Towards Low Overhead Provenance Tracking in Near Real-Time Stream Filtering*

Nithya N. Vijayakumar and Beth Plale

Department of Computer Science, Indiana University
{nvijayak, plale}@cs.indiana.edu

Abstract. Data streams flowing from the physical environment are as unpredictable as the environment itself. Radars go down, long haul networks drop packets, and readings are corrupted on the wire. Yet the data driven scientific models and data mining algorithms do not necessarily account for the inaccuracies when assimilating the data. Low overhead provenance collection partially solves this problem. We propose a data model and collection model for near real time provenance collection. We define a system architecture for stream provenance tracking and motivate with a real-world application in meteorology forecasting.

1 Introduction

Dynamic data-driven scientific applications utilize data streaming in real-time from environmental sensors and instruments to effect simulation, modeling, and analysis that is more responsive to the physical domain (such as the atmosphere) and the computational environment. Responding in near real time to events in the environment, however, requires minimizing the latency between the occurrence of an event in the environment and the detection and processing of that event through a reduction and analysis pipeline. When decisions are being made in near real time to process incoming data and trigger the appropriate model or service, keeping records of the activities being applied is jettisoned in favor of keeping service time low. Avoiding the recording of historical data is not a viable solution because scientists need the ability to trace a result, such as a statistical result, back to the one or more events in streams that caused them. We need a low overhead model for provenance collection on streams.

The general approach to stream processing is to execute a set of tasks continuously on the incoming events. These tasks can be defined as database queries [1,2,3,11] or a pipeline of computational entities [4] that operate on the data events. We use the term *filters* to refer to the tasks executed on data streams. In this paper we focus on provenance tracking for stream filtering systems. Provenance collection in stream filtering systems pose the following challenges:

1. ***Identifying provenance entities*** - Provenance systems generally collect data about datasets [6]. In a stream filter system, events can be tiny, just a

* This work supported under NSF cooperative agreement ATM-0331480 and DOE DE-FG02-04ER25600.

L. Moreau and I. Foster (Eds.): IPAW 2006, LNCS 4145, pp. 46–54, 2006.

few kilobytes in size and can be generated at high rates. Collecting provenance on an event by event basis can overwhelm the system. The challenge is in identifying the correct "atomic unit" that is traceable by the stream filter system and strikes a reasonable balance between efficiency and meaningful provenance.

2. ***Capturing stream filtering conditions with low overhead*** - Filters executing on streams can be dynamically deployed and are subject to reconfiguring on the fly [3]. Data streams drop data periodically, such as when a feed from one of the 120 continental U.S. Doppler radars goes down briefly. Network congestion may cause delays and bursts in transfer of stream events. Transport over long haul networks can corrupt data as well. Filters accommodate these changes by changing modes (e.g., approximation mode under imbalanced load [2]). The challenge lies in dynamically and efficiently tracking provenance of filter execution in a distributed environment.
3. ***Maintaining relevance with non-persistent data*** - Filters are typically associated with a lifetime, i.e., they can be specified to run for a particular duration of time [11]. Yet, the products they produce can long outlive the filters that produced them. The challenge is to trace back the source of a derived event and the conditions of stream filtering long after the filtering task was completed.
4. ***Dynamic accuracy estimation*** - Provenance can be used to produce quality of service guarantees. In stream filtering systems, aggregation of the stream data enables the detection of global behavior that cannot be done on single streams. Approximations and accuracy changes in input streams may affect the accuracy of the derived streams [18]. It is challenging to collect provenance across multiple streams and thus deduce the accuracy of derived streams.

In this paper, we address these challenges by creating a data model and a collection model for representing and capturing stream-related provenance that targets the unique needs of stream-driven applications. We define an architecture for stream provenance tracking and show how the provenance collection system fits within the context of the Calder [17] grid-based stream processing system. We demonstrate the feasibility of provenance collection through an example of meteorology forecasting application [5]. We restrict our discussion to provenance collection over data streams that monitor the physical environment, such as the atmosphere, soil, ocean, etc. In future work we will examine the computer-induced streams that support fault tolerance and reliability in a dynamic data driven application.

The remainder of the paper is organized as follows. Section 2 motivates a new model for stream provenance tracking and discusses related work. Section 3 describes the data and collection models for stream provenance tracking. Section 4 discusses an implementation of this provenance model in the Calder [17] system and Section 5 demonstrates an application of this provenance service in meteorology forecasting. Section 6 summarizes the conclusions and outlines the future work.

2 Motivation and Related Work

Currently, provenance techniques do not target stream filtering systems and hence only partially satisfy its needs. In this section, we discuss the challenges identified in Section 1 and related work in this area. We have leveraged on the provenance techniques proposed by the provenance solutions [6,14,9,18], in creating a new provenance tracking model for streams. A recent survey [12] provides an overview of the existing provenance solutions for e-Science applications.

The first challenge is to identify the smallest unit for which provenance is collected in stream filtering systems. A data stream is an indefinite sequence of time ordered events, $(e_1, e_2, \ldots, e_n, e_{n+1})$ where $timestamp(e_n) < timestamp(e_{n+1})$. The virtual data schema used by the Virtual Data Grid project [6] represents the datasets, their relationships and the computations. In streams, a dataset corresponds to an event in a stream, so tracking provenance of datasets without burdening the system is a challenge. Provenance information can be encoded along with each dataset, but the events are not persistent and hence the provenance information is not available to the stream filter system after the event is processed. We would like to trace the source of events in derived streams to the events of inputs streams without identifying every event individually.

The second challenge is in capturing the provenance history of streams and filter states with low overhead on the system. Response time is crucial for stream filtering systems. Thus, low overhead (in time, resources etc) for provenance collection is of utmost importance. Log4j [8], which logs error and status messages to a log file, for instance creates a non-trivial load on the service about which it records data. Also aggregating the provenance traces into meaningful information is difficult with systems like Log4j. Capturing the provenance of filter execution is internal to the system and is expected to have a lower overhead compared to provenance collection for data streams.

The third challenge is to trace the source of a stream long after the filtering process has completed. We need to be able to clearly identify the environment in which a particular set of events (subset of a stream identified by timestamp) were generated. Also stream filtering systems adapt themselves to changes in underlying resources [2,11]. This involves changes in query execution plans and approximations when streams are not available. The provenance model needs to accommodate these changes and preserve the details for future reference.

The PASOA [14] project focuses on collecting provenance information on interaction between services in a workflow using a formalized protocol [7]. Karma provenance service [13] is used in the LEAD [5] project for tracking provenance of meteorological data and it's usage in web-services. In stream filtering systems, the provenance is collected for each stream and the filters that execute on the streams. The communication between the internal components of the stream filtering system is not as important as the entities as a whole. Security issues in a SOA-based provenance system is discussed in [16]. Security of provenance data is an important issue that is applicable to streams as well.

Finally, the provenance model needs to enable tracing the accuracy of a subset of the stream to a specific time period. Deducing an accuracy value for an

derived event based on the accuracy of the input streams and stream filtering environment is a challenge in itself. Trio [18], is a database system that augments conventional data model with accuracy and lineage and enables querying using understandable extensions to SQL. This is made possible by associating lineage and accuracy information with datasets in Trio. This is not applicable to streams, because the events are not persistent. We need to be able to trace accuracy of a subset of stream long after the stream was generated.

3 Provenance Tracking in Stream Systems

We propose the following data model and provenance collection model for stream filtering systems to address the challenges discussed in previous sections.

Data Model
We identify three atomic units of provenance collection in streams: *base streams*, *adaptive filters* and *derived streams*. *Base streams* are streams that are generated outside the stream filtering system. The generation source may be a instrument, experiment, or any process. *Adaptive filters* are declarative queries or application code that are associated with a life time and continuously execute on the data streams; *Derived streams* are streams that are produced by executing adaptive filters on base streams or other derived streams.

We propose a timestamp based *append only stack* approach for collecting provenance of streams and filters, and a *bottom-up provenance tree* to associate the base streams and derived streams. By append only stack we mean a data structure in which information can only be added not removed; and also that the latest information identified by the timestamp represents the current status. This provenance stack accommodates a set of information collected initially (base provenance information like the input streams, filter used etc) and a list of changes (dynamic provenance information like changes in stream rates, filter mode changes etc). The provenance stack can to be stored as a file or as a table in a database, in a persistent manner. Provenance information for a base stream constitutes the data format of the stream, its sources, information on the stream generation process, owner and permissions, user defined annotations and metadata. When users specify the filter to be executed, it could be appended with some annotation on its purpose. The execution plan and annotations serve as base provenance information for the filters. For a derived stream, the base provenance information is the list of input streams and the filters executed to derive the stream. The derived streams refer to the provenance of their input streams and that of their filters. Thus the lineage of a derived stream can be traced using a provenance tree where the input streams are at the root level and derived stream is at the leaf (bottom-up provenance tree).

System metadata (owner, permissions, etc) and user defined metadata, can be stored as name value pairs or using predefined schemas. The model supports annotations by storing them as inline text or storing them independently as in [9], and referring to them using URLs.

Low Overhead Provenance Collection Model
Our collection model is based on the assumption that each stream consists of a sequence of time ordered events and each event is associated with a timestamp. A subset of stream can be defined by a starting timestamp and an ending timestamp. Data streams are subject to rate and accuracy changes. To store the provenance of a stream, it is important to capture the changes in input streams and associate them to the set of events in the derived stream that are affected by the change. To facilitate this, the changes are logged with a starting timestamp. It is sufficient to match a change with the starting timestamp as it applies to the rest of the stream from that timestamp onwards. This model helps to capture the provenance information of a stream dynamically and keep the information up-to-date.

We need to be able to trace the source of a derived event and the stream filtering environment under which it was produced, long after the filtering task is completed. For this we only need to store the provenance history of all streams and filters in a persistent storage and not the events themselves. After the base provenance collection, information is logged only when something changes in the environment and hence the overhead for provenance collection and need for storage space are minimal. The provenance history is stored as one provenance stack per stream or filter. By looking through the provenance information of the derived stream, their input streams and the filters, one can trace the filter states and conditions under which a particular set of events were derived. The same methodology applies to deducing the accuracy of the derived stream events. Given, that a well-defined formula exists in a particular domain to calculate the accuracy of the derived stream from that of the input streams and the filter being used, the provenance model can support it.

Example - A sample provenance document for a derived stream is given in Figure 1. The stream under consideration is a derived stream (ID D0010) and uses two input streams (base stream with ID B0011 and derived stream with ID D0005). The stream was started at Feb-10-2006 at 13:00:00 hours and the steady rate was 50 events/sec recorded 15 minutes after stream was registered. From the change log, we can see that the stream B0011 was missing for about 10 minutes during which the filters changed mode to approximate the missing stream. The accuracy of the derived stream reduced to 85% during the approximation. From the next change log timestamped at 13:45:00 it is known that the B0011 stream came back up and hence the approximations were removed and the accuracy of the derived stream increased to 100%.

4 Calder Provenance Service

Calder [17], is a distributed stream processing system that supports a service interface for query processing. The Calder system, is an extension of the dQUOB [11] project, and is composed of two subparts: a set of data management services and a set of dynamically configurable query processing engines,

```
<derivedstream>
    <name>Temperature Feed</name>
    <uniqueID>D0010</uniqueID>
    <queryID>Q0099</queryID>
    <inputstreams>
        <streamID>B0011</streamID>
        <streamID>D0005</streamID>
    </inputstreams>
    <systemmetadata>
        <name> owner </name> <value> foo </value>
        <name> permissions </name> <value> open to everyone </value>
    </systemmetadata>
    <starttime> <timestamp> 13:00:00 Feb-10-2006 </timestamp></starttime>
    <changelog>
        <event>
            <timestamp> 13:15:00 Feb-10-2006 </timestamp>
            <rate> 50 events/sec </rate>
        </event>
        <event>
            <timestamp> 13:34:56 Feb-10-2006 </timestamp>
            <description> B0011 down</description>
            <approxmiation> Sampling </approximation><accuracy> 0.85 </accuracy>
        </event>
        <event>
            <timestamp> 13:45:00 Feb-10-2006 </timestamp>
            <description> B0011 up</description>
            <approximation> None </approximation><accuracy> 1.0 </accuracy>
        </event>
    </changelog>
</derivedstream>
```

Fig. 1. Sample Provenance Document for a Derived Stream

as shown in Figure 2. Filters are specified as database queries expressed using a subset of SQL. Calder uses an extended OGSA-DAI v 6 [10] grid data service interface to support a data stream resource. The provenance service of Calder implements the provenance models described in Section 3 and supports a service oriented interface for querying the provenance information. It uses a native XML database to store the provenance history of streams and filters. Users register the base streams and filter queries by invoking the provenance service. Registration of derived stream is made by the system when a new query is submitted. The derived streams can then be retrieved in a timely manner as streams or asynchronously as chunks from the rowset service (Figure 2). Once a stream/filter query is registered, users can append it's provenance stack with additional information like annotations and metadata.

We appended the Calder system with a monitoring service to facilitate dynamic collection of provenance information during stream processing. The provenance service interfaces with the monitoring service by an event-notification interface. Figure 2 shows the architecture of Calder with provenance and monitoring services. The provenance information propagates as shown in Figure 3. The query planner is responsible for executing the filtering query. It updates the monitoring service whenever the execution plan of a query changes. A query execution plan may change due to a set of streams going down or a processing node failure. The monitoring service also gets updated by the query processing engines when an event of interest occurs. Events of interest include rate changes, missing streams, approximations and accuracy changes. The flexible service interface of Calder enables a scalable framework. A single instantiation of the Calder system

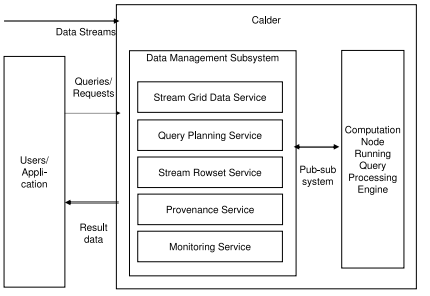

Fig. 2. Calder Architecture

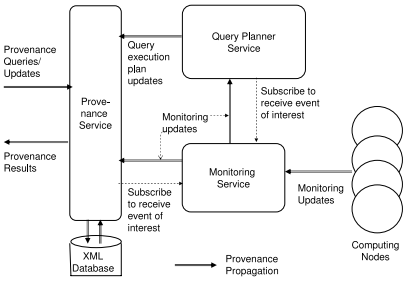

Fig. 3. Provenance Updates in Calder

can spawn multiple internal services and query processor engines on-demand to accommodate the load.

5 Application in LEAD

We discuss an application of Calder and it's provenance service in the context of the LEAD [5] meteorology forecasting project. *Time bounded stream mining* is the dynamic deployment of data mining agents that run for a bounded period of time, look for environmental events of interest, and report their results in the form of a trigger that can be used to invoke subsequent behavior. Our approach to on-demand data mining is to view the streams generated by heterogeneous instruments as belonging to a single data domain over which processing can be performed. The user interacts with the data domain through a declarative query.

The value to the user of time bounded stream mining can best be illustrated by means of an example. An atmospheric scientist is studying spring severe weather over the US Midwest. When a large storm front is moving in from across the Plains, she wishes to kick off a small, regional forecast simulation wherever storm cells emerge. She does this by sprinkling data mining agents in front of the storm line, each configured to run for a specified time (say 3 hours). Each mining agent executes a tight loop for the specified time looking at NEXRAD Level II Doppler scans within a small geospatial region for severe storm precursors. If the agent finds something of interest, it triggers a regional (small scale) forecast prediction simulation. When complete, the simulation invokes statistical analysis on the results. Figure 5 shows a sample continuous filtering query that can be executed on the incoming Level II data for the given scenario. Calder dynamically instantiates the query and query processing engine on a remote node, say on the TeraGrid [15]. The provenance service collects information on each of the data mining agents and streams used. It tracks changes in stream rates, temporary outages if any, and the conditions under which the operation was conducted. Such provenance information can be used to trace the result of a forecast model back to one or more events in streams that caused them.

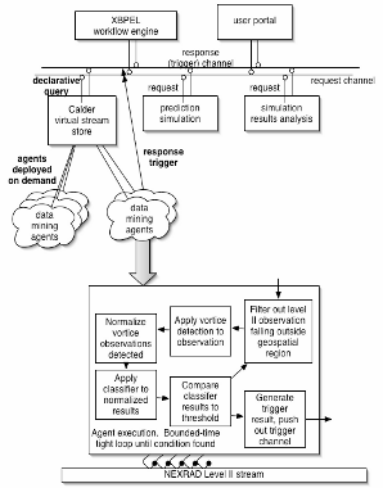

Fig. 4. Data Mining Agents in LEAD

```
RULE C:1
SELECT FROM level II as scan
WHERE
 (classif (normalize(
 MDA (scan in "bounding box")))
 > "threshold"
START 2006-04-25 12:00:00
EXPIRE 2006-04-25 20:00:00
THEN
 ACTION (threshold, scan)
```

Fig. 5. Query: Run the given data mining algorithm on incoming data streams and do the specified action

6 Conclusion

This paper discusses the unique challenges of low-overhead provenance collection in stream filtering applications. We introduce data and collection models addressing these challenges and discuss a prototype implemented as part of the Calder stream processing system. Our work is motivated through a real-world application of meteorology forecasting. Our current effort is focused on evaluating the performance of the system in the context of the dynamic weather forecasting. We are also working on extending the provenance collection service into a larger context management service that provides usage patterns, history of streams, user annotations and feedback on quality of streams.

References

1. Abadi, D. J. et. al.: The Design of the Borealis Stream Processing Engine. Conference on Innovative Data Systems Research (CIDR) (2005)
2. Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J.: Models and issues in data stream systems, ACM Symposium on Principles of Database Systems, (2002)
3. Chandrasekaran, S., et. al.: TelegraphCQ: continuous dataflow processing. International conference on Management of Data (SIGMOD) (2003).
4. Chen, L., Reddy, K., and Agrawal, G.: GATES: A Grid-Based Middleware for Processing Distributed Data Streams. IEEE International Symposium on High-Performance Distributed Computing,(2004)
5. Droegemeier, K., et al.: Service-oriented environments in research and education for dynamically interacting with mesoscale weather. IEEE Computing in Science and Engineering, (2005), 7(6)

6. Foster, I., Vockler, J., Wilde, M., and Zhao, Y.: The Virtual Data Grid: A new model and architecture for data-intensive collaboration. Conference on Innovative Data Systems Research, (2003).
7. Groth, P., Luck, M., and Moreau, L.: A protocol for recording provenance in service-oriented Grids. International Conference on Principles of Distributed Systems (2004).
8. Log4j. Apache Software Foundation. http://logging.apache.org/log4j/
9. Myers, J.D., Chappell, A., Elder M., Geist A., and Schwidder, J.: Re-Integrating the Research Record. IEEE Computing in Science and Engineering (2003) 5(3):44-50.
10. The OGSA-DAI Project. http://www.ogsadai.org.uk/
11. Plale, B., Schwan, K.: Dynamic querying of streaming data with the dQUOB system. IEEE Transactions on Parallel and Distributed Systems, 14(4):422–432, (2003).
12. Simmhan, Y. L., Plale, B., and Gannon, D.: A survey of data provenance in e-science. ACM SIGMOD Record, (2005) 34(3):31-36.
13. Simmhan, L. Yogesh, Plale, B., Gannon, D., and Marru, S.: Performance Evaluation of the Karma Provenance Framework for Scientific Workflows. Intl Provenance and Annotation Workshop (2006).
14. Szomszor, M., and Moreau, L.: Recording and reasoning over data provenance in web and grid services. Int. conference on ontologies, databases and applications of semantics (2003).
15. TeraGrid. http://www.teragrid.org.
16. Tan, V., Groth, P., Miles, S., Jiang, S., Munroe, S., Tsasakou, S., and Moreau, L: Security Issues in a SOA-based Provenance System. Intl Provenance and Annotation Workshop (2006).
17. Vijayakumar, N., Liu, Y., Plale, B.: Calder query grid service: Insights and experimental evaluation. To appear in CCGrid (2006).
18. Widom, J.: Trio: A system for integrated management of data, accuracy, and lineage. Conference on Innovative Data Systems Research (2005).