StreamingHub: A Framework for FAIR Data Stream Processing

Yasith Jayawardana
yasith@cs.odu.edu
Department of Computer Science
Old Dominion University
Norfolk, VA

Andrew T. Duchowski
duchowski@clemson.edu
School of Computing
Clemson University
Clemson, SC

Sampath Jayarathna
sampath@cs.odu.edu
Department of Computer Science
Old Dominion University
Norfolk, VA

ABSTRACT

Reusable data and reproducible results are cornerstones of quality research. Ideally, all research assets should be reusable and reproducible. In practice, however, this aspect is easily overlooked. As a solution, we propose a framework to build FAIR-compliant data stream processing workflows that are both reusable and reproducible by design. We evaluate our framework on two real-time applications in the 1) *eye movement analysis* and 2) *weather analysis* domains. Results show that our framework generalizes across both domains, and facilitates building real-time applications with reproducible results.

KEYWORDS

Metadata, Data Stream Processing, Scientific Workflows

1 INTRODUCTION

In this era of data-intensive, collaborative research, ensuring that research assets are reusable and reproducible is equally as important as creating them. Having quality, FAIR-compliant [15] (i.e., *findability*, *accessibility*, *interoperability*, and *reusability*) metadata is thus of paramount importance. While advancements in data sharing technology have encouraged data reuse, creating reusable, reproducible research assets is a painstaking process that involves creating metadata that is unambiguous to a novice [12]. These circumstances create a causality where research assets are published with poor metadata, and end-users thereby struggle to reuse them. Hence, the process of generating metadata has potential for improvement.

Data-intensive research typically involve developing, executing, and validating a series of data manipulation and visualization steps (i.e., a workflow) that lead to a meaningful outcome. A scientific workflow (SWF) system is a form of a workflow management system built for this specific need. SWF systems allow complex data computations and parameter-driven simulations to be fine-tuned at a component-level, and to test out alternative setups with ease [5]. While a plethora of SWF systems exist [5, 11], SWF systems such as *KNIME*[1], *Kepler*[2], and *Node-RED*[3] allow users to build workflows through visual programming [2]. This appeals to non-programmers, such as scientists with no programming background, as it abstracts away the complexity of underlying programming [1]. Moreover, SWF systems such as *Node-RED* and *Flink*[4] are geared towards stream-processing, but supports batch-processing as a special case of stream-processing. In either case, the optimal execution strategy may differ across workloads as it depends on the nature of the data [6]. Having FAIR-compliant metadata on both data and workflows could be beneficial towards such optimization. **We hypothesize that a SWF system driven by FAIR-compliant metadata and visual programming, would simplify the design and execution of data stream processing workflows.**

As a solution, we propose StreamingHub: a framework to simplify data stream processing through reusable data, reproducible workflows, and visual programming. Our contribution is three-fold:

(1) We propose an extensible, FAIR-compliant metadata format to describe *data sources*, *data analytics*, and *data sets*.
(2) We propose two heuristics to identify bottlenecks and data-intensive operations in workflows.
(3) We demonstrate how we built self-describing workflows and data-driven visualizations on StreamingHub.

Using three datasets, we evaluate StreamingHub on real-time applications in the domains of 1) *eye movement analysis* and 2) *weather analysis*. Here, we describe our data source(s) using the proposed metadata format, build stream processing workflows that leverage this metadata, and observe their resource utilization using the proposed heuristics. Based on our observations, we discuss the utility of StreamingHub for data stream processing, and uncover important challenges that inspire future work.

2 STREAMINGHUB ARCHITECTURE

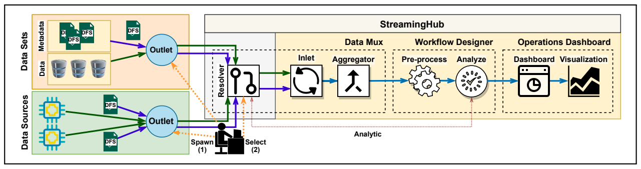


Figure 1: A High-level Overview of StreamingHub

StreamingHub consists of four components: 1) Dataset File System, 2) Data Mux, 3) Workflow Designer, and 4) Operations Dashboard (see Figure 1).

2.1 Dataset File System (DFS)

DFS[5] is a collection of schemas for describing data-sources, data-analytics, and data-sets [7, 8]. **Data-source** metadata provides attributes to describe data streams, such as frequency and channel count. **Data-analytic** metadata provides attributes to describe analytic data streams and their provenance (i.e., the hierarchy of transformations that lead to it). **Data-set** metadata provides attributes to describe the ownership, identification, provenance, and "spatial dimensions" of a dataset. Here, spatial dimensions indicate different perspectives that data could be viewed from. For instance, data obtained from multiple subjects engaging in different tasks

[1]https://www.knime.com/
[2]https://kepler-project.org/
[3]https://nodered.org/
[4]https://flink.apache.org/

[5]DFS Version 2.0 (https://github.com/nirdslab/streaminghub-dfs)

can be viewed "subject"-wise and "task"-wise. Table 1 provides a summary of the metadata attributes used in DFS.

Table 1: Summary of metadata attributes used in DFS

DATA SOURCE	
info	version, timestamp, and checksum (for identification)
device	model, manufacturer, and category of data source
fields	dtype, name, and description of all fields
streams	information about all streams generated
DATA ANALYTIC	
info	version, timestamp, and checksum (for identification)
sources	pointer(s) to the data source metadata
fields	pointer(s) to the fields used in analysis
inputs	pointer(s) to the streams used in analysis
streams	information about all streams generated via analysis
DATA SET	
info	version, timestamp, and checksum (for identification)
name	name of the data set
description	description of the data set
keywords	keywords describing the data set (for indexing)
authors	name, affiliation, and email of data set authors
sources	pointer(s) to the data source metadata
fields	pointer(s) to the fields used in analysis
groups	spatial dimensions of data in the dataset
resolver	path to an executable invoked to resolve data files

2.2 Data Mux

The data mux operates as a bridge between connected sensors, datasets, and data-streams. It uses DFS metadata to create the data-streams needed for a task, and provides three modes of execution:

- **Live** – stream live data from connected sensors
- **Replay** – replay a dataset as a data-stream
- **Simulate** – generate a simulated data-stream from a dataset

In *live* mode, it utilizes only *data-source* metadata. In *replay* and *simulate* modes, it utilizes all DFS metadata. The Data Mux uses LabStreamingLayer[6] wrapped within a WebSocket API to stream data into workflows and back.

Usage: The user initiates the data mux by spawning *outlets* on the local network, which, depending on the execution mode, may stream either live, replayed, or simulated data. Next, the user discovers data-streams on the network via the *resolver*, and selects a subset of them for analysis. Note that spawning of outlets and subscribing to outlets are kept independent to improve scalability. Next, the resolver spawns *inlets* for the selected data-streams, performs time-synchronization, and passes time-synchronized data into the *aggregator* to merge them together. When merging data-streams, the *aggregator* adheres to the data frequencies specified in the metadata, to facilitate temporally-consistent replay and simulation. The data mux then transitions into an awaiting state, and remains there until it receives a subscription from a *workflow*. Once a workflow subscribes, the data mux transitions into a streaming state, and remains there until that subscriber disconnects, or until the data stream itself is exhausted.

2.3 Workflow Designer

The workflow designer is the front-end for users to build scientific workflows. Workflows define which operations are performed on the data streamed from the data mux. A workflow comprises of *transformation nodes* and *visualization nodes* that are bound into a directed graph using *connectors*. *Transformation nodes* define the operations performed on input data, and the output(s) generated from them. Each transformation node may accept multiple data streams as input, and may generate multiple data streams as output. *Visualization nodes*, on the other hand, may accept multiple data streams as inputs, but instead of generating output streams, they generate visualizations. In this work, we primarily use Vega [14] to declare visualization nodes in JSON format. Additionally, a node itself can be defined as a workflow, allowing users to form hierarchies of workflows. This allows users to re-use existing workflows to form more complex workflows, which also serves as a form of abstraction (See Figure 2 for an example). We use Node-RED to implement the workflow designer, as it offers a visual programming front-end to build workflows, while allowing users to import/export workflows in JSON format. This facilitates both technical and non-technical users to design workflows and share them among peers.

2.4 Operations Dashboard

The operations dashboard allows users to monitor active workflows, generate interactive visualizations, and perform data-stream control actions. When designing a workflow, users may add visualization nodes at any desired point in the workflow. Each visualization node, in turn, would generate a dynamic, reactive visualization on the operations dashboard. In terms of data-stream control, we propose to include five data-stream control actions: start, stop, pause, resume, and seek. These actions would enable users to temporally navigate data streams and perform visual analytics [10]; a particularly useful method to analyze high-frequency, high-dimensional data. We use Node-RED to implement the operations dashboard (See Figure 4 for a sample dashboard).

3 FLUIDITY AND GROWTH FACTOR

Apart from metrics such as latency, throughput, scalability, and resource utilization, we propose two heuristics: 1) *fluidity* and 2) *growth factor*, to characterize the performance of a stream processing workflow. In this study, we use these heuristics to identify bottlenecks (via fluidity) and data-intensive operations (via growth factor) in our workflows.

Fluidity (F): We define F as the ratio between inbound and outbound data frequency of a transformation $T : S_i \to S_o$, where S_i is the set of inbound streams, and S_o is the set of outbound streams. Here, each $s_o \in S_o$ will have an expected outbound frequency f_e, and a mean observed outbound frequency f_μ. Formally,

$$F(s_o, t) = 1 - \sqrt{1 - (f_\mu / f_e)^2}$$

where $F(s_o, t) \in [0, 1]$, and $f_\mu \in [0, f_e]$. For any outbound stream s_o, f_e is a constant. However, f_μ is affected by runtime parameters such as hardware, concurrency, parallelism, and scheduling, and therefore varies with time. f_μ can be estimated by dividing the number of samples generated from T by the time taken; however, adding a Kalman filter may improve the estimation of f_μ. Ideally, $f_e = f_\mu$ which gives $F = 1$. However, F decreases exponentially with f_μ, and reaches 0 as $f_\mu \to 0$. Heuristically, slow-performing

[6]https://github.com/sccn/labstreaminglayer

transformations yield a low F, and can be improved through code-level optimization and runtime/scheduling optimizations.

Growth Factor (GF): We define GF as the ratio between inbound and outbound data volume of a transformation $T : S_i \longrightarrow S_o$, where S_i is the set of inbound streams, and S_o is the set of outbound streams. Formally,

$$GF(s_i, s_o) = \Big(\sum_{s \in S_o} V(s) \Big) / \Big(\sum_{s \in S_i} V(s) \Big) \ , \ V(s) = f_s \sum_{c_i} w_{c_i}$$

where $GF(s_i, s_o) \in [0, \infty)$, and $V(s) \geq 0$. For any stream s, the data volume $V(s)$ is calculated using its frequency f_s, and the "word size" w_{c_i} of each channel c_i in s. Here, the word size w_{c_i} represents the number of bits occupied by a sample of data from channel c_i (e.g., a c_i of type *int32* has $w_{c_i} = 32$). Furthermore, $GF < 1$ indicates a data compression, $GF > 1$ indicates a data expansion, and $GF = 1$ indicates no change. Heuristically, transformations with $GF < 1$ are likely candidates for caching and transmitting over networks, as they generate less data than they receive.

4 EVALUATION

Next, we evaluate StreamingHub through two case studies in the domains of *eye movement analysis* [3, 9] and *weather analysis* [4]. We select these particular domains to find whether StreamingHub generalizes across applications serving different scientific purposes. In both case studies, we evaluate three aspects of StreamingHub.

- Can we use DFS metadata to replay stored datasets?
- Can we build workflows for domain-specific analysis tasks?
- Can we create domain-specific data visualizations?

4.1 Case Study 1: Eye Movement Analysis

4.1.1 Data Preparation. We use two datasets, *ADHD-SIN* [9] and *N-BACK* [3], each providing gaze and pupillary measures of subjects during continuous performance tasks. We first pre-processed these datasets to provide normalized gaze positions (x,y) and pupil diameter (d) over time (t). Any missing values were filled via linear interpolation, backward-fill, and forward-fill, in order.

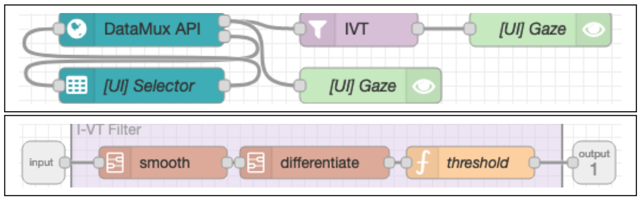


Figure 2: Eye movement analysis SWF on Node-RED

4.1.2 Experiment Design. In this experiment, we perform three tasks: 1) *replay* eye movement data, 2) obtain eye movement *analytics* in real-time, and 3) observe data/analytics through eye movement *visualizations*.

Task 1: Replay: We first generate DFS metadata for the N-BACK and ADHD-SIN datasets. Then we implement a *resolver* (using Python) for each dataset, which maps queries into respective data files. Next, we use the DataMux API to accesses the file locations returned by each *resolver* and instantiate *outlets* for streaming.

Task 2: Analytics: We create an eye movement analysis workflow on Node-RED, using visual programming. We begin by creating empty sub-flows for each transformation, and wiring them together, as shown in Figure 2. Next, we implement these sub-flows to form the complete workflow. For instance, we implement the I-VT algorithm [13] in the IVT sub-flow to classify data points as fixations or saccades.

Task 3: Visualization: We implement an interactive 2D gaze plot using the Vega JSON specification [14]. It visualizes gaze points as a scatter plot, connects consecutive gaze points using lines, and overlays a heat map to highlight the distribution of gaze points across the 2D space. It also provides a *seek* bar to explore gaze data at different points in time. The resulting operations dashboard shows the streams available for selection, the selected streams, and a real-time visualization of data.

4.2 Case Study 2: Weather Analysis

4.2.1 Data Preparation. We use a weather dataset from PredictionGames [4], providing daily min/mean/max statistics of metrics such as temperature, dew point, humidity, and wind speed between 1950-01-01 and 2013-12-31 in 49 US-cities. We first pre-processed this dataset by splitting weather data into seperate files by their city, and ordering records by their date.

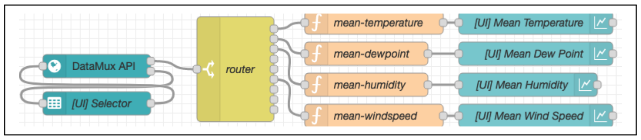
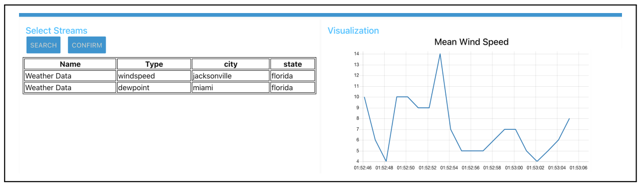


Figure 3: Weather Analysis SWF on Node-RED

4.2.2 Experiment Design. In this experiment, we perform three tasks: 1) *replay* weather data from different cities, 2) calculate moving average *analytics* in real-time, and 3) observe data/analytics through chart-based *visualizations*.

Task 1: Replay: We first create DFS metadata for the weather dataset. Here, we set the replay frequency as 1 Hz to speed up analysis (i.e., 1 day = 1 second). Then we implement a *resolver* (using Python) for this dataset, which maps queries into respective data files. Next, we use the DataMux API to access the file locations returned by the *resolver* and instantiate

Task 2: Analytics: We create a workflow to perform moving average smoothing on weather data and visualize each metric as shown in Figure 3. For the scope of this experiment, we only use the temperature, dew point, humidity, and wind speed metrics. *outlets* for streaming.



Name	Type	city	state
Weather Data	windspeed	jacksonville	florida
Weather Data	dewpoint	miami	florida

Figure 4: Operations Dashboard for Weather Analysis

Task 3: Visualization: Here, we accumulate and visualize the averaged weather data through interactive charts. Figure 4 shows a snapshot of the operations dashboard while visualizing replayed weather data. It shows the streams available for selection (left), the selected streams (rows), and a real-time visualization of data (right).

4.3 Heuristics

Here, we evaluate the consistency of our heuristics with two performance metrics: 1) latency and 2) data size. We pick four transformations from the case studies, and rank them in their increasing order of compute demand and outbound data size. Next, we send 100,000 samples into each transformation at 50 Hz, and compare our heuristics (fluidity F, growth rate GF) with the performance metrics (mean latency $\bar{t}$, inbound data size V_i, outbound data size V_o) (see Table 2).

Table 2: Metrics ($\bar{t}$, V_i, V_o) vs Heuristics (F, GF) of transformations (T) ranked by compute demand (R_c) and outbound data size (R_d)

T	R_c	R_d	$\bar{t}$ (ms)	V_i	V_o	F	GF
Mean	1	1	0.018	782 MB	15.64 MB	1.000	0.02
Threshold	2	2	0.097	782 MB	322.9 MB	0.999	**1.00**
Differentiate	3	4	0.243	782 MB	1.54 GB	0.934	2.00
Smooth	4	3	0.645	782 MB	781.9 MB	0.742	1.00

Here, $\bar{t}$ was higher for $F < 1$ than for $F = 1$. Moreover, when $V_o \ll V_i$, $GF < 1$ and when $V_o \gg V_i$, $GF > 1$, with an exception in the **threshold** transformation. In explanation, the **threshold** transformation filters data below a certain threshold, and the outbound data size is thus data-dependent. Since GF does not capture this behavior, it may not be suited for operations such as these.

5 DISCUSSION

Since F is based on observed frequency, it is not impacted by latency unless throughput is affected. Furthermore, F varies with time, but GF is independent of time. Thus, GF can be used to pre-optimize the workflow execution on constrained resources. Moreover, both F and GF can be applied at transformation-level to determine which outputs to cache or re-generate. However, a comprehensive evaluation is needed to validate their behavior across different applications.

StreamingHub currently relies on LabStreamingLayer for time synchronization. In the future, we plan to implement a time synchronization sub-flow in Node-RED, which would allow us to experiment with alternative communication methods like *MQTT*[7] and serialization methods like *ProtoBuf*[8]. Furthermore, we plan to conduct a user study to improve our design and identify new features. We also plan to support distributed workflow execution in StreamingHub by integrating workflow engines like *Flink*[9] or *Kinesis*[10]. By doing so, we plan to compare StreamingHub with existing SWF systems.

6 CONCLUSION

Using two eye-tracking datasets and a weather dataset, we demonstrated how metadata could be used to a) replay datasets as data streams, b) pass data streams into stream processing workflows, and c) perform data stream control and visualization for exploratory data analysis. We built two stream processing workflows using DFS, LabStreamingLayer, WebSockets, and Node-RED for two case studies: 1) eye movement analysis and 2) weather analysis. In the eye movement analysis case study, we developed a workflow to identify fixations and visualize gaze data. In the weather analysis case study, we developed a workflow to visualize weather-related statistics, and replay data at different frequencies than recorded. Moreover, we showed that the proposed heuristics can be used to identify bottlenecks and data-intensive operations in workflows. To promote reuse, this work is publicly available on GitHub[11].

[7]https://mqtt.org/
[8]https://developers.google.com/protocol-buffers
[9]https://flink.apache.org/
[10]https://aws.amazon.com/kinesis/

ACKNOWLEDGEMENTS

This work was supported in part by NSF CAREER IIS-2045523. Any opinions, findings and conclusions or recommendations expressed in this material are the author(s) and do not necessarily reflect those of the sponsors.

REFERENCES

[1] G.H. Brimhall and A. Vanegas. 2001. Removing Science Workflow Barriers to Adoption of Digital Geologic Mapping by Using the GeoMapper Universal Program and Visual User Interface. In *Digital Mapping Techniques*. U.S. Geological Survey Open-File Report 01-223, Tuscaloosa, AL, USA, 103–115.

[2] Margaret M Burnett and David W McIntyre. 1995. Visual programming. *COMPUTER-LOS ALAMITOS-* 28 (1995), 14–14.

[3] Andrew T Duchowski, Krzysztof Krejtz, Nina A Gehrer, Tanya Bafna, and Per Bækgaard. 2020. The Low/High Index of Pupillary Activity. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–12.

[4] Gabriel Dzodom, Akshay Kulkarni, Catherine C Marshall, and Frank M Shipman. 2020. Keeping People Playing: The Effects of Domain News Presentation on Player Engagement in Educational Prediction Games. In *Proceedings of the 31st ACM Conference on Hypertext and Social Media*. 47–52.

[5] Carole Goble, Sarah Cohen-Boulakia, Stian Soiland-Reyes, Daniel Garijo, Yolanda Gil, Michael R Crusoe, Kristian Peters, and Daniel Schober. 2020. FAIR computational workflows. *Data Intelligence* 2, 1-2 (2020), 108–121.

[6] Haruna Isah, Tariq Abughofa, Sazia Mahfuz, Dharmitha Ajerla, Farhana Zulkernine, and Shahzad Khan. 2019. A survey of distributed data stream processing frameworks. *IEEE Access* 7 (2019), 154300–154316.

[7] Yasith Jayawardana and Sampath Jayarathna. 2019. DFS: a dataset file system for data discovering users. In *2019 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*. IEEE, 355–356.

[8] Yasith Jayawardana and Sampath Jayarathna. 2020. Streaming Analytics and Workflow Automation for DFS. In *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries in 2020*. 513–514.

[9] Gavindya Jayawardena, Anne Michalek, Andrew Duchowski, and Sampath Jayarathna. 2020. Pilot study of audiovisual speech-in-noise (sin) performance of young adults with adhd. In *ACM Symposium on Eye Tracking Research and Applications*. 1–5.

[10] Daniel Keim, Gennady Andrienko, Jean-Daniel Fekete, Carsten Görg, Jörn Kohlhammer, and Guy Melançon. 2008. Visual analytics: Definition, process, and challenges. In *Information visualization*. Springer, LNCS, 154–175.

[11] Chee Sun Liew, Malcolm P Atkinson, Michelle Galea, Tan Fong Ang, Paul Martin, and Jano I Van Hemert. 2016. Scientific workflows: moving across paradigms. *ACM Computing Surveys (CSUR)* 49, 4 (2016), 1–39.

[12] Roger D Peng. 2011. Reproducible research in computational science. *Science* 334, 6060 (2011), 1226–1227. https://doi.org/10.1126/science.1213847

[13] Dario D Salvucci and Joseph H Goldberg. 2000. Identifying fixations and saccades in eye-tracking protocols. In *Proceedings of the 2000 symposium on Eye tracking research & applications*. 71–78.

[14] Arvind Satyanarayan, Kanit Wongsuphasawat, and Jeffrey Heer. 2014. Declarative Interaction Design for Data Visualization. In *ACM User Interface Software & Technology (UIST)*. http://idl.cs.washington.edu/papers/reactive-vega

[15] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. 2016. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific data* 3 (2016), 160018.

[11]https://github.com/nirdslab/streaminghub-conduit/