. Dynamic querying of streaming data with the dQUOB system. *IEEE Transactions in Parallel and Distributed Systems*, 2003.

Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC'04)
1082-8907/04 $ 20.00 IEEE