Making Sense of Temporal Queries with Fine-Grained Provenance

Leilani Battle, Danyel Fisher, Mike Barnett, Badrish Chandramouli, Rob DeLine, Jonathan Goldstein

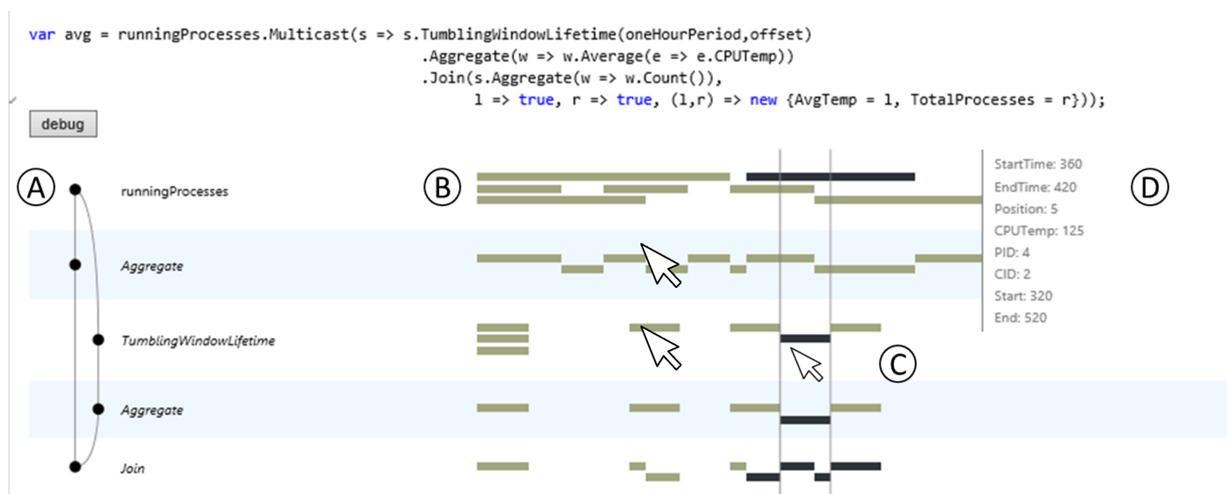


Figure 1: A complex temporal query, visualized. The query execution diagram (A) shows that only one side is running the *TumblingWindow* operation; the corresponding timeline bars (B) are windowed for only that side. The user is highlighting one bar (C); the upstream and downstream output bars in the join operation are also highlighted. (D) A tooltip shows information about the highlighted bar

Abstract

As real-time monitoring and analysis becomes increasingly popular, more researchers and developers are turning to data stream management systems (DSMS's) for fast, efficient ways to pose temporal questions over their datasets. However, these systems are inherently complex, and even individuals with database expertise find it difficult to understand the behavior of DSMS queries. To help analysts better understand their temporal DSMS queries, we developed a visualization tool that illustrates how a temporal query manipulates a given dataset, step-by-step. StreamTrace produces an interactive visualization that allows the user to audit their queries. We incorporated feedback from three expert DSMS users throughout our design process.

1 Introduction

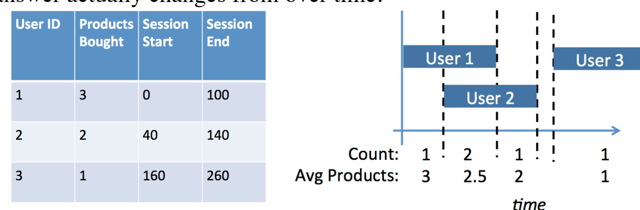
The recent rise in large-scale streaming data—from machine telemetry, to social streams, to scientific analysis—has led to an increasing demand for the skills and tools to allow data scientists to analyze data with a temporal component. *Data stream management systems* (DSMS) are a class of tools designed for continuous computation over temporal data *streams*; they are controlled by *stream query languages*. Stream query language queries are executed over DSMS's, just as SQL queries are executed by relational databases. The records stored in a data stream are referred to as *stream events*.

Consider a motivating example: a data scientist wants to answer two questions about a newly-opened online store:

- How many users are signed into our store?
- On average, how many products are they buying?

The online store generates log files like those seen in Figure 2.

Reading this as a relational table, a data scientist might assume that the number of users signed in is 3, and the average number of products purchased is 2. The timeline on the right side of the figure shows that this is not the case: records go in and out of scope over time as users carry out sessions at the store. These changes in the timeline result in multiple answers for our aggregates: the correct answer actually changes from over time.



User ID	Products Bought	Session Start	Session End
1	3	0	100
2	2	40	140
3	1	160	260

Figure 1: Table and timeline representation for an example data stream, with corresponding aggregate results

Tracking these changes is precisely what DSMS's are designed for—but that can also make DSMS queries hard to understand. Data scientists who are new to DSMS's—even if they are already experienced with relational databases—can find temporal queries that behave in ways that are difficult to understand. Before data scientists will readily incorporate DSMS's into their temporal analysis pipeline, they need the ability to more easily interpret what queries are doing.

To this end, we have developed StreamTrace, a visualization tool to help DSMS users better understand their temporal queries. Our visualizations were influenced by current strategies DSMS users employ while debugging their temporal queries.

For a given temporal query, StreamTrace allows DSMS users to see the series of transformations applied to each input data point to produce the corresponding query output. Enabling a history for any given element in a query, or the provenance of a query, makes it possible for users to better understand these transformations. We focus on displaying "fine-grained" provenance (as opposed to workflow provenance), where users can view the history of any individual event in a DSMS stream.

- *Leilani Battle is at MIT. E-mail: leibatt@mit.edu.*
- *The other authors are at Microsoft Research. E-mail: {danyelf, mbarnett, badrishc, rdeline, and jongold}@microsoft.com.*

2 Visualizing Fine-Grained Provenance

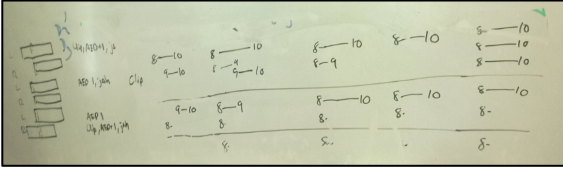


Figure 3: User-drawn timeline for a DSMS query

We first surveyed two expert DSMS users to get a sense of the strategies they currently employ to debug their temporal queries. We found that these users had trouble explaining how temporal queries work to their teammates new to DSMS's. We also retrieved sample queries and data from these experts. We found that these users build very small, hand-curated datasets to test their temporal queries.

From this initial feedback, we developed a prototype of StreamTrace. Figure 1 is a snapshot of StreamTrace. Our visualizations consist of a workflow execution graph (section (A) of Figure 1), and a series of timelines (section (B) of Figure 1).

Initially, we only drew a list of timelines, one per input or operator in the query. Each timeline represented the visualized result of executing the corresponding query operator at this point in the query's execution. However, early feedback from potential users showed us that timelines alone were difficult to navigate, and additional annotation was needed. We added workflow execution graphs, where each node in the graph is either an input to the query, or query operator. Edges in the graph represent the ordering and relationship between query operators.

We also incorporated interactions for exploring query provenance. When users hover over an individual event in one of the timelines, they see a tooltip (section (D) of Figure 1), showing the contents of the event at this point in the execution. We also highlight the current event and related events (section (C) of Figure 1). When events are highlighted *above* the current event, these represent inputs that contributed to the creation of the current event earlier in the query's execution. Similarly, when events are highlighted *below* the current event, these represent (possibly intermediate) outputs that the current event contributed to later in the query's execution. In Figure 1, the black bars show the current highlighting.

We again reached out to two expert DSMS users (including one from our previous survey) for feedback on our prototype. We were surprised to see that these users already relied on drawing flow charts, to track dependencies between query operators; and timelines, to track interactions between events. Figure 3 is a timeline diagram drawn by one of these users. The hand-drawn timeline diagrams validated our design choices to render events in timeline format. The flow charts confirmed that users interpret queries in multiple ways, validating the two parts to StreamTrace's display.

3 Provenance Tracking in the DSMS

We developed a new approach for recording the lineage of individual records as a temporal query is being executed in the DSMS. The key idea behind our approach is to have each event store its own list of the previous input events that contributed to it. To do this, each stream event is assigned a provenance identifier. As a query is executed, these identifiers are propagated through each query operator to the corresponding output events.

Our implementation consists of two steps: 1) modifying the structure of each event to include the list of past inputs; and 2) building wrappers around each query operator, which tell the operator how to propagate provenance identifiers from input events to the corresponding output events.

We implemented provenance tracking as a separate module outside of the DSMS, making our provenance techniques easily transferable to other systems.

Original Query

```
var result = Input1.Where(row => row.val > 10)
        .Join(Input2, (str1, str2) => new {str1.val, str2.name})
        .Select(row => row.val);
```

Intermediate Queries

```
var _dbg1 = Input1.Where(row => row.val > 10);

var _dbg2 = _dbg1.Join(Input2,
        (str1, str2) =>  new {str1.val, str2.name});
```

Provenance Graph

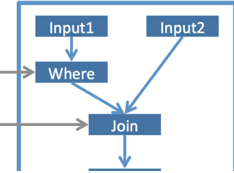


Figure 2: Query re-writing example

4 Query Re-writing

The DSMS is designed to return only the final output of queries. In the case of provenance tracking, we will only see for each final output record, which of the original input records contributed to this record. However, the user wants to see how the data is manipulated after every query operation is performed, not just the final output. To retrieve this metadata, we need to capture provenance information for all intermediate steps of the query.

An example of our query re-writing technique for LINQ [3] queries is provided in Figure 4. In this example, the user is filtering the Input1 stream for row values greater than 10, joining the filtered result with the Input2 stream, and modifying the joined result to include only the filtered values. We rewrite the user's original query (top left) to be a series of smaller queries (bottom left), where each new query represents an intermediate step in the original query's execution flow. We also inject a special provenance call into rewritten queries to trigger provenance tracking in the DSMS. The output of the new queries is used to build a provenance graph (right), where each node in the graph is the result of one of the new queries. The final provenance graphs are used to build our visualizations.

We rewrite queries automatically, so users can retrieve provenance information of a temporal query with one button click.

StreamTrace is implemented as a debugging feature in the Tempe system (previously Stat [3]). Tempe is a web-based application, providing users with a live programming environment in the browser. Tempe uses the Trill streaming engine [1] to execute temporal queries (written using LINQ **Error! Reference source not found.**).

5 Conclusions and Future Work

In this paper, we have presented StreamTrace, a new visualization tool to help DSMS users better understand their temporal queries. Our visualizations are influenced by diagrams frequently drawn by DSMS users when debugging their own queries by hand. StreamTrace allows users to explore the fine-grained provenance of a query, or the lineage of any record in the query.

We plan to conduct a study measuring the efficacy of our tool in helping LINQ users learn to write Trill queries.

Acknowledgments

This work was carried out while the first author was an intern at Microsoft Research. Our thanks to the DSMS experts who helped design our visualizations.

References

[1] B. Chandramouli, J. Goldstein, M. Barnett, R. DeLine, D. Fisher, J. C. Platt, J. F. Terwilliger, and J. Wernsing. *The Trill Incremental Analytics Engine*. no. MSR-TR-2014-54. 2014

[2] Meijer, E. The world according to LINQ. *Comm. ACM* 54, 10 (October 2011), 45-51.

[3] M. Barnett, B. Chandramouli, R. DeLine, S. M. Drucker, D. Fisher, J. Goldstein, P. Morrison, and J. C. Platt. *Stat!: an interactive analytics environment for big data*. SIGMOD 2013.