Dealing with Disorder*

Peter A. Tucker, David Maier
{ptucker,maier}@cse.ogi.edu

OGI School of Science & Engineering at OHSU

Abstract

Often, queries over data streams require input that is ordered in some predetermined way. This order may simply be arrival order or given explicitly on specific attributes, such as timestamp or sequence numbers. For various reasons, data in a stream may not arrive in the expected order. We first introduce a new operator, called the *punctuate* operator, which is one way to embed punctuations into a data stream. We then discuss two ways for handling disorder in data streams, and compare them to how disorder can be addressed using punctuations embedded with the punctuate operator.

1 Introduction

Order can be important in processing data streams. The order of items in a stream may carry information, such as temporal sequence of events or measurements. Preserving such information may require query operators to maintain order of stream elements. Query specifications may depend on order, such as selecting all local maxima or windowing based on position of items in order (for example, sliding average over the last n items). Knowledge of stream order can benefit certain operators, such as aggregates, that might otherwise block or retain state until the end of the input.

Order can be implicit (for example, arrival sequence of items) or explicit (for example, via timestamps or sequence numbers). There may be multiple explicit orders of interest on a single stream [3]. Obviously, implicit order can be converted to explicit order by augmenting data items with an additional attribute. Converting explicit order to implicit order (that is, sorting on an ordering attribute) is not always so simple with data streams, nor is it always desirable.

Disorder arises for several reasons: Data items may take different routes, with different delays, from their source; the stream might be a combination of many sources with different delays; the ordering attribute of interest (event start time) may differ from the order in which items are produced (event end time).

As a running example we use an online auction system similar to EBay [2], shown in Figure 1. Bids are sent over the Internet to an online auction management system. We want to execute many kinds of queries over these input streams, for example: output bids for a particular item in the order they were posted (which may differ from the order in which they arrived), or determine the average price for bids on computer monitors posted over the past hour.

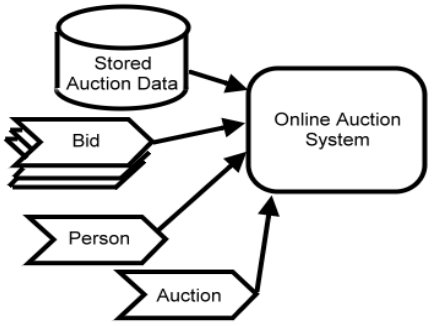

Figure 1: The Online Auction Monitoring System

There have been several approaches suggested to deal with disordered streams. A common approach is to sort data items into the order of interest as they arrive. That approach has several problems. One is that it introduces delay – a received item cannot be released for processing until it is known that all items earlier in the order have been received. Another is that sorting requires space for buffering out-of-order tuples. A third is that it may conflict with policies (such as quality of service) that want to give certain data items higher priority for processing. A second class of approaches leaves the stream disordered, but uses (or enforces) constraints on the amount of disorder so that data items may be processed as they arrive. One such approach is the use of *slack* in operators of the Aurora system [1], which defines how far displaced (in terms of number of data items) any data item can be from its correct position.

Our own approach to dealing with disorder is based

*Funding provided by DARPA through NAVY/SPAWAR Contract N66001-99-1-8908 and by NSF ITR award IIS0086002.

on the use of *punctuations* [5], which are extra items placed "in stream" to convey information about the progress of the stream. A punctuation p can be thought of as a predicate indicating that any data item t satisfying p has already been received. Punctuations can thus be used to indicate progress along an order of interest, even when the stream is disordered.

We discuss elsewhere [5] some of the ways punctuations can be generated in a stream. We describe a query operator that embeds punctuations into a data stream in Section 2. Then we discuss specific issues and how they can be addressed in Sections 3 and 4. We conclude in Section 5.

2 The Punctuate Operator

Punctuations may be embedded into data streams using the *timer* operator and the *punctuate* operator. The timer operator receives signals from the operating system at a fixed rate and outputs the current system time minus a given offset parameter, similar to a *heartbeat* [4]. The punctuate operator takes two data inputs: the timer operator output and another data stream whose items contain a datetime attribute. It outputs punctuations of the same structure as data items in the data stream, with *wildcard* values for the non-datetime attributes and with the timer value for the datetime attribute. These punctuations therefore state that no more data items with a datetime value less than the output from the timer will appear in the stream. Figure 2 shows a punctuation for the bid stream.

```
<bid>                      <PUNCT:bid>
  <time>5567</time>          <time>[,5567]</time>
  <person>p1</person>        <person>*</person>
  <price>10.00</price>       <price>*</price>
  <item>i1</item>            <item>*</item>
</bid>                     </PUNCT:bid>
```

Figure 2: Sample bid data and matching punctuation. The punctuation matches bids for all items where `time` $\leq$ 5567.

The punctuate operator also enforces its own punctuations. That is, it checks input data items to make sure they do not match previously generated punctuations. Any data items that do match are filtered out. This feature is particularly useful when we do not have any guarantees on the data streams. In the auction example, if posted bids were allowed to be delayed indefinitely, we would never be able to determine an auction's winner. We set a policy that all bids that arrive t minutes later than their posted time are invalid. The punctuate operator enforces this policy for us.

The punctuate operator outputs punctuations every time a value appears on the timer stream, so determining the timer rate is important. If punctuations occur frequently in the data stream, blocking operators can output results more often and stateful operators can reduce state more often. The tradeoff is that the punctuations take up bandwidth in the stream and require more CPU cycles. Understanding this tradeoff for different queries and query operators is an area we are currently investigating.

3 Querying Over "Nearly-ordered" Attributes

In the online auction system, bids should be processed in the order they are posted, not the order they arrive. We cannot assume that individual items arrive in order. However, in most cases we can assume items will arrive within some constant bounds of where they belong in the sort order. The authors of the Aurora project [1] take advantage of this property by defining a *slack* parameter for window operators. The slack parameter specifies the number of data items to read after data for a new window has arrived before closing the current window. The slack value is the bound on how "out of order" an input can be.

We use punctuations to address this issue. Suppose, for any data item in the stream, we knew that no data items would arrive that precede it in the sort order after s seconds. We set up a timer stream to regularly output timer values of the current system time minus s seconds. The punctuate operator receives these values and outputs punctuations stating that no more data items will appear in the stream with the datetime value less than the value from the timer. The advantage of this approach as compared to the slack approach is that other operators in the query can take advantage of the punctuations that are output.

In the online auction example, suppose we know that bids may be delayed up to five minutes. We can set up a timer operator to output every 60 seconds the system time minus five minutes. When each timer value arrives in the punctuate operator, a punctuation is generated in the same schema as data items in the bid stream, using the timer value in the `time` attribute and wildcard values for all other attributes.

4 Maintaining Order

Even if the input is sorted, we must still maintain the sort order throughout the query. For many operators, such as select, this requirement is trivial. However, other operators must take extra care to ensure their output is in order. For example, merge (the order-preserving version) must get data items from all inputs before outputting a result to ensure the output remains sorted. In a traditional DBMS, this behavior is common – it is simply the merge behavior in the sort-merge implementation of join. However, data streams complicate processing. We cannot guarantee that data items will be available from all input streams. One

stream input may be intermittent or down, causing merge to block. It cannot output a result until it has a data item from every input.

The Gigascope authors [3] address maintaining order in an order-preserving merge. Gigascope executes queries over network packet data, ordered by timestamp. Fragmented packets are split from the stream into a special operator that reconstructs fragments into complete packets. The complete packets are merged back into the original stream using an order-preserving merge. The stream of newly reconstructed packets is much slower than the original stream, often causing merge to block for a reconstructed packet. Tokens are inserted denoting a minimum timestamp into the reconstructed packet stream when it has no data items to process. Data items from the original stream with a timestamp less than the token are output.

We can use the punctuate operator to handle slow rate data streams as well. Packets from each stream are passed to a punctuate operator along with a timer stream. Even when one input has stopped, the timer forces the punctuate operator to emit punctuations. The punctuations are sent to merge, allowing it to emit data items that match the punctuations as well as the punctuations themselves. The data items can be output ordered by timestamp. As before, the advantage of using punctuations is that the punctuations emitted from merge can be used by other operators in the query (such as joins or aggregates). The query subplan for our approach is shown in Figure 3.

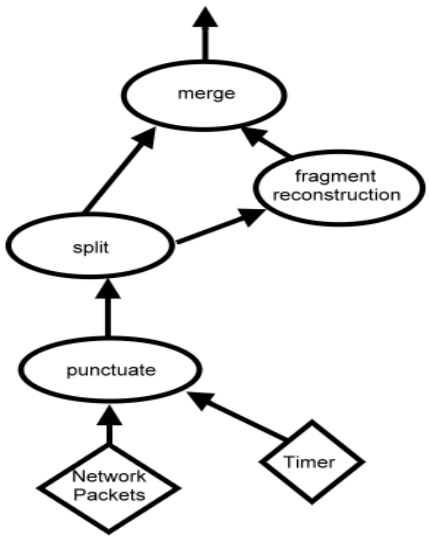

Figure 3: Query subplan to handle packet reconstruction with a punctuate operator.

We can take this example a step further. Embedding punctuations in the stream allows us to remove the order-preserving property of the merge, and simply output data items as they arrive. The punctuations embedded in the stream give us a bound on how disordered a stream can become. Operators further along in the query tree can use the punctuations to to process the data stream without having to rely on order (nor restore it). Data items can be emitted sooner, since they do not have to wait to get back in order.

5 Conclusions

We have discussed the challenges with disorder in data streams, and indicated why forcing a stream physically into a particular order may be undesirable. We have briefly presented our own approach, using the notion of punctuations and a new punctuate operator. A future challenge is dealing with queries where different operators within the query rely on different orders.

References

[1] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. B. Zdonik. Monitoring streams – a new class of data management applications. In *Proceedings of the 28th International Conference on Very Large Data Bases*, Aug. 2002.

[2] eBay home page. http://www.ebay.com/.

[3] T. Johnson, C. Cranor, O. Spatscheck, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *Proceedings of the ACM Special Interest Group on Management of Data (to appear)*, June 2003.

[4] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, resource management, and approximations in a data stream management system. In *Proceedings of the 2003 Conference on Innovative Data Systems Research*, pages 245–256, Jan. 2003.

[5] P. A. Tucker, D. Maier, T. Sheard, and L. Fegaras. Exploiting punctuation semantics in continuous data streams. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):555–568, May 2003.