Reproducibility for streaming analysis

***Abstract*—The natural and physical sciences increasingly need streaming data processing for live data analysis and autonomous experimentation. Furthermore, data provenance and replicability are important to assure the veracity of scientific results. Here we describe a software system that combines high performance computing, streaming data processing, and automatic data provenance capturing to address this need. Data provenance and streaming data processing share a common data structure, the directed acyclic graph (DAG), which describes the order of each computational step. Data processing requires the DAG to specify what computations to run in what order, and the execution can be recreated from the graph, reproducing the analyzed data and capturing provenance. In our framework the description and ordering of the analysis steps (the pipeline) are separated from their execution (the streaming analysis) and the DAG created for the streaming data processing is captured during data analysis. Streaming data can have high throughputs and our system allows users to choose among multiple parallel processing backends, including Dask. To guarantee reproducibility, unique links to the incoming data, and their timestamps are captured alongside the DAG. Analyzed data, along with provenance metadata, are stored in a database, which can re-run analysis from raw data, enabling verification of results, exploring how parameters change outcomes, and data processing reuse. This system is running in production at the National Synchrotron Light Source-II (NSLS-II) x-ray powder diffraction beamlines.**

***Index Terms*—Streaming data processing, data provenance, directed acyclic graph**

I. Introduction

The acceleration of data acquisition rates for natural and physical sciences necessitates new approaches for processing data. This is especially important for live data visualization and automated high throughput experimentation, taking us towards autonomous operations where results of previous experiments feed back to make decisions about subsequent experiments. Often the data comes as a stream that can be high volume and throughput, requiring the computational load to be distributed and necessitating high performance computing. With such high data rates it is difficult to track every piece of analysis manually, especially for the live data processing needed for autonomous operations. Solving this problem also sets on the road to solving the reproducibility problem by storing the provenance of the analysis execution and results.

Streaming data processing is often considered separately from data provenance and reproducibility. Few workflow managers with strong provenance capture, like Sumatra [1] and VisTrails [2], have easy to use streaming systems as they are built for batch computation. However, streaming processing and provenance share a common representation, the directed acyclic graph (DAG). The DAG represents the execution of data processing steps on the streaming data. Each node in the DAG corresponds to a transformation of the data, conditional flow, or combination of data. The edges of the graph indicate the data flows from one step to the next. To properly define a streaming workflow one needs to define a DAG which encompasses how data are transformed and how they flow between transformation steps. This description is also needed to reproduce the exact execution sequence for analyzed data reproduction and to track data analysis for comparison.

Using this common representation we have built a system which combines the streaming data processing DAG with some additional information about the incoming data to automatically track the provenance of the data analysis. In this paper we will discuss our approach for streaming data processing and provenance. Additionally we will discuss how this system was put into production at National Synchrotron Light Source-II x-ray powder diffraction beamlines.

II. Approach

The design of our streaming provenance system focuses on three major parts: 1) automatic provenance capture, 2) separation of provenance tracking from data processing specification, and 3) separation of computation execution from the pipeline definition. Provenance capture can be difficult and tedious for users [1] so our system was designed to capture automatically both the transformed data and its provenance as it flows through the data analysis pipeline. Part of making the system easy to use is separating the specification of how to do the data processing from the provenance tracking. This enables users to build data processing pipelines independently from our system, and attach our system as needed, helping to preserve generality. By separating pipeline definition from the computation execution we enabled users to push their computation to any parallel processing system, including HPC systems.

To facilitate these goals the system's software is broken up into four parts: 1) Streamz and Streamz extension for streaming data processing with various backends for parallel computing, 2) the NSLS-II Data Acquisition, Management and Analysis (DAMA) group document/event model for structuring raw and analyzed data, 3) the NSLS-II DAMA group databroker for storing data, and 4) Streaming Heterogeneous Event Data (SHED) for translating between base-types and the event model data structure and for capturing provenance.

A. Streamz Streaming Framework

The Streamz and Streamz extension software is designed to specify and execute streaming data processing protocols. There are three major parts to each node: which upstream nodes the node receives data from, what the node does when it receives

the data, and which downstream nodes the transformed data is passed down to.

The frameworks also provide a layer of abstraction for parallelizing data processing. In this layer the graph nodes submit jobs to a backend allowing the backend to manage the data allocation and task execution. This allows advancements to be made in task scheduling and allocation without any changes to the pipeline. There is a dask [3] backend which uses the dask task scheduler and can scale to thousands of nodes on HPC systems via dask-jobqueue (https://dask-jobqueue.readthedocs.io/en/latest/). This model is not limited to dask, as there are backends in development for multi-threading and multi-processing support. These backends use a the Executor python class so any system with an Executor like object could be supported.

B. Data Structure and Management

Our system reuses the NSLS-II raw data model, also called the document or event model for analyzed data. This is a powerful data-structure for handling experimental data that is inherently heterogeneous and may consist of multiple distinct, but linked, asynchronous streams. For example, one stream could be triggering a detector whilst another, slower, stream could be changing sample temperature according to some temperature-time map. The document model is based around 4 document types: 1) start documents, which describe information known at the start of an experiment, simulation or analysis, 2) descriptor documents, which hold the metadata that defines the data in each distinct asynchronous stream, 3) event documents, which contain the data. Each stream is a sequence of events. 4) stop documents, which describe when and how the experiment, simulation or analysis ended. This data structure is backed by a database, the NSLS-II DAMA group databroker. During data acquisition data is automatically produced in event format and saved to the database.

Sharing a common data model is advantageous because: 1) users need to know only one system, 2) advancements made in one system can help the other 3) raw data is guaranteed to be backed by a database, allowing us to reference the data, and 4) raw data and analyzed data can be treated within the same framework. Our system stores a representation of the graph and references to the incoming data are stored in the start documents. The stop documents are used to store the order of incoming data, allowing for exact replication of the execution by reproducing the data ordering. This set of metadata enables the replay of the analysis and comparison of results with different processing parameters.

C. SHED

Putting all these pieces together is the Streaming Heterogeneous Event Data (SHED) software. The purpose of this software is to: 1) translate the incoming event model data into data which the data processing pipeline can understand 2) capture information about the data processing graph, the references to the incoming data and the incoming data ordering, 3) translate analyzed data into the event model for storage in a database, visualization, or further processing.

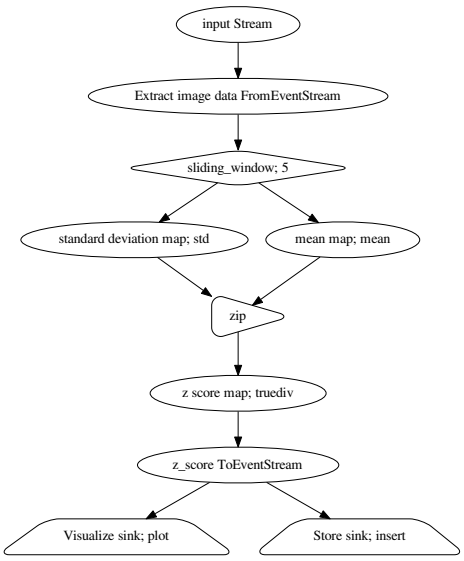

Fig. 1. An example pipeline which computes the Z-Score, a statistical measure, of the data with a sliding window of 5 images.

REFERENCES

[1] A. Davison, "Automated Capture of Experiment Context for Easier Reproducibility in Computational Research," Computing in Science Engineering, vol. 14, no. 4, pp. 4856, Jul. 2012.

[2] C. Scheidegger et al., "Tackling the Provenance Challenge one layer at a time," Concurrency and Computation: Practice and Experience, vol. 20, no. 5, pp. 473483, Apr. 2008.

[3] M. Rocklin, "Dask: Parallel computation with blocked algorithms and task scheduling," in Proceedings of the 14th Python in Science Conference, 2015.