UNIVERSITY OF SOUTHAMPTON

Faculty of Engineering and Physical Sciences
Electronics and Computer Science

Reducing Streaming Data Storage Required for Provenance Retrieval Using Fourier Transform

by

Zheng Huang

A thesis for the degree of
Master of Philosophy

January 2023

University of Southampton

Abstract

Faculty of Engineering and Physical Sciences
Electronics and Computer Science

Master of Philosophy

Reducing Streaming Data Storage Required for Provenance Retrieval Using Fourier Transform

by Zheng Huang

In this work, we investigate existing works on provenance in the streaming environment. Despite the various reduction techniques proposed for provenance or stream storage, the storage for whole source and intermediate streams is always necessary to answer how-provenance ("show me the data and operations that lead to the output data"). This makes the size of streaming data required for provenance retrieval unworkably large.

In this work, we investigate a method for manipulating the streams that provides information to answer how-provenance without pre-determining what information to keep and what to remove. We look at the Fourier transform (FT) as a tool to encode portions of the streaming data information for provenance retrieval.

We use a real-world, respiratory streaming use case to highlight the needs for provenance information. We build our stream reduction model and test it against the use case. The experiments show that FT can reduce the size of streaming data (in our demonstration of the technique over the second one-minute time window, it leads to a 15.7 times reduction effect for eligible streams for a streaming application to get the respiration rate and a 36.6 times reduction effect for eligible streams for a streaming application to find the best position), yet the utility of the streaming data for provenance retrieval comes with some limitation. While using the FT technique doesn't affect the answerability of the query that requires stream ID but not specific data (such as PQ1 for the use case), the query that requires examining data content (such as PQ2 for the use case), and the query that requires contributing operators (such as PQ3 for the use case), for the query that requires returning the figure of the stream (such as PQ4 and PQ5 for the use case), it's possible that there can be some pattern losses. The post-processing time for respiration sensor data from the second one-minute window is reasonable enough (0.873 seconds for a streaming application to get the respiration rate and 2.816 seconds for a streaming application to find the best position) to support more reactive provenance retrieval which is usually desirable for stream processing. While there is no significant difference between the medians of the query time using unreduced streaming data and reduced streaming data for PQ1 which doesn't require specific data at a 5% significance level, there exists a positive shift in the median of query time from the query using original data to query using reduced data at the 5 % significance level for PQ2, PQ4, and PQ5 which requires reconstruction of specific contributing data. The use of the FT technique doesn't affect the query time for PQ3 as the contributing operators can be retrieved from the table storing the metadata.

Contents

List of Figures

List of Tables

Declaration of Authorship

I , Zheng Huang, declare that the thesis entitled *Reducing Streaming Data Storage Required for Provenance Retrieval Using Fourier Transform* and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. None of this work has been published before submission

Signed:..... Date:.....

Acknowledgements

I would like to thank Professor Vladimiro Sassone for giving me the opportunity to do my research at the University of Southampton and for his hospitality when I first arrived.

I would like to thank my supervisors, Professor Adriane Chapman and Professor Neil M. White for their guidance and insight, their patience, kindness, and energy which is extraordinary, especially in this challenging time. I'm much obliged to them.

I would like to thank Doctor Gary Wills and Professor Christopher James for their feedback on the thesis.

I would also like to thank all my friends from home and from Southampton for their support, friendship, and encouragement.

I would like to thank the University of Southampton for its well-rounded support.

I gratefully acknowledge financial support from China Scholarship Council.

Finally, I would like to thank my family for their constant support.

Chapter 1

Introduction

1.1 Motivation

Provenance refers to the information describing the source and the production process of a product, be it a physical (e.g., a piece of art) or an immaterial object (e.g., an experimental result) (Moreau et al. (2009)). More specifically in computer science, it refers to a set of semantic relationships between data and processes which also includes basic metadata such as timestamp, ownership, description of the data and processes etc (Allen et al. (2010a); Allen et al. (2011); Allen et al. (2012); Buneman and Tan (2019)). It is useful in finding the cause of an error, understanding how a result came into being, and estimating data quality and data reliability (Cheney et al. (2021)). Use scenarios include, for example, detecting suspicious behaviours in data collaborations (Allen et al. (2011)), supporting results explainability in earth science findings (Olaya et al. (2022)), achieving shared understanding in an operational agile command and control (C2) system (Allen et al. (2012)), estimating biological and biomedical data quality in life sciences research (Jagadish and Olken (2004)), and so on. There are plenty of works dealing with provenance capture, storage, and use within transactional systems, which typically involve a discrete (and relatively low-rate) request-response style of interaction (Wang et al. (2007)).

However, relatively limited work deals with provenance within the streaming environment (see Section 2.3). These works can be classified into two main categories according to when provenance is retrieved: on the fly with the stream processing result (Chen and Plale (2015); Suriarachchi et al. (2018)) or later when requested (Wang et al. (2007); Huq (2013); Pouchard et al. (2018)). In this work, we focus on persistent provenance that can be retrieved later for answering how-provenance ("show me the data and operations that lead to the output data". See Section 2.2.1 for query overview). Several works have dealt with the persistent storage of stream provenance and reduction strategies are proposed (see Section 2.3.1). Yet despite the various

reduction techniques presented for provenance or stream storage, the storage for whole source and intermediate streams is always necessary to answer how-provenance which makes the size of streaming data required for provenance retrieval still large.

1.2 Research questions

Given the lack of research into the persistent storage of large amounts of streaming data from stream processing, this work will investigate a method for reducing the storage requirement for streaming data, while still retaining its utility for provenance retrieval.

We consider a technique usually used in signal processing called Fourier transform (FT). It converts the signal into a series of frequency bins of different magnitudes and phases. Depending on the nature of the signal, sometimes this process gives back just a few frequency bins with different magnitudes and phases which significantly reduces the size of the signal.

Different from a signal which is usually more carefully obtained and processed in an offline manner, the streaming data can come in various sampling intervals with different types of data content. Even eligible streaming data that possesses constant sampling with numerical data content has the intrinsic feature of being processed in the relatively short time window. This poses yet another challenge to explore whether implementing FT on the source or intermediate stream can correspondingly support more reactive stream provenance retrieval.

The aim of the thesis is to explore the applicability of the signal processing technique -the Fourier transform in reducing streaming data while retaining its utility for provenance retrieval.

3 research questions are proposed for exploration.

RQ 1 What are the types of provenance questions that users are likely interested in answering from a streaming application?

RQ 2 Does the well-known signal processing technique -FT provide the ability to reduce streaming data while also retaining its utility for provenance retrieval?

RQ 3 What is the performance impact of using FT to reduce streaming data for provenance retrieval?

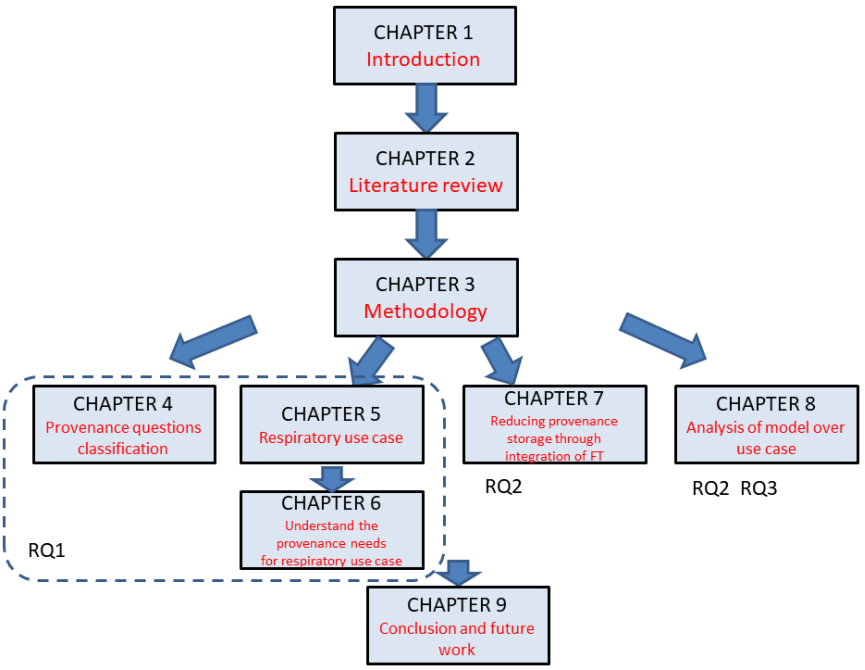

FIGURE 1.1: Illustration of thesis structure

1.3 Thesis structure

The remainder of this thesis is structured as follows: Chapter 2 introduces definitions related to stream processing, literature reviews on provenance, provenance within the streaming environment, and methods for compressing streaming data. In Chapter 3 we describe the methodology used to answer 3 research questions. In Chapter 4 we list provenance questions that summarize the use of provenance in answering RQ1. In Chapter 5 we describe in detail a medical sensing stream use case. Chapter 6 describes specific provenance needs for the medical sensing stream use case that also answers RQ1. In Chapter 7 we show how Frourier transform can be used for streaming data to succinctly store necessary data. This is to answer RQ2. In Chapter 8 we evaluate this approach in the case of the medical sensing stream to answer RQ 2 and RQ3. Chapter 9 outlines the conclusion and future work. The thesis structure is highlighted in Figure 1.1.

1.4 Contributions

The main contributions of this work can be summarized as follows:

- Analysis of current stream provenance approaches and their reduction techniques for provenance or stream storage and the identification of a gap (shown in Chapter 2)

- A classification of provenance questions for stream processing system (shown in Chapter 4)
- A specific analysis of provenance needs for a real respiratory streaming use case (shown in Chapter 6)
- Design to integrate Fourier transform in streaming data (shown in Chapter 7)
- Comparison between provenance retrieval using original streaming data and provenance retrieval using reduced streaming data in terms of post-processing time, storage consumption, the ability to answer provenance questions, and query time for a medical sensing stream application and its implication (shown in Chapter 8)

Chapter 2

Literature review

In this chapter, we first describe definitions related to stream processing in Section 2.1. In Section 2.2, we introduce definition of provenance, a set of standardised, theoretical provenance queries, the W3C PROV standard, and capture mechanisms for provenance along with their strengths and weaknesses. In Section 2.3, we describe literature on provenance within streaming environment. In Section 2.4, we introduce methods for compressing streaming data.

2.1 Stream processing

In this section, we give an introduction to streaming data in Section 2.1.1. In Section 2.1.2, we describe what window and stream processor are and extract a list of stream operators.

2.1.1 An introduction to streaming data

The **streaming data** in stream processing refers to a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items (Golab and Özsu (2003)). Arrival time can also be called ingestion time. The sampling interval of the streaming data can be constant or variable due to, for example, the broken sensor (using event timestamp) (Vijayakumar and Plale (2006)) or network delay (using ingestion time) (Huq (2013)), or the nature of the stream such as selling quote for a store (using event timestamp) (Huq (2013)) (See below in Section 2.1.2 for definitions for event time and ingestion time). Stream processing is needed in scenarios where real-time insight is of high value. These application scenarios include stream processing in the physical environment domain such as meteorology forecasting (Vijayakumar and Plale (2006)), stream processing in the health care domain such as

element type \ sampling interval	constant	variable
numerical	temperature measurement taken at a constant interval (Huq (2013)); continuous ECG readings from a cardiac monitor (Wang et al. (2007))	selling quote for a store (Huq (2013))
string	sentiment log documented at regular interval; weather report updated regularly	twitter stream from one individual

TABLE 2.1: Summary of different types of streams with examples

Symbol	Description
S_{in}	input stream
$\mid S_{in} \mid$	the number of data in the input stream
E	a single stream record combined of timestamp and data element
t	timestamp
e	data element
W	a window in abstraction
$\mid W \mid$	size of a window
W_{time}	time-based window
$\mid W_{time} \mid$	size of the time-based window
W_{count}	count window
$\mid W_{count} \mid$	size of the count window
T'	trigger interval for the time-based window
N'	trigger interval for the count window
A	segment of streaming data that can contain one window or some windows of streaming data

TABLE 2.2: List of symbols used

health monitoring (Chowdhury et al. (2010)), and stream processing in the social environment such as the stock market (De Pauw et al. (2010)).

For the clarity of the rest of the thesis, we use notations to denote concepts within the streaming environment as shown in Table 2.2. We assume the stream elements arrive in time order. Let S_{in} be the input stream. We then have $S_{in} = \{E_1, E_2, E_3, ...\}$, where E_1 is the combination of t_1 (the timestamp of the first element) and e_1 (the first element), E_2 is the combination of t_2 (the timestamp of the second element) and e_2 (the second element), and so on. The stream S_{in} can be rewritten as $S_{in} = \{\{t_1, e_1\}, \{t_2, e_2\}, \{t_3, e_3\}, ...\}$. The elements can be strings, numerical values etc. We let $\mid S_{in} \mid$ be the number of data in the input stream. Table 2.1 summarizes different types of streams with examples. The example streams all use event timestamps.

2.1.2 Window and stream processor

Time-based window

The notion of time is usually discussed before talking about the time-based window concepts in stream processing. In Apache Flink [1], for example, there are three different notions of time, viz. processing time, event time, and ingestion time. For processing time, windows are defined according to the wall clock of the machine that builds and processes a window. For event time, windows are defined according to timestamps attached to each event record. For ingestion time, event records are assigned wall clock timestamps as soon as they arrive in the system and then processed with event time semantics based on the attached timestamps. We decided that the discussion of event time and ingestion time is enough for this work. Suppose the time-based window is sized T ($\mid W_{time} \mid = T$) in terms of time. Whether the time points mentioned above in the stream model are event times or ingestion times, the first time-based window collects elements whose timestamps sit in the range of $[t1, t1 + T)$. The difference is that the result computed using event time reflects what has happened for events happening in history from time $t1$ to time $t1 + T$. Whereas the result computed using ingestion time reflects what has happened in arrival time $t1$ to $t1 + T$.

Count window

The count window is defined based on the number of elements within that window. For example, if the count is fixed as 100, every window will have 100 elements. The size of the window in terms of time does not matter. Suppose the count window is sized N ($\mid W_{count} \mid = N$) in terms of element number. The first count window will collect the first N elements, i.e., E_1 to E_N regardless of the time type.

Tumbling window

The tumbling window dissects a stream into non-overlapping finite sets. A tumbling window can be a time-based or a count window. If the window is defined as a tumbling time-based window size T, the second window will collect elements whose timestamps sit in the range of $[t1 + T, t1 + 2T)$. The result computed using event time reflects what has happened for events happening in history from time $t1 + T$ to time $t1 + 2T$. Whereas the result computed using ingestion time reflects what has happened in arrival time $t1 + T$ to $t1 + 2T$. If the window is defined as a tumbling count window sized N, the second window will collect elements E_{N+1} to E_{2N}.

Sliding window

Sliding windows are overlapping windows that result in more smoothed responses. A sliding window can be a time-based or a count window. If the window is defined as a

[1] https://flink.apache.org/news/2015/12/04/Introducing-windows.html

sliding time-based window sized T with a trigger interval T' (T' shall be smaller than T), the second window will collect elements whose timestamps sit in the range of $[t1 + T', t1 + T' + T)$. Again the result computed using event time reflects what has happened for events happening in history from time $t1 + T'$ to time $t1 + T' + T$. Whereas the result computed using ingestion time reflects what has happened in arrival time $t1 + T'$ to $t1 + T' + T$. If the window is defined as a sliding count window sized N with a trigger interval N' (N' shall be smaller than N), the second window will collect elements $E_{N'}$ to $E_{N'+N-1}$.

A **stream processor** is a task that executes continuously on the incoming data. These tasks can be defined as database queries or a pipeline of computational entities that operate on the data (Vijayakumar and Plale (2006)). There are stateful and stateless processors. A stateful processor requires a state stored to process input and produce output, whereas a stateless processor does not require a state stored for processing [2] (Wang et al. (2007)).

We extract operators that commonly used in a modern stream processing system called Apache flink [3], operators appear in properties of Fourier transform (FT) (see Section 2.4.1.2), some operators from MATLAB that we will use later as the building blocks for stream processing (see Chapter 5) as listed in Table 2.3.

In summary, stream processing possesses several features: 1)The data being analysed shall be time-ordered data; 2)The time window for collecting data shall be relatively short, like some seconds, minutes, or hours; 3) New data are expected to come in when the processing is going on.

2.2 Provenance

Provenance refers to the information describing the source and the production process of a product, be it a physical (e.g., a piece of art) or an immaterial object (e.g., an experimental result) (Moreau et al. (2009)). This is provenance in big concept.

In practice in computer science, provenance refers to a set of semantic relationships between data and processes. It also includes basic metadata such as timestamp, ownership, and description of the data and processes. It can also include annotations such as quality information (Allen et al. (2010a); Allen et al. (2011); Allen et al. (2012); Buneman and Tan (2019)).

[2] https://kafka.apache.org/20/documentation/streams/developer-guide/dsl-api.html#stateless-transformations

[3] https://nightlies.apache.org/flink/flink-docs-master/zh/docs/dev/datastream/operators/overview/

Classification	Name	Description	Reference
Data transformation	map	takes one element and produces one element. It involves one stream. It is stateless.	flink
	scaling	takes in a stream and scale it in time by a constant. It involves one stream. It is stateless.	Section 2.4.1.2
	shift	takes in a stream and shift it in time by a constant amount. It involves one stream. It is stateless.	Section 2.4.1.2
	filtfilt functor	does zero-phase forward and reverse digital IIR filtering. It filters out the noise and smooths the stream. It involves one stream. It is stateless.	MATLAB
Data augmentation	flatmap	takes one element and produces zero, one, or more elements. It involves one stream. It is stateless	flink
Data reduction	filter	evaluates a boolean function for each element and retains those for which the function returns true. It involves one stream	flink
	findpeaks functor	finds peaks of the stream. It involves one stream. It is stateless.	MATLAB
	length	returns back the length of the stream. It involves one stream. It is stateless.	MATLAB
User-defined functors	-	functors defined by users	-

TABLE 2.3: Summary of different operators

In the context of an application, what will be recorded in provenance is usually application dependent and motivated by the user's needs. In one case, people want to have a comprehensive understanding of the sales figure in the company earning report that was generated by some source data and some intermediate sales results from specific regions. Then apart from the source data and the program that was used to generate the report, programs used to generate the intermediate results may also be tracked (Buneman and Tan (2019)). In another case where software reproducibility is of importance, typically, things like the version of the operating system and software libraries are required to be documented on top of the data and program used for the experiment (Buneman and Tan (2019)). On a more practical note, for example, in order to enable detailed debugging needs of the user, use cases were reviewed and extracted from forums such as the FAQ section on the Orange website prior to the collection of provenance in Orange 3, an open-source data visualization, machine learning, and data mining toolkit (Chapman et al. (2021)).

The choice of granularity for provenance again depends upon the user's need. The process can range from a single operator, a function to a whole program (Chapman et al. (2008)). The data can be as small as a tuple (a row of values in a relational database) and as big as a document. Chapman et al. (2021), for example, used different provenance instrumentation techniques containing GUI-based insertion, embedded within scripts, and via expert hand-encoding to investigate how granularity of data provenance affects instrumentation costs and query answer-ability. Their analysis shows that though varied in detail and modelling, the three approaches capture the same type of information. For query answer-ability, the expert hand-encoding approach and GUI-based insertion approach can only answer parts of the questions while the scripts embedded approach can answer all questions since it captures provenance about specific data changes. For the cost, while the GUI-based approach may have fewer calls than the script-based approach, it comes with some limitations.

2.2.1 Query overview

We've introduced provenance in terms of the big concept, in terms of computer science, and in the context of an application. And over the years, the provenance community has identified a set of more standardised, theoretical provenance queries that execute on the provenance and data collected. For example, why-provenance asks for the source data that contributes to the output data (Cui and Widom (2000); Cui et al. (2000)). Why not-provenance attempts to understand why some data are not part of the result of a query (Chapman and Jagadish (2009); Bidoit et al. (2014); ten Cate et al. (2015)). How-provenance describes the input data (source and intermediate data) and the operations that lead to the output data (Buneman and Tan (2019); Chapman et al. (2020)). Where-provenance tries to identify locations in the source database from which the data item was copied (Buneman et al. (2001)).

2.2.2 W3C PROV standard

To enable the interoperability of provenance information in heterogeneous environments such as the Web, a W3C standard, W3C PROV was proposed. It refers to a set of PROV Documents which defines a model, corresponding serializations and other supporting definitions (Groth and Moreau (2013)). An illustration of PROV Core Structures is shown in Figure 2.1.

In PROV, entities can be digital objects such as a web page, physical things such as a book as well as abstract concepts and ideas. 'An activity is something that occurs over a period of time.' Activities can contain a broad range of notions: information processing activities may be moving digital entities; physical activities can include printing a book. 'An agent is something that bears some form of responsibility for an

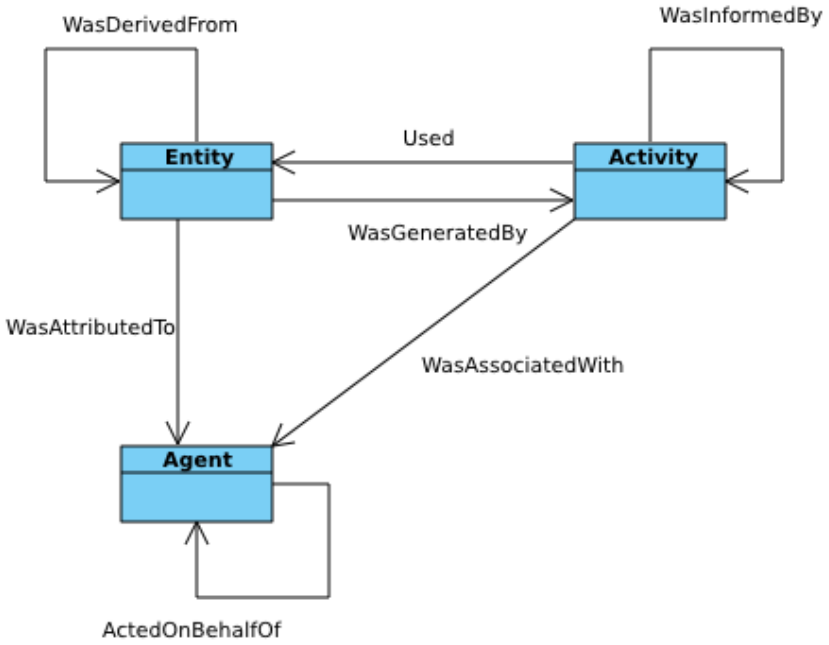

FIGURE 2.1: PROV core structures (Moreau et al. (2013))

activity taking place, for the existence of an entity, or for another agent's activity.' Activities utilize entities and produce entities. A transformation of a pre-existing entity into a new entity is described as a derivation. Communication between two activities is written as wasInformedBy. 'Communication is the exchange of some unspecified entity by two activities, one activity using some entity generated by the other'. An agent can act on behalf of another agent, each bears some responsibility for the activity that took place (Moreau et al. (2013)).

PROV, though being an inter-operability standard for provenance, is not a storage standard for provenance. There are many other ways to store provenance such as using a relational data model, semi-structured data model, graph-based data model and so on (Holland et al. (2008); Cheney et al. (2018)).

2.2.3 Capture mechanisms for provenance

In this section, we categorize provenance capture mechanisms according to where the capture point is placed and where the provenance is captured from. Section 2.2.3.1 introduces capturing provenance at the coordination point. Section 2.2.3.2 describes capturing provenance from an application. Section 2.2.3.3 introduces capturing provenance from execution snapshot. Section 2.2.3.4 describes work that combines different capture mechanisms.

2.2.3.1 Capturing provenance at the coordination point

Coordination points between multiple systems or programs in an operating system are generally considered the most efficient place to capture provenance. They serve as coordinators and negotiators on top of different systems or programs and can therefore provide a wide breadth and coverage of provenance (Allen et al. (2012)). Typical examples of coordination points include Enterprise Service Bus (ESB), UNIX kernel, and so on. Detailed descriptions and analyses of such examples are listed as follows.

i) Capturing across machines

In almost every distributed system, there exists a point of coordination which connects different systems or applications. In the client-server case, for example, the point of coordination is the server (Allen et al. (2010b)). We present a few possible places for capturing cross-machine provenance.

Example 1: enterprise service buses

'An enterprise service bus (ESB) is an abstraction on top of a common messaging architecture.' It allows multiple applications to be coordinated by routing and sequencing messages between those applications (Allen et al. (2010b)).

Proof-of-Concept Example: The Mule ESB (Allen et al. (2010a); Allen et al. (2010b))

By using the Mule ESB as a test bed, the IM-PLUS project has demonstrated the feasibility of provenance capture through ESBs.

In this approach, Mule's "Envelope Interceptors" were added to each of the services configured on the Mule ESB. The MULE Capture Agent (MCA) uses MULE's "Envelope Interceptor" to form provenance graph. It also has access to MULE's metadata relevant to service invocations, adding richer information to the node. Finally, MCA uses several techniques to introspect into messages when future provenance queries might need it.

The advantage of this approach is that no underlying application modification or user knowledge is needed. Also, no modifications to the capture mechanism are needed as applications evolve over time.

The drawback, as with all coordinating point capture mechanisms, is that applications are treated as "black boxes". This may limit the fidelity of analyses that this type of collection can provide.

Example 2: the Ozone Widget Framework (OWF) (Allen et al. (2012))

'The Ozone Widget Framework (OWF) is a messaging framework that allows applications, called widgets, to be run within a web browser.' It is used within the

Intelligence Community for providing application and data resources built and released by any organization to analysts. The OWF provenance collector thus provides opportunities to capture provenance about interactions across different organizations.

There are three possible methods for a widget developer to access data or processes in a web environment. The OWF thus presents a challenge for provenance capture: finding appropriate points to insert provenance collection calls.

Despite that, all three capture methods corresponding with the three accessing methods would be scalable with dozens or hundreds of different applications on top of OWF.

Example 3: web proxy

'In computer networking, a proxy server is a server application that acts as an intermediary between a client requesting a resource and the server providing that resource' [4]. As a result, a proxy is able to see all data requests sent from the client, as well as the response. This makes it a potential place to put provenance capture points for tracing public data use.

This method can offer interesting information concerning who owns the data, or whether the data comes from untrusted sources.

One drawback of capturing provenance within a web proxy is that it does not offer much detail on the nature of the processes running on remote web servers, other than their names, inputs and outputs (Allen et al. (2012)).

Example 4: workflow system

A workflow system is a highly specialized system owning a defined sequence of tasks. 'The workflow system controls the calling of processes and the access of data.' Thus placing provenance capture points within the workflow system itself enables us to see all – particularly environmental parameters to which other capture methods may not have access (Allen et al. (2012)).

ii) Capturing in a single OS on a single machine

In a single machine, a similar approach can be used for capturing provenance by observing interactions between programs in the operating system. Below we list an example OS-level capture solution.

Example 1: Provenance-Aware Storage Systems (PASS) (Muniswamy-Reddy et al. (2006))

PASS modified the Linux kernel. A collector generates a provenance record for each provenance-related system call and binds the record to the appropriate structure.

[4] https://en.wikipedia.org/wiki/Proxy_server#cite_note-1

This approach has the advantage of seeing everything that occurs in the file system. There is no need to make applications provenance-aware.

However, it has several disadvantages: 1)it is not possible to automatically collect provenance when data originates from a non-PASS source, such as a user, another computer, another file system that is not provenance-aware, or files that are transmitted across the net. 2)provenance can be quite low-level and contains possibly uninteresting information such as intermediate file read and writes (Allen et al. (2012)).

2.2.3.2 Capturing provenance in a single application

Capturing provenance from coordination points typically observes the interactions between different applications. It does not log how a single application accomplishes its individual task. However, in some scenarios where richer and more interesting domain-specific analyses are desired, it is worthwhile to modify the individual application to generate more detailed provenance. The downside of this approach is that it can be very labour-intensive. Further modifications are also required as applications evolve over time. In addition, application modification is not always workable since the access to application source code is not often available. We list an example approach as follows.

Example 1: RDataTracker (Lerner and Boose (2014))

RDataTracker is an R library that supports the collection of data provenance from executed R script. It relies upon two basic techniques. First, the user adds calls to functions defined in the RDataTracker library. Second, the called functions utilize the capabilities that R provides for examing runtime state, including the call stack and variable bindings, thus capturing more detailed information.

2.2.3.3 Capturing provenance from execution snapshot

All the above approaches assume that provenance capture is implemented during execution, which can introduce a certain amount of overhead. A different approach is to extract provenance from the execution snapshot after a program is finished. The advantage of such a strategy is that it keeps down the runtime overhead. Furthermore, it enables provenance capture to be performed after analysis needs are better known. However, the granularity of the provenance that this type of capture can offer is pre-determined and may not be suitable for later analysis. The following sections detail examples of such approaches.

i) Capturing from the memory snapshot

Example 1: PROV$_{2R}$ (PROVenance Record and Replay) (Stamatogiannakis et al. (2017))

In this approach, an execution trace is generated as raw material for iterative user-defined provenance capture and analysis. The concrete implementation stages are as follows: 1)execution capture: an execution trace (an even lower semantic level than the kernel) is recorded. 2)application of instrumentation: an instrumentation plugin is selected based on the goal of the analysis to process the execution trace and generate a provenance graph. 3)provenance analysis: 'the user interrogates the provenance graph using a query language to focus on, and/or select portions of the graph.' 4)selection and iteration: 'based on the provenance analysis, the user can select a portion of the execution trace on which to apply additional, more intensive, instrumentation. To do this, the user starts again from stage 2.'

All execution was done on a single machine in the scenario proposed by the authors. They did, however, describe possible ways to apply their approach to the distributed setting. The idea is that execution traces of distributed machines will be sent to a central provenance store. 'Execution traces of interest can be replayed at the provenance store' and 'the provenance generated by distributed machines can be connected through a number of possible mechanisms.'

The strengths of this approach lie in that: 1)decoupling provenance capture and analysis from execution keeps the runtime overhead at manageable levels. 2)it enables provenance capture to be performed after analysis needs are better known. 3)it offers unparalleled flexibility in the types of provenance analyses that can be used.

ii) Capturing from log file

The information contained in a program's execution log is in many ways similar to provenance information. Therefore it should not be neglected as a potential source for provenance information. Generating provenance from log files generally involves a log file parsing program (Allen et al. (2010b)).

Example 1: Git2PROV (De Nies et al. (2013))

De Nies et al. (2013) came up with the idea of mapping versioning of data from the version control system (VCS) to provenance information. They built a lightweight RESTful Web service to demonstrate the idea.

2.2.3.4 Combination of different capture methods

Rupprecht et al. (2020) present URSPRUNG, a transparent provenance collection system which combines provenance collected from the underlying compute and storage system (e.g., OS and file system) with application-specific provenance

Main capture method	Main advantage	Main disadvantage
Capturing at the coordination point	wide breadth of provenance	little knowledge about the internal of individual application
Application modification	more detailed provenance	labour-intensive; source code not always available
Execution snapshot	reduce runtime overhead	pre-determined granularity

TABLE 2.4: Comparison of different capture mechanisms

information, collected through a rule-based language. The event aggregation model in URSPRUNG can aggregate lower-level events into higher-level semantics relevant to a user.

2.2.3.5 Summary of provenance capture

The choice of provenance capture approaches largely depends on the requirements of the project at hand. Still, there are some general observations on making good choices.

The most promising and general place to capture provenance is the coordination point. When more specific domain provenance is required, application modification for provenance should come in place. Finally, when application source code is not available or when capturing during run time introduces too much overhead, it is worth considering capturing provenance from the after-the-fact execution snapshot.

We summarize the main advantages and disadvantages of different capture mechanisms in Table 2.4

2.3 Provenance within the streaming environment

Vijayakumar and Plale (2006) first tried to tackle the sub-problem of provenance of stream in the physical environment domain such as the atmosphere, soil, ocean etc. The goal of their work is to use provenance to partially account for the inaccuracies caused by things like when radars go down in the unpredictable physical environment. To them, the communication between the internal components of the stream filtering system is not as important as the provenance for each stream and the filters that execute on the streams.

The authors listed several challenges provenance collection will impose on stream filtering system to motivate a data model and a collection model for stream provenance tracking.

They identified the smallest unit (granularity) for provenance collection. They would like to trace the source of events in derived streams back to the events of input streams without identifying every events individually. They decided to put the capture place in the filters because they are internal to the system. They anticipated that this will impose a low overhead compared with collecting provenance for data stream. The filters being discussed can be dynamically deployed and are subjected to re-configuring on the fly. For example they can change to approximation mode when encountering missing stream.

Figure 2.2 shows a sample provenance document for a derived stream. It contains basic information about the stream, its input streams, and changes happened to the stream.

They discussed provenance collection model in the context of Calder grid-based stream processing system. In their system, each stream consists of a sequence of time ordered events and each event is associated with a timestamp. 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. After that, information is only logged whenever changes happens (e.g., filter mode changes). A derived stream can then trace back to its input stream and filters. They argued that this type of provenance can support deducing the accuracy of the derived stream events given a well-defined formula in a particular domain.

They then discussed a meteorology forecasting project and its provenance service.

The pros of the provenance approach are: 1) it is fitted for documenting dynamically deployed filters; 2) it incurs low storage overhead.

The con of the provenance approach is that it can not identify dependency for individual data elements.

The limitation of the work lie in: 1)window concept is not discussed in the paper; 2) the notion of time is not discussed in the paper; 3) the paper is vague about how a particular set of events can be traced back to the events that caused it.

Wang et al. (2007) proposed a simple Time-Value Centric (TVC) provenance model in order to validate the processing logic or perform fault-diagnosis in the case of anomalies in the context of healthcare IT infrastructure.

Being aware of the limitation of the work of Vijayakumar et al. in which only the dependency relationships among streams instead of elements are stored, they came up with a model that can identify the dependencies for data elements. In the model, each output data element must have a well-defined timestamp and be associated with a stream possessing a stream ID and a PE (processing element) ID. The author of a PE is

```
<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>
```

FIGURE 2.2: Sample provenance document for a derived stream in XML format (Vijayakumar and Plale (2006))

responsible for manually specifying the IDs and logic of each rule block. Such rule blocks are stored in the provenance storage subsystem.

In order to retrieve the input data that a specific output data depends on, first the "rule dependency" block associated with the output data will be retrieved. The stream IDs will be determined by resolving the dependency and a SELECT query will be issued to retrieve the data possessing the relevant stream ID and having a timestamp within the corresponding time window.

The pro of the provenance approach is that it can identify dependency for individual data elements.

The cons of the provenance approach are: 1) it is not applicable for operators whose dependency rule can not be predefined; 2) the internal logic of PE is unknown.

Misra et al. (2008) carried out an initial implementation with a TVC-based provenance solution for Century, a health IT infrastructure. One of the challenges revealed from their experience is the storage load of storing both external and intermediate streams. They then proposed a new approach called Composite Modeling With Intermediate Replay (CMIR) that does not require persistence of all intermediate streams. However,

to the best of our knowledge, there is no follow-up implementation of the CMIR model.

Sansrimahachai (2012) aims to address a form of "why provenance". He first designed a fine-grained provenance model. Then he devised inverse functions (stream ancestor functions) for common stream operation: Map, Filter, Sliding time window, Length window, Time-window join, and Length-window join to return the provenance of a certain stream processing result. He further introduced optimized stream ancestor functions to reduce the stream storage by cutting the content of intermediate streams and only maintain the first input events. Fine-grained provenance can be obtained using stream ID, the temporal time order, and the business logic of processing operations. He proposed persistent provenance storage as well as a novel on-the-fly-tracking mechanism for provenance. (Sansrimahachai et al. (2012); Sansrimahachai et al. (2013))

We only discuss the persistent approach here. Though the approach can reduce stream storage, it is done by cutting the content of intermediate streams. Thus the approach can only address why-provenance but not how-provenance.

Huq (2013) assumed that the stream processing infrastructure utilizes processing time, he came up with a suite of inference-based methods to infer fine-grained data provenance for stream addressing different execution environments at reduced storage cost. His work took system dynamics such as processing time and sampling interval into consideration.

For the basic provenance inference method, Huq documents the workflow provenance (prospective provenance - what is planned to happen instead of what actually happened (Allen et al. (2011))) before the actual execution of stream processing. Figure 2.3 shows an example of such workflow provenance. During execution, each tuple is associated with a transaction time. The processing delay of a processing element is observed beforehand. Thus the output tuple can be linked to the input tuple(s) that contribute to it. They argue that through this approach, the outcome result can be validated with high accuracy (Huq et al. (2011)).

Figure 2.4 and Figure 2.5 together show how provenance is actually computed with a project operator. The sampling interval is 2 time units. We thus have t1, t3, t5 until t9 as input. The window size is 3 tuples which means that 3 tuples are taken to produce a set of results. Since the processing delay is 1 time unit, we have 3 tuples timestamped t6 as output. The operator is triggered after the arrival of 1 tuple.

For backward computation, in order to know what window of tuples contribute to the chosen tuple, a simple formula '$upperBound = referencePoint - processingDelay$' is used. In this case, $upperBound = 8 - 1 = 7$. Since the size of the window is 3 tuples, we thus conclude that the window containing t3, t5, and t7 contributes to t8.

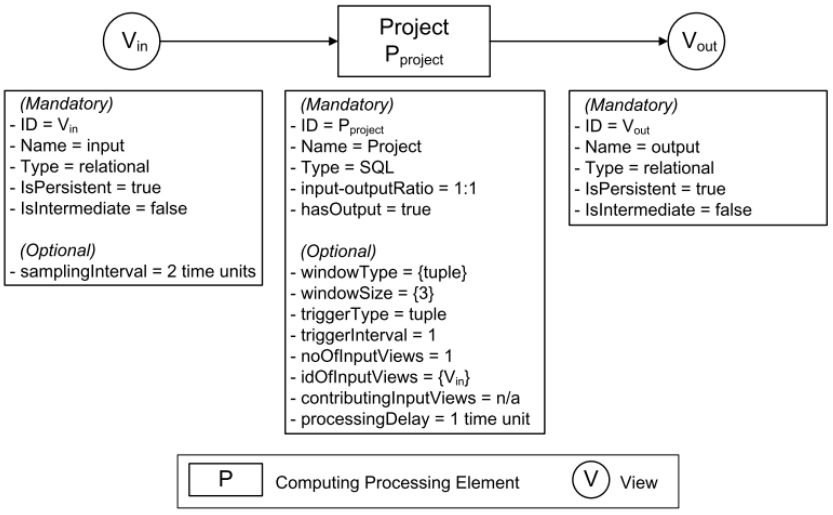

FIGURE 2.3: Example of the explicated workflow provenance (Huq et al. (2011))

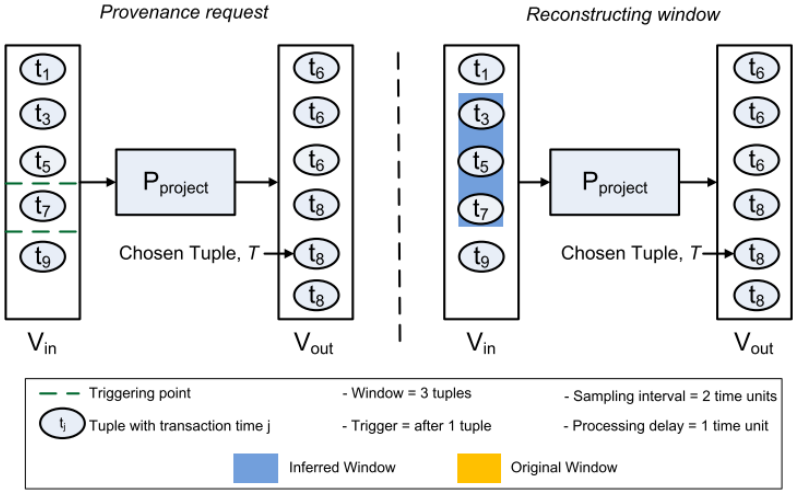

FIGURE 2.4: Backward computation (Huq et al. (2011))

For forward computation, since the chosen tuple is in the second position, we thus conclude that t5 contributes to the chosen tuple. Forward computation is subjected to the logic of the operator.

Huq then further proposed the probabilistic provenance inference method and the multi-step probabilistic provenance inference.

This work resembles the TVC work to some extent, but it did better in that there is more detailed documentation of operators compared with TVC work.

Glavic et al. (2014) use tuple-identifiers (TID) to identify the tuples in the form of stream-id: tuple-id. They then modify the behaviour of operators to generate and propagate these TID through several operators of a query network. They propose two provenance generation and retrieval methods (eager vs. lazy). For the eager method, they propose 3 provenance compression methods namely interval encoding, delta

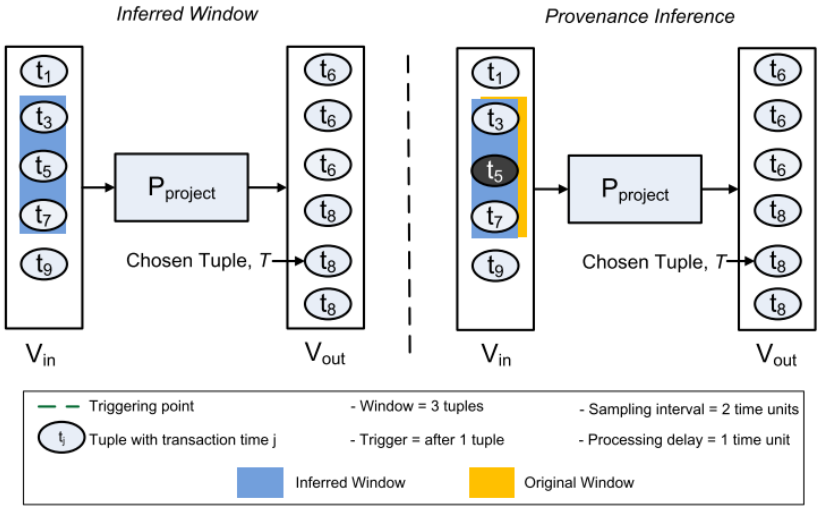

FIGURE 2.5: Forward computation (Huq et al. (2011))

encoding, and generic dictionary compression. Provenance here can be used for diagnosis and assurance, revision process, and query debugging.

The pro of operator instrumentation used in their work is that non-determinism can be handled in a natural way.

Chen and Plale (2015) identify the large volume of provenance in scientific experiments that run continuously for extended periods. They envision a solution that processes Big Data provenance as it is generated to provide real-time results. Thus no provenance data needs to be stored. They further propose the idea to store the processed provenance inside a temporal sliding window and through the internal state maintained in memory.

The pro of such a solution is that it incurs minimal storage.

The con is that it may not be suited for stream application that requires long-term storage for later retrieval.

Pouchard et al. (2018) reduce provenance data by storing only all the provenance and a detailed trace for a time window of interest before the detected anomalies, an approach they called prescriptive provenance.

The pro of their solution is that it strikes a balance between storage and utility for provenance.

The limitation is that this approach may constrain the type of application it can support (i.e., anomaly detection).

Suriarachchi et al. (2018) target at dealing with infinite provenance streams generated continuously from stream processing data-intensive computation (DIC). They thus

came to the recognition of the need for on-the-fly processing of provenance which is similar to the idea coming from the work of Chen and Plale (2015). Their proposed stream processing algorithm that processes fine-grained provenance is able to reduce provenance, preserve backward and forward provenance, and handle provenance events arriving out-of-order.

Gordani Shahri et al. (2021) identify the unwanted computational power and memory consumed of provenance for streaming applications when no event of interest requires it. They thus come up with the idea to activate/deactivate provenance recording based on user-defined rules. They present Twins, a new adaptive provenance tool built on top of Apache Flink and GeneaLog. Their result shows that Twin's performance resembles the query that never records provenance when its provenance is inactive. It resembles the query that always records provenance when its provenance is active (Erlandsson and Gordani Shahri (2021)).

2.3.1 Summary and the gap

The work of Misra et al. (2008) and Chen and Plale (2015) are excluded from the discussion and comparison for different stream provenance approaches as they are envisioned ideas.

Apart from Suriarachchi et al. (2018) who provide one pass provenance result in real time, all other works retrieve provenance after their provenance and streams are being stored in persistent storage. We will discuss different reduction techniques for provenance or stream storage for these works as this work intends to deal with persistent provenance for stream.

A table that summarizes reduction techniques for works that maintain persistent provenance storage is presented in Table 2.5.

As it is shown, there is no reduction technique for either the stream or provenance for the work of Wang et al. (2007) as it is not the focus of their work. The focus of their work is to get fine-grained provenance for medical event streams. Pouchard et al. (2018) and Gordani Shahri et al. (2021) reduce provenance storage by selecting some specific time of interest for collection. Vijayakumar and Plale (2006) reduce provenance by documenting coarse-grained provenance for the stream. This contains changes to the streams and filters with their corresponding duration. Sansrimahachai (2012) reduce stream storage by removing intermediate streams. Glavic et al. (2014) reduce provenance information by compressing provenance representations. Huq (2013) reduce provenance storage by pre-observing system dynamics to infer provenance, but this reduction technique is limited to be applied to provenance that uses processing timestamp.

Work	Reduction technique
Vijayakumar and Plale (2006)	Documenting coarse-grained provenance which contains changes to the streams and filters with their corresponding duration
Wang et al. (2007)	No
Sansrimahachai (2012)	Cutting content for the intermediate streams
Huq (2013)	Observing system dynamics beforehand to infer fine-grained provenance using processing time
Glavic et al. (2014)	Compressed provenance representations
Pouchard et al. (2018)	Collecting provenance only for a time window of interest preceding the detected anomalies
Gordani Shahri et al. (2021)	Activating provenance recording for a specific time based on user-defined rules

TABLE 2.5: Different reduction techniques for work maintaining persistent stream provenance storage

Despite various reduction techniques proposed for provenance or stream storage, the storage for whole source and intermediate streams is always necessary to answer how-provenance. There is a lack of the reduction technique that reduces source and intermediate streams.

2.4 Methods for compressing streaming data

In this section, we introduce Fourier transform, wavelet transform, and compressed sensing in Section 2.4.1, 2.4.2, and 2.4.3 respectively.

2.4.1 Introduction to Fourier transform

The **Fourier Series** (Zheng et al. (2011)) takes a **continuous periodic signal** in time (its period is T_1. Its radian frequency $w_1 = \frac{2\pi}{T_1}$. Its frequency $f_1 = \frac{1}{T_1}$) and decomposes it into the sum of a constant and a series of sines and cosines like this:

$f(t) = a_0 + \sum_{k=1}^{\infty}(a_k cos(kw_1 t) + b_k sin(kw_1 t))$

where $f(t)$ represents the original signal and k is a positive integer.

A **Fourier transform (FT)** (Zheng et al. (2011)) is implemented on a more generalized form of the above signal (including periodic and non-periodic signals). It calculates the coefficients at each particular frequency. To do that, the FT calculates the correlation between the original signal and analyzing function, the sinusoids:

$X(w) = \int_{-\infty}^{\infty} x(t)e^{-jwt}dt$

where $x(t)$ represents the original signal and $X(w)$ is the spectrum function of $x(t)$.

In reality, though, an analogue to digital converter cannot sample continuously. So what we have eventually is a set of discrete samples. In order to conduct Fourier transform on such samples, **Discrete Fourier Transform (DFT)** (Smith (2007)) is used:

$X(w_k) = \sum_{n=0}^{N-1} x(t_n) \cdot e^{-jw_k t_n}$

where

$t_n = nT, \Omega = \frac{2\pi}{NT}, w_k = k\Omega$

N represents the number of the samples. T represents the sampling interval (sec). t_n is the nth sampling instant. Ω is the radian-frequency sampling interval. w_k is the kth frequency sample.

Different from the continuous Fourier Transform which calculates the coefficient of any frequency we want, the DFT calculates coefficients of a set of frequency bins which are determined by the sampling frequency and the number of samples. The DFT assumes the sampled signal contains full period(s) from some periodic signal. When having fewer samples than the full period(s) of that periodic signal, windowing is introduced to deal with that (see Section 2.4.1.1). The disadvantages of the DFT can include picket-fence, leakage, and aliasing effects (see Section 2.4.1.1) which are most significantly manifested in the Fourier analysis of mixed-structure signals (Alexey and Olga (2020)).

The **inverse Discrete Fourier Transform (IDFT)** (Smith (2007)) converts the signal back from the frequency domain into the time domain. The IDFT is given by:

$x(t_n) = \frac{1}{N} \sum_{k=0}^{N-1} X(w_k) \cdot e^{jw_k t_n}$

where $n = 0, 1, 2, ...N - 1$

Note that a continuous-time signal needs to be uniformly sampled at the minimum-sampling rate so that the original signal can be reconstructed by these samples. This is often referred to as Shannon's sampling theorem (Lai (2003)).

To compute DFT directly from the definition is often too slow to be practical, thus a fast implementation of DFT called fast Fourier transform (FFT) came into being [5].

2.4.1.1 An Introduction to Windowing

The FFT is performed on a finite set of data. Since for the FFT, both the time domain and the frequency domain are circular topology, the two endpoints of the time waveform are interpreted as if they were connected together. When the measured

[5] https://en.wikipedia.org/wiki/Fast_Fourier_transform

signal is periodic and an integer number of periods fill the time interval, the FFT gives back neat spectrum line(s).

However, when the measured signal does not contain an integer number of periods, the two endpoints are discontinuous. This causes the fine spectral line(s) we get by a FFT to spread into a wider spectrum. This phenomenon is known as spectral leakage.

We can use a technique called windowing to minimize the effects of performing FFT over a non-integer number of periods. By multiplying the time record with a finite-length window with an amplitude that varies smoothly and gradually towards zero at the edges, the endpoints of the waveform meet.

There are different types of window functions we can apply depending on the signal. To choose a window function, we must estimate the frequency content of the signal.

The **Hanning** (Hann) window can satisfy 95 percent of cases. It has good frequency resolution and reduced spectral leakage. We can apply the Hann window even if we do not know the nature of the signal.

Different from the Hann window, the **Hamming window** does not quite reach zero at both ends and has a slight discontinuity in the signal. It thus does a better job of cancelling the nearest side lobe but a poorer job of cancelling any others.

The **Blackman-Harris** window is similar to Hamming and Hann windows. It results in a wide peak, but good side lobe compression. There are two main types of these windows——the 4-term Blackman-Harris window and the 7-term Blackman-Harris window.[6]

2.4.1.2 Properties of the Fourier transform

The Fourier Transform has a set of properties as follows [7]. We let the Fourier Transform of $g(t)$ and $h(t)$ be $G(f)$ and $H(f)$ for all the equations listed below.

Linearity of Fourier transform

The Fourier Transform is a linear transform. The Fourier Transform of any linear combination of $g(t)$ and $h(t)$ can be found:

$$\mathcal{F}\{c_1 g(t) + c_2 h(t)\} = c_1 G(f) + c_2 H(f)$$

where c_1 and c_2 are any constants (real or complex numbers).

Shifts property of the Fourier transform

[6] https://download.ni.com/evaluation/pxi/Understanding%20FFTs%20and%20Windowing.pdf
[7] http://www.thefouriertransform.com/transform/properties.php

The Fourier Transform of $g(t)$ with time shift is:

$\mathcal{F}\{g(t-a)\} = e^{-i2\pi fa}G(f)$

where a is a real number.

Scaling property of the Fourier transform

When the function $g(t)$ is scaled in time by a constant c, the resultant Fourier Transform will be:

$\mathcal{F}\{g(ct)\} = \frac{G(\frac{f}{c})}{|c|}$

Derivative property of the Fourier transform (differentiation)

The Fourier Transform of the derivative of g(t) is given by:

$\mathcal{F}\{\frac{dg(t)}{dt}\} = i2\pi f \cdot G(f)$

Convolution property of the Fourier transform

The convolution of two functions in time is defined as:

$g(t) * h(t) = \int_{-\infty}^{\infty} g(\tau)h(t-\tau)d\tau$

The Fourier Transform of the convolution of $g(t)$ and $h(t)$ is given by:

$\mathcal{F}\{g(t) * h(t)\} = G(f)H(f)$

Modulation property of the Fourier transform

If two functions $g(t)$ and $f(t)$ are multiplied in time, the Fourier Transform of the product is:

$\mathcal{F}\{g(t)h(t)\} = G(f) * H(f)$

2.4.2 Introduction to wavelet transform

The wavelet transform differs from the Fourier transform in that it approximates a signal using a sum of short waves called wavelet.

There are an infinite number of different wavelets which are characterized by compact support. Another characteristic of a wavelet is that the average of the wavelet itself must be zero [8].

A wavelet transform multiplies the signal by a wavelet analyzing function. It outputs a 2 by 2 matrix of coefficients which are identified by the scale and translation:

[8] https://paos.colorado.edu/research/wavelets/wavelet2.html

$X(a,b) = \int_{-\infty}^{\infty} x(t)\Phi^*_{a,b}(t)dt$

The disadvantages of the DISCRETE wavelet transform (DWT) include shift sensitivity, poor directionality, and lack of phase information (Fernandes et al. (2003)).

2.4.3 Introduction to compressed sensing

Compressed sensing can be used to reconstruct a signal when it has been sampled significantly below the Nyquist rate. [9] There has been varying degrees of success in accelerating MRI acquisition Magnetic resonance imaging (MRI) with this technology (Jaspan et al. (2015)).

One of the conditions required for possible recovery is sparsity. The signal is required to be sparse in some domains [10].

[9] https://www.sciencedirect.com/topics/computer-science/compressed-sensing
[10] https://www.sciencedirect.com/topics/computer-science/compressed-sensing

Chapter 3

Methodology

3.1 Categorize uses of provenance from past literature

An initial observation of work from Section 2.3 shows that there is no major difference between the use of provenance in a stream processing system and that in the transactional system. Thus we plan to return to the basis where provenance usage for the transactional system is more systematically categorized in past literature and from the provenance use cases proposed by the W3C provenance incubator group [1] that stream provenance usage can draw from. After categorizing the provenance usage for transactional system, we will then analyze in more detail how works on stream provenance from Section 2.3 fit into this categorization. This categorization will also be useful for us to come up with a set of reasonable provenance questions for the streaming use case gathered later.

This is later shown in Chapter 4. This will partially answer RQ1.

3.2 Understand the provenance need for real streaming use case

To start with, the appropriate streaming use case should be gathered. We expect the source code of the streaming application to be fully available. We've collected a real-world, respiratory use case that calculates respiration rate using a small, lightweight respiration sensor from an NIHR i4i project titled "Continuous respiration monitoring using a novel, wearable capaciflector sensor for early detection of distress, enabling quicker intervention for improved patient outcomes" (grant number: NIHR202107) where Professor Neil M White is the principal investigator. This use

[1] https://www.w3.org/2005/Incubator/prov/wiki/W3C_Provenance_Incubator_Group_Wiki

case is developed by Neil M White and Omar Emara deriving from the work of White et al. (2017). The provenance need for the use case can be identified with the following methods:

1)Literature review

The provenance need of the application can be drawn from studying literature that deals with similar issues around the provenance need for the healthcare domain.

Whittemore et al. (2014) pointed out the importance of identifying the purpose and assessing the quality of the studies in the literature review.

Our purpose here is to specifically focus on the provenance need within the health care context of various work. We intend to include works that contain a more realistic health context. We locate the work of Wang et al. (2007) as the early stage where the need for provenance for the medical stream within the healthcare context was raised and trace backwards and forward to the more current age.

2)Interviewing and seeking expert opinions

The provenance need can be derived by interviewing and seeking expert opinions (Cheney et al. (2018)). As with Cheney et al. (2018) who identified provenance use cases through interviewing individuals from related sectors, I list the experts and the topics for discussion as follows.

The expert I'm going to seek opinions from is Professor Neil M White who is the primary investigator of the use case and has associations with the clinician and technician. The discussion I'm going to initiate revolves around the provenance needs of various stakeholders such as the clinician and technician during his interaction with them. Another expert to seek opinions from is Professor Age Chapman who is well-versed in provenance research. I'm going to seek her opinions on the general provenance needs for the use case.

These will be conducted by introducing the related background knowledge through presentation or talking verbally, initiating discussion on provenance needs for the use case, and collecting opinions from both experts.

For professor White specially, provenance needs can be drawn from experience during his interaction with clinicians at University Hospital Southampton.

3)Studying provenance question classification

We are going to come up with the provenance usage classification as described in Section 3.1. The provenance need of the application can be identified by studying and analyzing its applicability to the application.

By working through a real use case, we will get a better understanding of the types of provenance questions users are interested in answering from a streaming application.

This is later shown in Chapter 6. This can also answer RQ1.

3.3 Design a new approach to post-process streaming data

We will create a theoretical model for streaming data reduction based on the Fourier transform. We will analyze the model to get an understanding of the situations when the technique can reduce the size of streaming data.

Then we will create implementation details for embedding the model in the context of the streaming environment. The stream used in our streaming environment will be as generic as described in Section 2.1.1. The window used will be the time-based, tumbling window using event time. We plan to use the works of Wang et al. (2007) and Huq (2013) to come up with the original fine-grained provenance tracking component added upon the streaming environment to be compared with the provenance approach that has the same 'backbone' but uses reduced streaming data record created by FT. We will analyze the input and output of different operators to identify the stream that can be applied with the FT technique. Finally, we will show how to retrieve contributing data using the original streaming data and the reduced streaming data record.

The analysis of the model and utility of the reduced streaming data for provenance retrieval will provide the answer to RQ2. (This is later shown in Chapter 7).

3.4 Test the goodness of the approach

The respiration sensor data we have at hand was collected from one of the volunteers. To get the data, a relaxation oscillator (containing a capacitive reflector used as a respiration sensor) is placed on the chest of the volunteer (there will be 4 relaxation oscillators placed on both sides and chests of the volunteer to get 4 channels of respiration data). A digital signal is produced after some breathing cycles. The digital signal is then fed into LabVIEW through the NI driver. The LabVIEW software processes the digital signal and produces respiration sensor data (White and Emara (2019)). The respiration data is not patient data, thus no ethical approval is required. The window used is the time-based, tumbling window using event time as described in Section 3.3. The original respiration rate processor is a MATLAB App. We plan to extract codes from MATLAB App to create MATLAB scripts that correspond to different stream use cases as listed in Chapter 5 in order to make it easier to compare

provenance retrieval using original streaming data with provenance retrieval using reduced streaming data. We are using MATLAB here even though it is not specially designed for stream processing for several reasons. Firstly, the focus of this work is on the post-processing of the streaming data instead of dealing with issues around real-time processing. Secondly, since the original respiration rate processor is a MATLAB App, we would like to ensure software coherence.

Since the scripts we have will be pretty much applications. We plan to capture intermediate streams for later provenance retrieval according to application modification from Section 2.2.3.2 and from memory from Section 2.2.3.3. We will first study the topology of the stream processing chain and denote the stream and operator with the stream ID number and operator ID number. Huq (2013) extract stream processing chain topology (what he called workflow provenance) for Python programs automatically. A similar approach to extract stream processing chain topology for MATLAB scripts can be achieved by using YesWorkflow tool (McPhillips et al. (2015b); McPhillips et al. (2015a)). We plan to use it to visualize the stream processing topology and denote the stream and operator with the stream ID number and operator ID number. Then we will code the stream processing topology into the MATLAB scripts. The operator metadata such as operator ID, operator name, and input stream ID is then stored in the excel table for later provenance retrieval. To get fine-grained provenance using original streaming data, we will add the operator ID to each intermediate stream that comes out of a functor and store intermediate streams to enable contributing data retrieval later.

To post-process the streams stored using FT, firstly we will make sure that the segment of the source stream is eligible to be applied with FT. Then we will study the topology of the stream processing chain to decide on streams that can be applied with FT. After that, we apply the FT on the eligible streams, observe their spectrums, and store information about the major spectral line(s).

3.4.1 Metrics

To test the goodness of provenance approach using reduced streaming data with FT in comparison with the fine-grained provenance approach using original streaming data, we come up with a set of metrics.

We will use the storage metric to compare the storage consumption for the streaming data and reduced streaming data using FT.

Storage. The storage for the stream segment(s) contains the timestamp, data content, and operator ID(s) attached (if there is any) for the stream segment(s) in discussion. The storage for reduced stream segment(s) using FT contains the start time, sampling,

end time, frequency bins and their corresponding magnitudes, phases, offsets, and operator ID(s) attached (if there is any) for the stream segment(s) in discussion.

We will use the usability of the streaming data to answer provenance questions as the metric for evaluating the utility of the streaming data (this metric is adapted from Chapman et al. (2021)). Instead of reviewing whether the provenance questions can be resolved by the provenance captured, our focus is on reviewing whether the provenance questions can be resolved using the original streaming data and the reduced streaming data since the provenance retrieval technique we use will be the same which will be achieved utilising streams stored and the table containing operator metadata).

Provenance questions answerability. The provenance questions answerbility reflects the utility of the streaming data or the reduced streaming data in answering provenance questions.

With the storage metric and the usability of streaming data to answer provenance questions, RQ2 can be answered.

To evaluate the performance impact of using FT on streaming data for provenance retrieval as stated in RQ 3, we come up with post-processing time and query time to answer provenance questions as metrics.

Since FT is deployed after streams are stored, we would like to know the time takes for that implementation. We plan to run the experiment 30 times to calculate the average time takes for that implementation.

Post-processing time. The post-processing time represents the average time to apply FFT on stream segment(s) and time to store information about the FT record(s).

We would also like to know the effect of using the reduced stream on query time. We plan to study the time differences between queries unitising original streaming data and queries utilising reduced streaming data. We are going to run each category of query for 30 times and use Wilcoxon Rank Sum Test to study if there is a statistically significant time difference between the two categories of queries.

Query time. Query time represents the time taken to return the result for the provenance question.

The experiments will be performed on a laptop, with an Intel i5-1135G7 @ 2.40GHz CPU and 16 GB of memory, running Windows 10. The version of the MATLAB software is R2020b (This is later shown in Chapter 8).

Chapter 4

Provenance questions classification

We summarize a number of provenance questions (PQs) in real-life scenarios in Section 4.1 to inspire provenance needs for later streaming use cases. After that, example provenance graphs are used to illustrate how they can be answered in Section 4.2. Section 4.3 maps stream provenance usage to PQ classification. Section 4.4 summarizes provenance usage.

4.1 Provenance questions classification

We summarize provenance questions in real-life scenarios in Table 4.1. According to the structure of provenance graphs, these questions can be divided into two categories, namely single graph provenance questions (questions that can be answered using one provenance graph) and multiple graphs provenance questions (questions that require multiple provenance graphs). Single graph provenance questions can be split into two types of questions that navigate either the upstream or the downstream of the graph. We also put it in the last column in Table 4.1 to denote the type of query the provenance question belongs to.

4.2 Explanations with provenance graphs

We use a few simplified example provenance graphs to intuitively explain how the above provenance questions can be answered. As shown in the lower right corner of Figure 4.1, the yellow circle represents the entity, the blue rectangle represents the activity, and the orange pentagon represents the agent.

1. Graph for booking a plane ticket: (in answering Q1, Q3, Q4, Q5)

ID	Type	Question	Reference	Graph classification	Query type
Q1	Justification	Is the decision justified?	Naja and Moreau (2010)	Single graph upstream	How-provenance
Q2	Compliance	Is the production procedure of a product compliant with the policy?	Groth (2010)	Single graph upstream	How-provenance
Q3	Trust	Can I trust this data?	Hartig (2010)	Single graph upstream	How-provenance
Q4	Debugging	What factors led to a problematic result?	Chapman et al. (2021)	Single graph upstream	How-provenance
Q5	Responsibility	Who is responsible for a certain outcome?	Zerva et al. (2013)	Single graph upstream	-
Q6	Taint analysis	What is the effect of the tainted data?	Allen et al. (2011)	Single graph downstream	-
Q7	Reproducibility	What causes the differences of two or more experimental results?	McCusker (2009)	Multiple graphs	-

TABLE 4.1: List of provenance questions

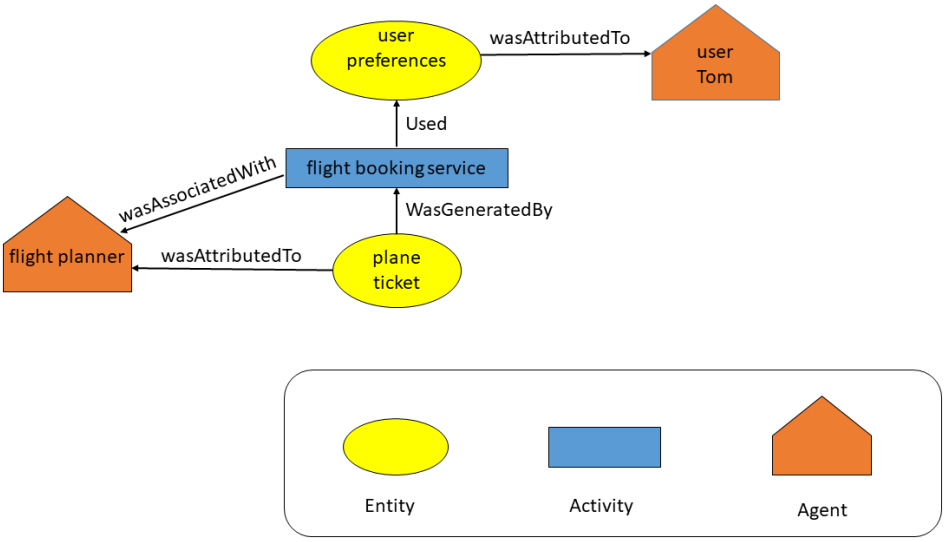

FIGURE 4.1: Graph for booking a plane ticket

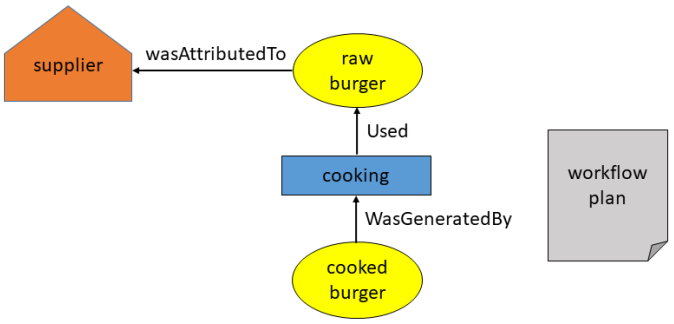

FIGURE 4.2: Graph for checking compliance of food preparation

In Figure 4.1, a user Tom interacts with a flight booking service by inputting his preferences (e.g., 2nd class seat etc) to it. The service generated a plan ticket accordingly. The flight booking service is provided by a flight planner.

Suppose that Tom is suspicious about a high-cost plane ticket being booked. Having the provenance of the event, he can trace backwards to justify the outcome (Q1). If he is assured that he did not put in the wrong preferences and that the flight booking service was working properly, then he decides he can trust the outcome (Q3). Whereas if he is sure that there is something wrong with this plane ticket, he can trace back to find out the bug (Q4). Suppose that he spots that the flight booking service automatically does the insurance ticking, he can find out the agent who is responsible for the service for the flight planner and make a complaint (Q5) (Zerva et al. (2013)).

2. Graph for checking compliance of food preparation (in answering Q2)
In Figure 4.2, a restaurant wants to ensure its food quality. To do that, they make up a workflow plan detailed for food production beforehand. To check compliance, they trace back the upstream of a food product (e.g., to see if the cooking temperature exceeds 75 °C or if the raw material came from a trusted supplier as documented in the workflow plan) (Markovic et al. (2016)).

3. Graph for analysing the consequences of tainted data: (in answering Q6)
In Figure 4.3, an organization released a dataset. An analyst Alice queried the data and got a result. She used the result when writing a report.

Suppose that the organization discovers that an attacker has subverted the dataset, this dataset will be annotated as ”tainted”. Having the downstream record of the

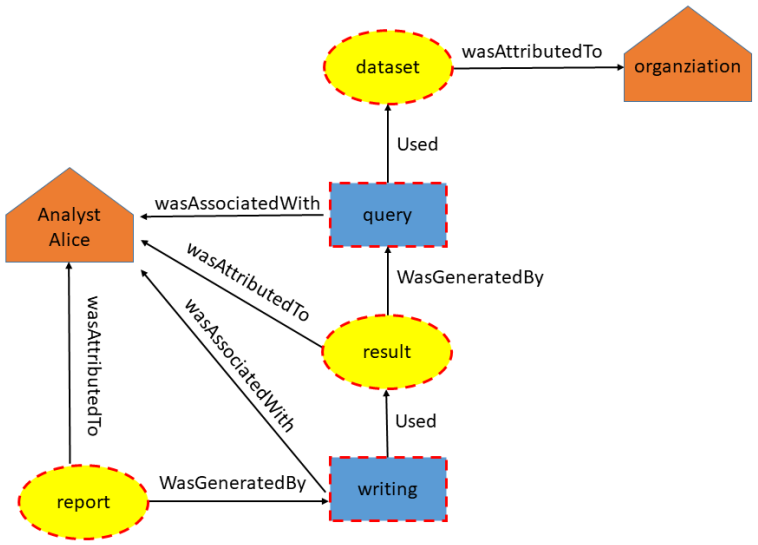

FIGURE 4.3: Graph for analysing the consequences of tainted data

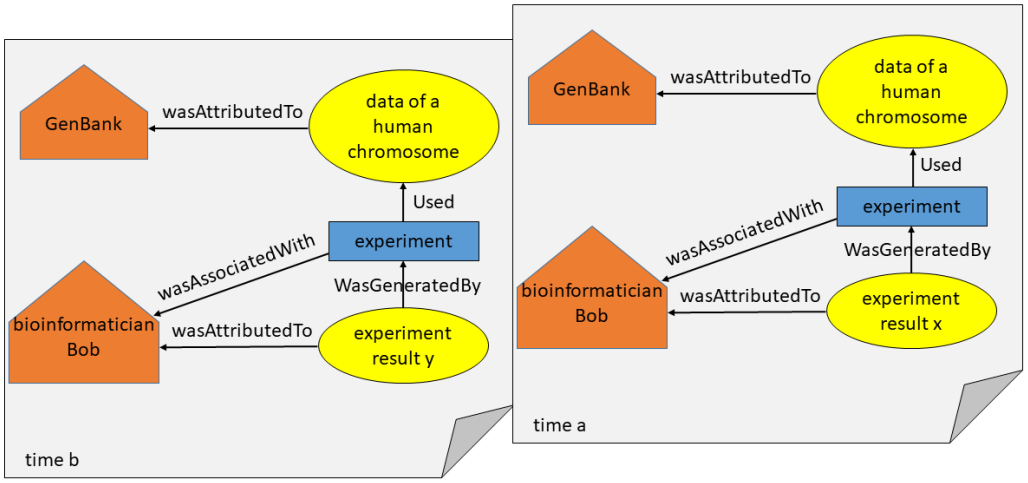

FIGURE 4.4: Graph for comparing result differences

tainted dataset, all the entities and activities that rely upon it will also be marked as "tainted" (nodes circled by the red dash line). The organization can then alert Alice about the harmful data and inform her to take further action.

4. Graph for comparing result differences: (in answering Q7)
In Figure 4.4, a bioinformatician Bob downloaded data of a human chromosome from GenBank and performed an experiment at time a. Later at time b, he performed the same experiment on data of the same chromosome, again downloaded from GenBank.

Work	PQ classification	Type
Vijayakumar and Plale (2006)	Q4	Debugging
Wang et al. (2007)	Q1, Q4	Justification, debugging
Huq (2013)	Q1, Q4	Justification, debugging
Glavic et al. (2014)	Q1	Justification
Pouchard et al. (2018)	Q1	Justification
Erlandsson and Gordani Shahri (2021)	Q1	Justification

TABLE 4.2: Mapping stream provenance use to PQ classification

Suppose Bob notices a difference between two experiment results, he can consult two provenance graphs to see if the difference was caused by the experimental process or change in configuration, or by the chromosome data being different. In this way, new insight or knowledge may be gained (Miles et al. (2007)).

4.3 Mapping stream provenance usage to PQ classification

A table that maps the use of provenance in a stream processing system to provenance questions classification from Table 4.1 is presented in Table 4.2 after reexamining works from Section 2.3.

We can see that Vijayakumar and Plale (2006) target at the provenance of stream processing in the physical environment domain. The provenance need is motivated by foreseeing the problems such as stream rate changes that may happen for a real-world application of meteorology forecasting. This provenance usage belongs to Q4 debugging.

Wang et al. (2007) deal with provenance within Century, an extensible framework for analysis of large numbers of remote health monitoring data streams. The provenance need comes from the medical and legal requirements. Various stakeholders such as doctors and family members should be able to inspect Century's internal "information flow", to either manually validate the processing logic or to perform fault-diagnosis when anomalies occur. This provenance usage belongs to Q1 justification and Q4 debugging.

Huq (2013) comes up with some artificial and simplified workflows from a real-time environmental experiment project RECORD that studies how river restoration affects water quality. The provenance need comes from foreseeing when an abnormal or unexpected value exists in the outcome. The provenance information can help debug the outcome as well as validate the model. This provenance usage belongs to Q1 justification and Q4 debugging.

Glavic et al. (2014) present an example application that detects overheating. The provenance need comes from the user who wants to understand which sensor readings caused an ”overheating” event. This provenance usage belongs to Q1 justification.

Pouchard et al. (2018) present a protein structure propagation workflow based on NWChemEx. The provenance need comes from the user to help interpret the result. This provenance usage belongs to Q1 justification.

Erlandsson and Gordani Shahri (2021) present three sample applications: one for broken-down cars query, one for car accidents query, and the other for blackout detection query. The provenance need comes from the need of the user to verify the origin and to understand a certain output. This provenance usage belongs to Q1 justification.

Sansrimahachai (2012) deals with a form of ”why provenance” which is not the focus of this thesis. Misra et al. (2008) envision CMIR on top of TVC-based provenance. The provenance need from their work is essentially the same as Wang et al. (2007). Rather than dealing with provenance of stream directly, Chen and Plale (2015) and Suriarachchi et al. (2018) have a different focus on doing processing on top of the provenance collected. Thus we exclude the remaining four pieces of work from our discussion on stream provenance usage.

4.4 Summary of provenance usage

We summarize the above provenance questions in Figure 4.5. They generally fall into the three categories we abstract (the circles in green).

Knowing the usage of provenance beforehand provides general guidance as to what to capture and at what granularity to capture. To be a bit more specific, the key things to capture are: 1)entity. This includes the static element ranging from data to file; 2)activity. This includes dynamic manipulation ranging from operator, code, and function to application. These two things can already depict a picture of what has happened. The context (e.g., some environmental variables) adds to the richness of the picture. The agent information provides the object for later consultancy or investigation. These two are relatively easier to capture. Intuitively capturing more wide breadth and coarse-grained provenance will be more suitable for application targeting at understanding and sense-making. Whereas fixing mistakes requires more detailed, fine-grained provenance. Gaining new knowledge and insight is more of a by-product in the process of fixing mistakes after some thinking process.

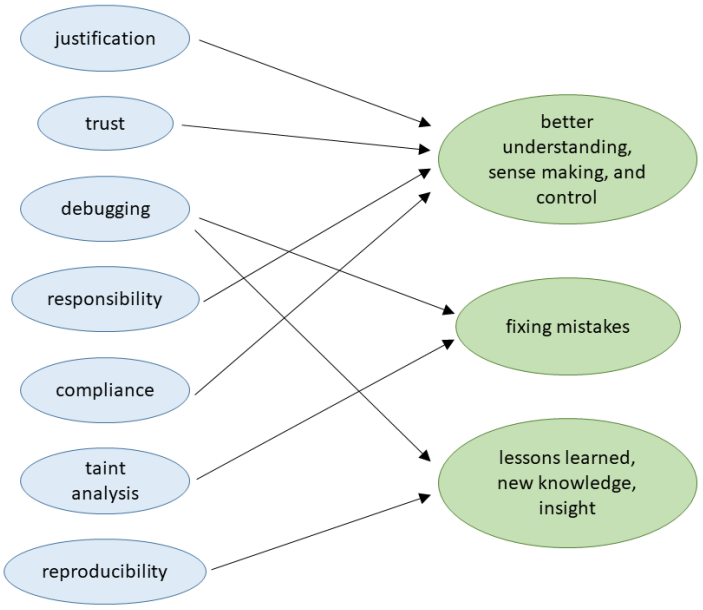

FIGURE 4.5: Provenance usage summary

Chapter 5

Respiratory use case

In this chapter, we present a real-world, respiratory use case. We describe the details of streams and processing of the use case in Section 5.1.

A Cardiopulmonary Exercise Test (CPET) is a non-invasive method used to assess the performance of the heart and lungs at rest and during exercise [1]. It contains heart-rate monitor and respiration monitor. Respiration rate is important in diagnosing diseases such as sepsis (Loughlin et al. (2018)) and COVID-19 (Stojanović et al. (2020)). In this use case, we focus on the respiration monitor developed by a small, lightweight respiration sensor. This case study derives from the work of White et al. (2017).

The capaciflector (capacitive reflector) used as a respiration sensor was originally developed by NASA as a capacitive proximity sensor for collision detection in robots. It is an active element of a relaxation oscillator. As the chest expands and contracts during the breathing cycle, the relaxation oscillator produces a variable frequency square wave due to the change in capacitance (White et al. (2017); Hayward et al. (2022)). This digital signal is then fed into LabVIEW through a NI driver. The LabVIEW software then calculates the reciprocal of the time period of the square wave which is proportional to the changing capacitance. This data is then fed into MATLAB. The MATLAB program filters the data, finds the peaks, and calculates the respiration rate.

Figure 5.1 shows the workflow diagram for the above process.

[1] http://www.geh.nhs.uk/directory-of-services/specialties-and-services/c/cardio-respiratory-unit-cru/cardiopulmonary-exercise-testing-cpet/

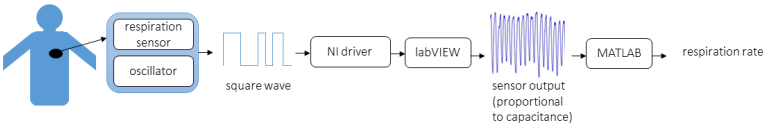

FIGURE 5.1: Workflow diagram

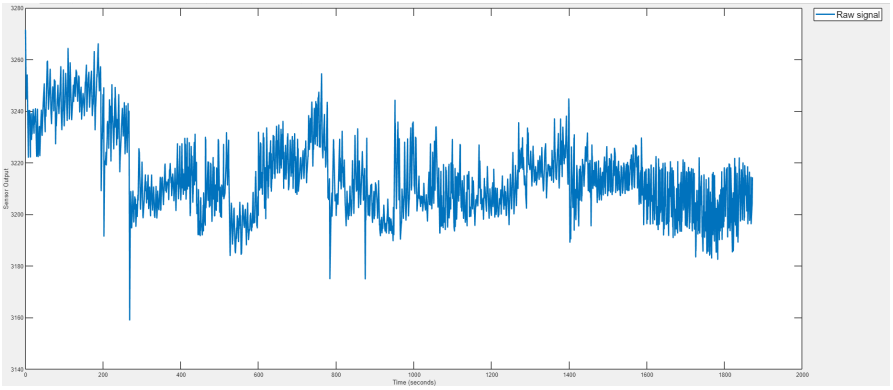

FIGURE 5.2: Raw streaming data

5.1 Details of streams and processing

5.1.1 Stream captured from capaciflector

The output of the sensor was sampled at 0.1s. Due to the configuration of the capaciflector and the relaxation oscillator, it gives back a value of around 3200 (proportional to the value of capacitance) at every 0.1s. These are the very raw, unfiltered data (White and Emara (2019)). The streaming data is numerical with a constant sampling interval.

Figure 5.2 shows the picture of the raw streaming data.

5.1.2 Initial transformation

The raw data is very noisy. Thus the built-in filtfilt function is used in MATLAB to filter the data. The filtfilt function performs zero-phase digital filtering by processing

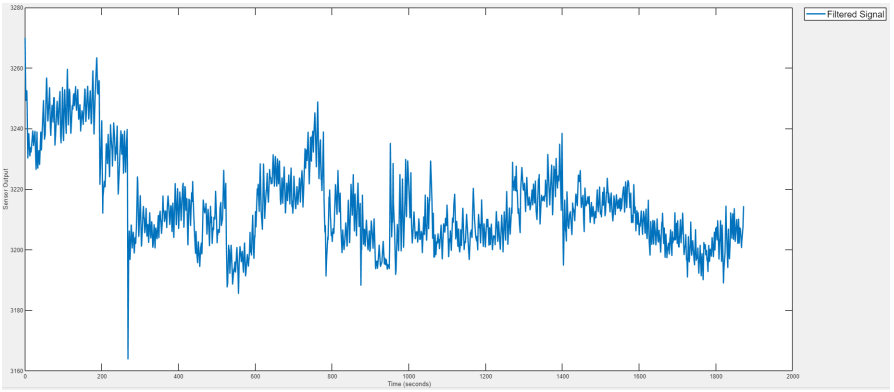

FIGURE 5.3: Filtered streaming data

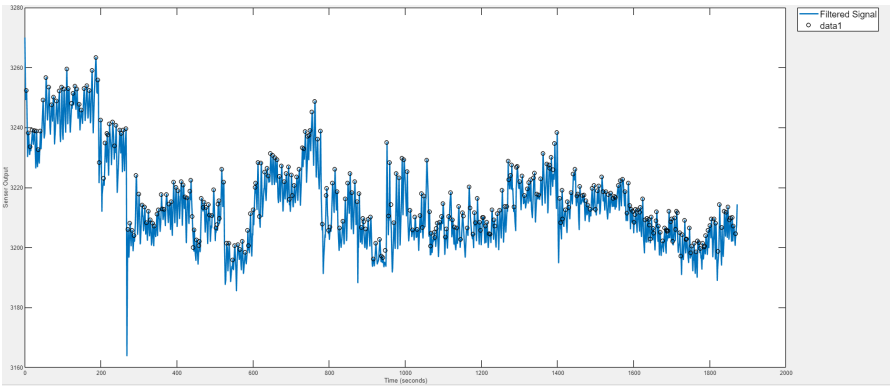

FIGURE 5.4: Peaks found over the filtered streaming data

the input data in both the forward and reverse directions. This gives us the filtered streaming data (White and Emara (2019)). The filtered streaming data is numerical with a constant sampling interval.

Figure 5.3 shows the picture of the filtered streaming data.

5.1.3 Finding the peaks

Over the filtered streaming data, the built-in findpeaks function in MATLAB is used (White and Emara (2019)). We use findpeaks function consisting of 'MinPeakProminence' and a real scalar as name-value Pair Arguments. This gives us the height and time value of each peak that has a relative importance of at least 'MinPeakProminence'. The height value is used to mark the peak on the filtered stream and the time value is used to calculate the respiration rate (also called BPM, breath per minute) later on. The peak stream is numerical with inconstant sampling intervals.

Figure 5.4 shows the peaks found over the filtered streaming data.

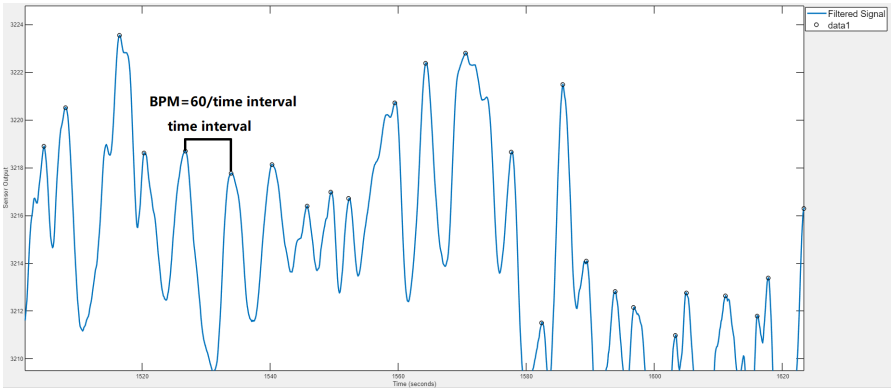

FIGURE 5.5: How BPM was calculated

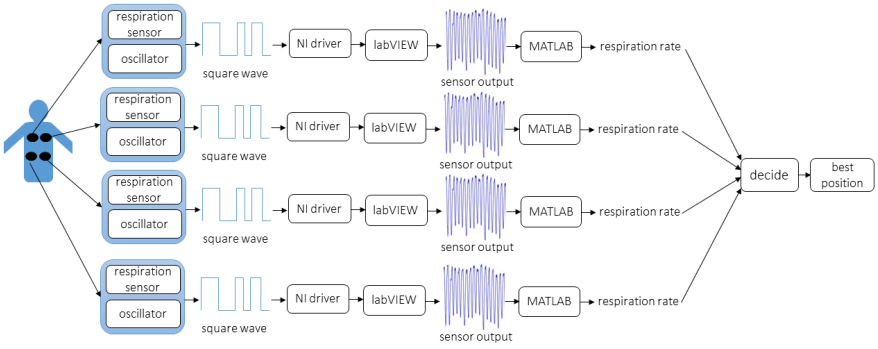

FIGURE 5.6: Workflow diagram for finding best position

5.1.4 Calculating respiration rate

The time interval between each peak reflects the time taken for one breath. A minute divided by the time interval gives us the value of the respiration rate (White and Emara (2019)). Figure 5.5 shows the time interval between peaks and how the respiration rate was calculated.

5.1.5 Finding the best position for a sensor

In order to find the best place for measuring respiration rate, sensors are placed on both sides and chests for experimentation. The output of the 4 channels will be fed into a human deciding process (meaning the expert technician picks up a position as the best according to his intuition and experience) for choosing the best position for the sensor.

Figure 5.6 shows the workflow diagram for the above process.

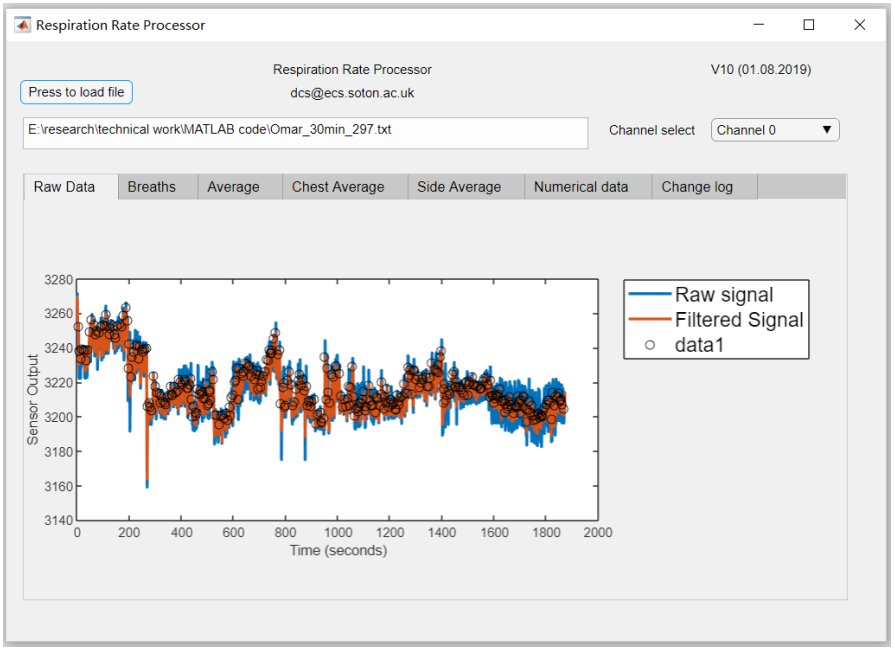

FIGURE 5.7: Respiration rate processor GUI

5.1.6 Respiration rate processor

The respiration rate processor that deals with the above processing is a MATLAB App developed by White and Emara (2019). The source code is available. The GUI view of the App is shown in Figure 5.7. It can be used to depict sensor output and BPM cross time for 4 different channels.

A user presses the 'Press to load file' button on the top left corner of the interface and upload the file containing the raw sensor output data from 4 channels.

The 'Channel select' drop-down menu on the top right corner allows a user to select the channel he wishes to observe.

The 'Raw data' tab displays the raw signal, filtered signal, and the peaks found of the selected channel.

The 'Breaths' tab shows the respiration rate calculated using the time interval between peaks, smoothed respiration rate calculated using MATLAB built-in filtfilt, and the respiration rate calculated from 7 peaks as shown in Figure 5.8.

The 'Average' tab shows the average raw signals of the 4 channels, the average filtered signals of the 4 channels and peaks found over the average filtered signals as shown in Figure 5.9.

The 'Chest Average' tab shows the average raw signals from the chests and the average filtered signals of the chests and peaks found over the average filtered signals as shown in Figure 5.10

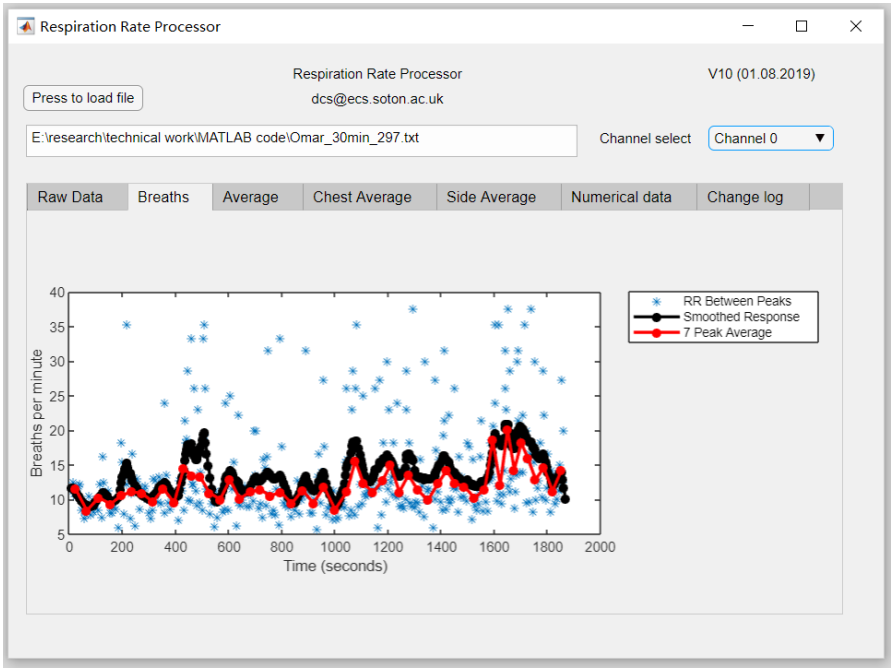

FIGURE 5.8: Breaths tab of GUI

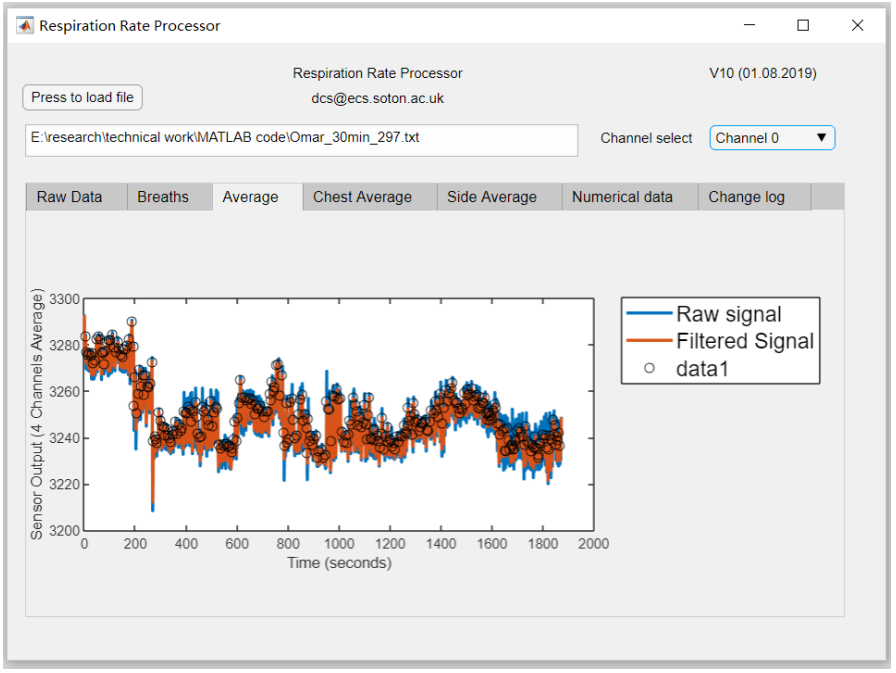

FIGURE 5.9: Average tab of GUI

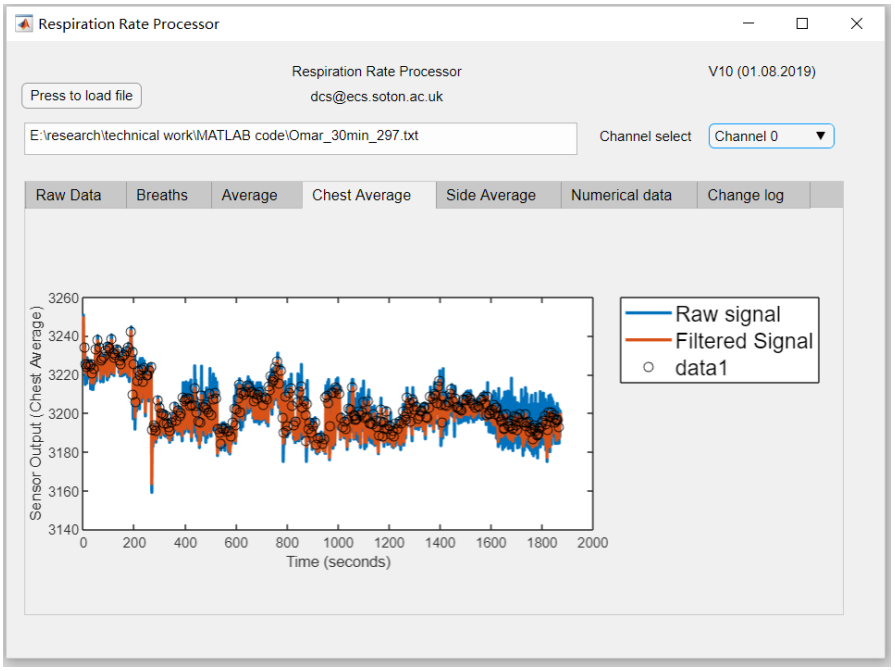

FIGURE 5.10: Chest Average tab of GUI

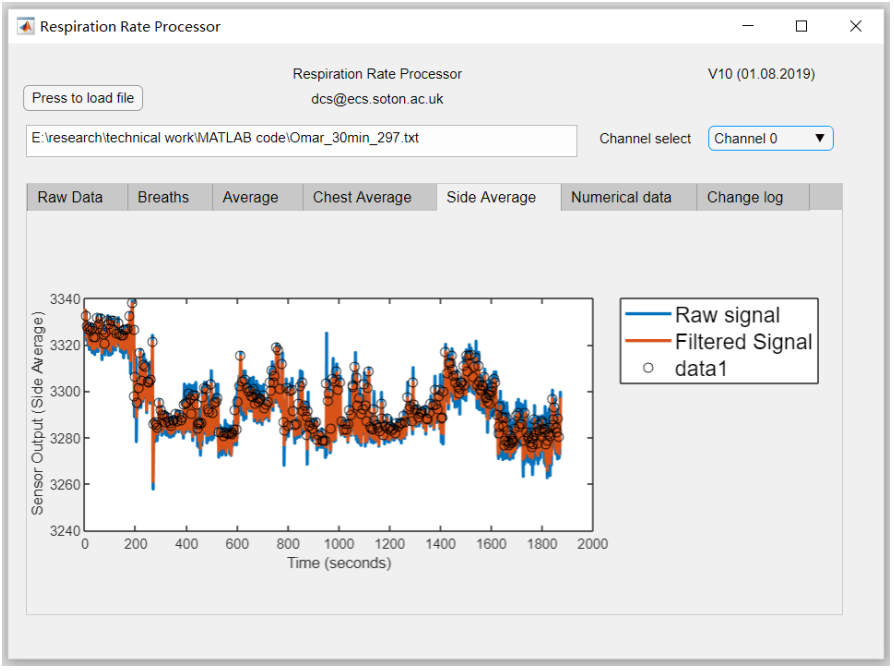

FIGURE 5.11: Side Average tab of GUI

The 'Side Average' tab shows the average raw signals from the sides and the average filtered signals of the sides and peaks found over the average filtered signals as shown in Figure 5.11.

The 'Numerical data' tab shows a list of peaks found and respiration rates calculated using these peaks from a certain channel as shown in Figure 5.12. It also displays the average respiration rate, the peaks number of a selected channel, the peaks number of

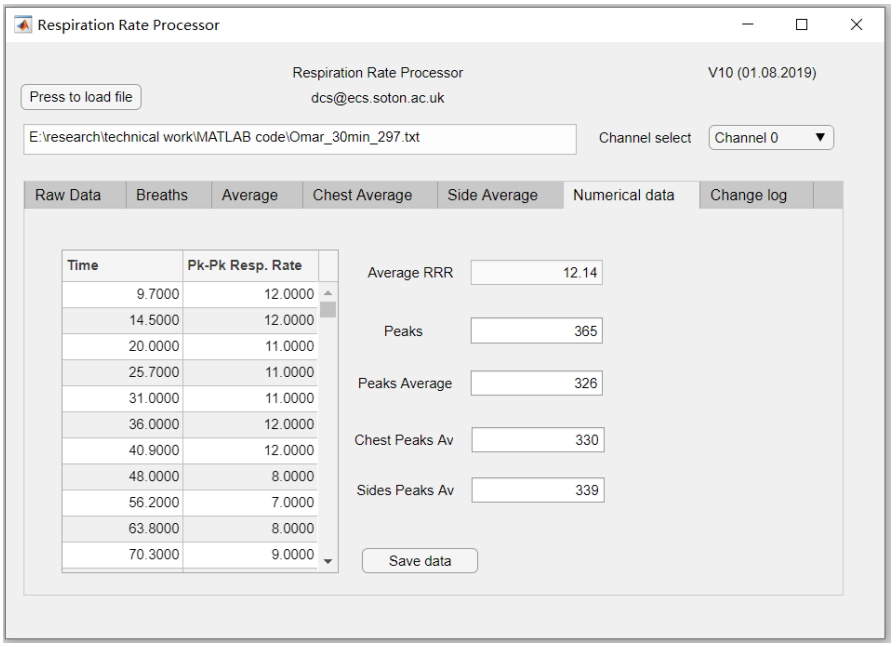

FIGURE 5.12: Numerical data tab of GUI

the average signal of 4 channels, the peaks number of the average chests signal and the peaks number of the average sides signal.

The 'Change log' tab shows the changes made to the GUI during development.

Chapter 6

Understand the provenance needs for the respiratory use case

In this chapter, literature review that deals with provenance need within the health care context is presented in Section 6.1. In Section 6.2 presents expert opinions sought on provenance needs for the respiratory use case. In Section 6.3, I synthesize provenance needs through literature review, interviewing and seeking expert opinion or studying provenance question classification.

6.1 Literature review

Kifor et al. (2006) and Deora et al. (2006) raised the need to document the processes that leads to the patient's health condition to validate a particular outcome or decision.

Wang et al. (2007) proposed the need that various stakeholders such as doctors, family members should be able to inspect the internal "information flow" to justify the processing logic in their Century healthcare IT project that manages medical streams for potential medical and legal purposes.

Dogan (2013) stated that Hospital Information Systems is one domain that motivates provenance. To illustrate this, he raised specific examples such as Health Care Portability and Accountability Act (HIPAA) which makes it mandatory to record access and medical records change history.

Sembay et al. (2021) proposed the provenance approach to improve blood donation quality using real data extracted from reports provided by a Brazilian hemotherapy centre.

We can see from the above literature that though not all work targets at provenance need for medical streams specifically (some targets at the medical outcome as in

medical records or blood products), there is this general provenance need for regulatory and validation purposes within the healthcare context.

6.2 Interviewing and seeking expert opinions

Initially, a PowerPoint on provenance concept and related background knowledge was presented to Professor Neil M White to seek for provenance research opportunity for the respiratory use case, he commented "I think it is a good fit".

At the early stage of identifying provenance needs, discussion on provenance needs for the use case was initiated and advice were sought from Professor Age Chapman who has expertise in provenance research. General provenance needs which include understanding how a respiration rate came into being and debugging need if there is an unusual respiration rate were identified.

Then through his interaction with clinicians at University Hospital Southampton, it is communicated by Professor Neil M White that during calibration of the respiration sensor, we have data coming from the CPET lab (the hospital setting) that shows that BPM as large as 80 when exercising was found. Even though the algorithm used to calculate the respiration rate from the CPET lab is unknown to us, it shows that there is a potential that we might end up with an inaccurate respiration rate in the real world. This provides us with potential provenance use in debugging to experiment with in our lightweight respiration sensor setting.

During the COVID-19 epidemic, it is also communicated through Professor Neil M White that clinicians would like to know if the resulting respiration rate is reliable while working with them.

6.3 Provenance needs

In order to identify the right data for collection, it is important to pre-define questions that need to be answered via provenance. I sort out and synthesize several provenance needs as follows through the aforementioned methods.

6.3.1 Provenance need 1: how did this respiration rate get created?

We can see from the literature in Section 6.1 that Wang et al. (2007) propose the need that doctors should be able to inspect the internal "information flow" to justify the processing logic in their Century healthcare IT project. Kifor et al. (2006) and Deora

et al. (2006) have also raised the need to document the processes that lead to the patient's health condition.

In Section 6.2, the expert opinion sought from Professor Neil M White communicates clinicians' need to justify the respiration rate. Adding to this is the expert opinion sought from Professor Age Chapman of the provenance need to understand how respiration rate came into being.

This provenance need to understand and justify how a respiration rate gets created also corresponds to provenance use for justification as presented in Table 4.1. This is important for a non-technician (e.g., a doctor) to get a sense of how a respiration rate came into being and whether it is justified to him.

The doctors will be expected to know the stream containing contributing data and processes without further investigating into it. This is a high-level understanding question. We anticipate the granularity to be low. We thus have a provenance question (PQ) as follows.

PQ1=what were the streams and processes used to calculate the respiration rate?

6.3.2 Provenance need 2: why did we get an unusual respiration rate?

In Section 6.2, the expert experience of getting inaccurate respiration rate provides us material use case to experiment with in our lightweight respiration sensor setting from a technician's perspective. The expert opinion from Professor Age Chapman complements the provenance need for debugging for the respiration rate.

This provenance need to debug corresponds to the provenance usage for debugging as presented in Table 4.1.

As this is a debugging need, we anticipate the granularity to be high. We list a couple of questions that a debugger would ask as follows. The debugger will be able to see the specific data, processes, and picture depicting the stream.

PQ2=what data constituted the abnormal respiration rate?

PQ3=what processes actually touched the data?

PQ4=what did the constituent filtered stream look like? (the reason why we ask this question is because the shape of the stream might provide expert technicians with material to work out what the problem is according to their experience and domain knowledge. We are asking for the filtered stream because it is good for the observing purpose since the the noise of stream has been removed.)

6.3.3 Provenance need 3: why was position X chosen as the placement site?

As described in Section 5.1.5, inputs from four channels were fed into a human deciding process for choosing the best spot for measurement. As this process is largely intuitive and experience-based, people (such as families of a patient as described in the work of Wang et al. (2007)) may question the validity of the position chosen. Why is a certain position chosen but not the other? What streams were used to produce respiration rate? Were they similar to each other at a given time?

This provenance need for validation corresponds to provenance usage for justification as presented in Table 4.1.

This gives birth to the 5th provenance question as follows. We expect the granularity to be high. The pictures depicting filtered streams will be expected to see.

PQ5 = what did the filtered streams at each individual sensor look like at a given time / during a certain time interval?

We also work through the remaining provenance questions from PQ classification to identify possible provenance use cases. Since there is no workflow plan that the respiratory processing needs to follow, we exclude the provenance use for compliance from the provenance use case (Q2). The trust aspect of provenance (Q3) is more subjective, thus our focus here is on the ”checking” of the information flow, namely provenance use for justification (Q1). The provenance use for responsibility (Q5) is trivial, thus we did not create a provenance use case out of it. The provenance use for taint analysis (Q6) deals with the security aspect of a system (Stamatogiannakis et al. (2017); Allen et al. (2010b)) which is not the main provenance concern for the respiratory use case. For the respiratory use case, we did not run the same experiment repeatedly. The experiments come with different configurations such as the sensor being placed at different parts of the body, thus the provenance use for reproducibility (Q7) is not applicable to the respiratory use case.

We map the provenance questions derived from the respiratory use case with provenance question classification from Table 4.1 in Table 6.1.

PQs from the use case	Query	PQ classification	Type
PQ1	what were the data and processes used to calculate the respiration rate?	Q1	Justification
PQ2	what data constituted the abnormal respiration rate?	Q4	Debugging
PQ3	what processes actually touched the data?	Q4	Debugging
PQ4	what did the constituent stream look like?	Q4	Debugging
PQ5	what did the streams at each individual sensor look like at a given time / during a certain time interval?	Q1	Justification

TABLE 6.1: Mapping provenance questions broken down from provenance needs to provenance question classification

Chapter 7

Reducing streaming data storage through the integration of FT

In this chapter, we present a model for streaming data reduction based on FT in Section 7.1. In Section 7.2, we analyze the model to understand when the technique can reduce the size of streaming data. We create implementation details for implementing the model into the stream processing infrastructure we present in Section 7.3.

7.1 Model for streaming data reduction

7.1.1 Preliminaries

For the FT technique to work, there are several prerequisites that the streams involved need to satisfy: 1)the time interval between records in the stream needs to be constant; 2)the data content of the stream record is numerical. In other words, the type of stream that is eligible to be applied with FT sits at the top left corner of Table 2.1 in Section 2.1.1. As streaming data comes in continuously, we plan to apply the FT techniques on the segment of data that has already sit in the log and are eligible to be applied with FT.

7.1.2 The model

Suppose we have segment of streaming data $A = \{\{t_0, x_0\}, \{t_1, x_1\}, ...\{t_{N-1}, x_{N-1}\}\}$. It contains N stream records. The time offset has been removed so that $t_0 = 0$. The sampling interval $T = t_1 - t_0 = t1$.

We then apply the DFT to this streaming data as described in Section 2.4.1:

$X(w_k) = \sum_{n=0}^{N-1} x_n \cdot e^{-jw_k t_n}$

This gives back N frequency bins with corresponding magnitudes and phases.

$B = \{b_0, b_1, b_2, ...b_{N-1}\}$

where b_0 is the zero frequency bin, b_1 is the first frequency bin, and so on.

These frequency bins can be further rewritten as:

$B = \{\{w_0, a_0, \alpha_0\}, \{w_1, a_1, \alpha_1\}, ...\{w_{N-1}, a_{N-1}, \alpha_{N-1}\}\}$

where $b_k = a_k \cdot e^{i(w_k + \alpha_k)}$, $w_k = \frac{2\pi k}{NT}$.

If the original provenance of the stream could be represented as:

$P = A_1 \leftarrow operator1 \leftarrow A_2 \leftarrow operator2 \leftarrow A_3 \leftarrow operator3 \leftarrow A_4$

where A_4 is the final result.

With the use of the DFT technique, we now have:

$P = B_1 \leftarrow operator1 \leftarrow B_2 \leftarrow operator2 \leftarrow B_3 \leftarrow operator3 \leftarrow A_4$

assuming the source stream is amendable to DFT and all operators produce an intermediate stream that is amendable to the DFT technique and the last output stream won't be altered so that humans can consume it.

7.2 Analysis of the model

It should be noted that it is a must for the stream segment to satisfy the prerequisites listed in Section 7.1.1 to be able to be applied with DFT. But satisfying the prerequisites does not necessarily lead to the successful deployment of this technique in reducing storage for stream. We discuss the reasons as follows.

The streaming data is sampled from a source. This source can be external such as the human body (White et al. (2017)) in the healthcare domain or light (Lindner et al. (2020)) in the natural physics domain. It can also be internal such as the stock exchange stream in the social domain (Guo et al. (2002)).

i) When we have some knowledge about the frequency of the source and the sampling theory is satisfied (see Section 2.4.1) and when the streaming data segment contains an integer number of period(s), some neat spectral line(s) are given back. The DFT can reduce the storage for streaming data.

ii) When we have some knowledge about the frequency of the source and the sampling theory is satisfied, but the streaming data segment does not contain an

integer number of period(s), we are likely to end up with many spectral lines. With an appropriate choice of window (see Section 2.4.1.1), the DFT can usually reduce the storage for streaming data.

iii) We are not clear about whether DFT can reduce the storage of streaming data whose source does not exhibit periodic behaviour or we do not have knowledge about its frequency. Mapping to the real world, there exists work that applies DFT to financial time series data (Guo et al. (2002)).

7.3 Implementation details

7.3.1 Overview of the streaming environment and the original provenance component

The stream is as generic as we describe in Section 2.1.1. Though there are many types of windows and different time notions associated with them (see Section 2.1.2), the window we use here is mostly the time-based window using event time. Event time has several advantages. Using event time, the program semantics is decoupled from the actual serving speed of the source and the processing performance of the system. It also prevents semantically incorrect results due to situations such as delay [1]. The window is the tumbling window.

As presented in Section 2.3, the coarse-grained provenance approach from the work of Vijayakumar and Plale (2006) cannot identify dependency for individual data elements and is not suitable to act as the original provenance approach to be compared against. The work of Misra et al. (2008) is envisioned idea with no follow-up implementation. Sansrimahachai (2012) aims to address a form of "why provenance" but not how-provenance. The focus of the work of Glavic et al. (2014) is on developing the provenance generation technique through operator instrumentation to handle nondeterministic. Both Chen and Plale (2015) and Suriarachchi et al. (2018) deal with on-the-fly provenance information rather than persistent provenance storage for later retrieval. Pouchard et al. (2018) and Erlandsson and Gordani Shahri (2021) focus on enabling provenance collection for a specific time. Among these works, we found the work of Wang et al. (2007) and Huq (2013) to be most suitable to serve as the original provenance approach to be compared with. Both the TVC model (Wang et al. (2007)) and Huq's work (Huq (2013)) feature the fine-grained stream provenance approach. Yet for the clearer illustration purpose of this work, we will collect more metadata about operators like Huq did to enable provenance retrieval later. Huq's work does more detailed documentation of operators compared with TVC work so that it has the ability to retrieve the specific

[1] https://flink.apache.org/news/2015/12/04/Introducing-windows.html

Provenance system	Stream ID required	Timestamp used	Operator ID required	Output stream marked with operator ID
TVC	Yes	event time	Yes	Yes
Huq's	Yes	processing time	Yes	No
this work	Yes	event time	Yes	Yes

TABLE 7.1: System differences between TVC, Huq's work, and this work

contributing data (see Section 2.3). However, we will store these metadata about the operator together with its input stream ID in the table like TVC work for more organized storage and more automatic query. We assume that the developer will be documenting these metadata while developing the stream application.

An input stream is read from sources and stored in the log (the idea of a log comes from Apache Kafka [2]). It contains stream ID, timestamp, and data content.

We also assume that the output stream element will be stored, marked with the operator ID. The output stream contains stream ID, timestamp, data content, and operator ID.

A table that compares system differences among TVC work, Huq's work, and this work is presented in Table 7.1.

By facilitating the operator ID of the output stream, the metadata about the operators, and the ID and timestamp of the input stream element, the contributing data can be retrieved.

An illustration of the architecture of the above stream processing system and provenance tracking component is presented in Figure 7.1. The component circled with the blue dashed line represents the stream processing system. The component circled with the red dashed line represents the provenance tracking component. The component circled with the orange dashed line represents the reduced provenance tracking component that we will present later.

7.3.2 Creating reduced stream record

When we have no knowledge about the sampling interval and the type of the data, an algorithm to decide if the segment of data is eligible to be applied with DFT is presented in Algorithm 1. We assume that the data type in the same log is the same.

Once the streaming data A is checked to be eligible to be applied with the reduction technique, we extract the data sequence from the streaming data segment and apply FFT on it. The length of A is even here.

[2] https://kafka.apache.org/

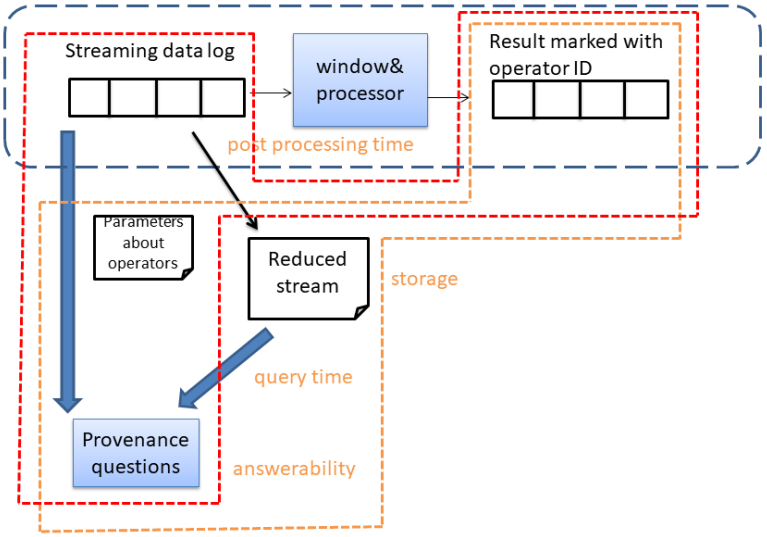

FIGURE 7.1: Illustration of the architecture of the above stream processing system and provenance tracking component

Algorithm 1 Checking if segment of data is eligible to be applied with FT

Require: Segment of streaming data A
Ensure: yes or no
1: $v1 \leftarrow isnumerical(A)$
2: **if** v1=0 **then**
3: print no
4: **end if**
5: $N \leftarrow length(A)$
6: $A' \leftarrow A(2:N)$
7: $interval \leftarrow A'.t - A(1:N-1).t$
8: $v2 \leftarrow max(interval) - min(interval)$
9: **if** v2=0 **then**
10: print yes
11: **else**
12: print no
13: **end if**

An algorithm to create frequency records out of the stream segment is presented as Algorithm 2. First, the DC offset is removed so that there is no disturbing spectral line at bin 0 (line 1-line 2). Because the FFT uses N points to calculate the spectrum, we divide each spectrum line by N in the result. Also As it is impossible to measure frequencies above the Nyquist limit, we maintain and double the spectral values below the Nyquist limit (line 4-line 7). Finally, we plot the spectrum figure using a series of frequencies F and its corresponding magnitudes (line 11).

Algorithm 2 Creating frequency records out of stream segment

Require: Segment of streaming data A
Ensure: A series of frequency F, magnitudes P_1 and phases *phase* for different frequency bins
1: $streamOffset \leftarrow mean(A.data)$
2: $s \leftarrow A.data - streamOffset$
3: $B \leftarrow FFT(s)$
4: $N \leftarrow length(B)$
5: $P2 \leftarrow abs(B/N)$
6: $P1 \leftarrow P2(1 : N/2 + 1)$
7: $P1(2 : end - 1) \leftarrow 2 * P1(2 : end - 1)$
8: $phase \leftarrow angle(B(1 : N/2))$
9: $f_s = 1/(t_2 - t_1)$
10: $F \leftarrow 0 : (f_s/N) : f_s/2 - f_s/N$
11: $plot(F, P1(1 : N/2))$

We then observe spectrum figure and store certain frequency bins that have more dominant magnitude values along with their phases. The algorithm used to store information about the selected frequency components is presented in Algorithm 3. It should be noted that the time sequence participating in reconstructing the stream shall start from 0 as the DFT assumes that the first timestamp starts from 0 so that the DFT can be applied straightly mathematically. This time sequence can be easily achieved by using the length and sampling frequency of the original time sequence. After the stream has been reconstructed, we can then append the original time sequence.

Algorithm 3 Storing information about the selected frequency components

Require: original streaming data A, a collection of selected frequencies f, magnitudes m, and phases p
Ensure: FT record *FTrecord* required to reconstruct the stream
1: $t \leftarrow A.t$
2: $FTrecord.starttime \leftarrow t(1)$
3: $FTrecord.sampling \leftarrow t(2) - t(1)$
4: $FTrecord.endtime \leftarrow t(end)$
5: $FTrecord.streamOffset \leftarrow mean(A.data)$
6: $FTrecord.frequency \leftarrow f$
7: $FTrecord.magnitude \leftarrow m$
8: $FTrecord.phase \leftarrow p$

7.3.3 Applying to real streams

The stream processing infrastructure can be built up using the example operators we list in Table 2.3. We discuss whether DFT can be applied to the input and output of different operators as follows. Note that we are not going to analyze all of the existing stream operators, but we aim to analyze whether the technique can be applied to the input and output of some typical operators to demonstrate the idea.

The **map** operator takes one element and produces one element. Consider an example where a map function doubles the values of the input stream. If the input stream segment satisfies the prerequisites to be applied with DFT, the output stream segment that comes out of it will still maintain a constant sampling interval. It can also be applied with DFT.

The **scaling** operator takes in a stream and scales it in time by a constant. If the input stream segment satisfies the prerequisites to be applied with DFT, the output stream segment that comes out of it will still maintain a constant sampling interval. It can also be applied with DFT.

The **shift** operator takes in a stream and shifts it in time by a constant amount. If the input stream segment satisfies the prerequisites to be applied with DFT, the output stream segment that comes out of it will still maintain a constant sampling interval. It can also be applied with DFT.

The **filtfilt** functor filters out the noise and smooths the stream. If the input stream segment satisfies the prerequisites to be applied with DFT, the output stream segment that comes out of it will still maintain a constant sampling interval. It can also be applied with DFT.

The **filter** operator evaluates a boolean function for each element and retains those for which the function returns true. Even if the input stream segment satisfies the prerequisites to be applied with DFT, the output stream that comes out of does not necessarily maintain a constant sampling interval, it can not necessarily be applied with DFT.

The **findpeaks** finds peaks of the stream. Even if the input stream segment satisfies the prerequisites to be applied with DFT, the output stream that comes out of it does not necessarily maintain a constant sampling interval, it can not necessarily be applied with DFT.

The **length** returns back the length of the stream. Even if the input stream segment satisfies the prerequisites to be applied with DFT, the output stream that comes out of it does not maintain a constant sampling interval, it can not be applied with DFT.

operator name	input	output
map	Yes	Yes
scaling	Yes	Yes
shift	Yes	Yes
filtfilt	Yes	Yes
filter	Yes	likely no
findpeaks	Yes	likely no
length	Yes	No
flatmap	No	No

TABLE 7.2: Applicability of DFT to the input and output of the operators

The **flatmap** operator takes one element and produces zero, one, or more elements. Consider an example where a flatmap operator splits sentences into words. As this operator deals with string, there is no place DFT can be used.

A table that summarizes the applicability of DFT to the input and output of the operators is presented in Table 7.2. When it is possible, we make the input of the operator amenable to the DFT technique.

The final output stream segment will always remain in unreduced form so that humans can consume it. So for single stream, single-operator processing chain, only the type of source stream segment matters. For the processing chain that is made up of multiple operators, we observe the input and output stream segments of the operator (when it is not the last operator) that we can apply this technique.

This analysis provides us guidance as to where to apply DFT. We do realize that it is prospective provenance that we are analyzing here, but we think that the situation when this guidance goes wrong will be rare.

It should still be noted here that the analysis in this section provides us guidance as to where to apply DFT, but it does not mean that the deployment of the technique can always result in reduced stream storage.

7.3.4 Retrieving contributing data

Algorithm 4 describes how to retrieve contributing data through original streaming data for the single stream, single operator structure. Algorithm 5 describes how to retrieve contributing data through reduced streaming data for the single stream, single operator structure. Both algorithms exclude the operator ”flatmap”. We assume the original streaming data in Algorithm 4 has a constant sampling interval. We assume the streaming data from which the FT records are created in Algorithm 5 has a constant sampling interval.

For algorithm 4, we first retrieve the timestamp of the output data and the operator ID it is marked with (line 1 to line 2). We then look up the operator name, the input-output ratio of the operator, the special parameter of the operator (meaning the scaling factor for the scaling operator and shift factor for the shift operator), and the ID of the input stream that fed into that operator from the table that stores metadata about the operators (line 3 to line 6). We then calculate the number of the window that the output data belongs to in line 7. Line 8 gives back the time vector of the aforementioned window while line 9 gives back the stream extract of that window. Then according to the property of the operator, different retrieval rule is utilised. Line 11 to line 22 describes the retrieval of contributing data of the scaling operator and shift operator. Line 23-line 26 describes the retrieval of contributing data of the map and filter operator while line 27 describes the retrieval of contributing data of filtfilt, findpeaks, and length operators.

For algorithm 5, the difference lies in that additionally it needs to retrieve start time, end time, sampling, frequency, magnitude, phase, and stream offset information (line 9-line 17) to reconstruct the input stream for later retrieval.

In order to retrieve contributing data for single stream, multi-operator structure, we trace back along the chain. When we encounter an unreduced stream, we follow algorithm 4. When we encounter an FT record, we follow algorithm 5.

Note that documentation and utilisation of operator rule is not the main focus of this work. We aim to demonstrate the general idea and to make it enough for the latter uses. These algorithms are also open for real-world use case adjustment.

Algorithm 4 Retrieve contributing data through original streaming data for the single stream, single step structure

Require: a data element d in the output stream in one window, time-based window sized W in time unit, samling interval t_s, a collection of input streams $inputStream$
Ensure: set of contributing data elements C in the input stream

$t_{out} \leftarrow d.t$
$opID \leftarrow d.opID$
$opname \leftarrow table.opID.opname$
$InputOutputRatio \leftarrow table.opID.ratio$
$a \leftarrow table.opID.specialParameter$
$inputStreamID \leftarrow table.opID.inputStreamID$
$WindowNum \leftarrow 1 + floor(t_{out}/W)$
$T \leftarrow [(WinodwNum - 1) * W, t_s, WindowNum * W - t_s]$
$inputStreamExtract \leftarrow inputStream.inputStreamID(T/t_s)$
if a ='null **then**
    if opname=='scaling' **then**
        $t_{in} \leftarrow a * t_{out}$
        $n \leftarrow order(t_{in}, T)$
        $C \leftarrow inputStreamExtract(n)$
    else
        if opname=='shift' **then**
            $t_{in} \leftarrow t_{out} - a$
            $n \leftarrow order(t_{in}, T)$
            $C \leftarrow inputStreamExtract(n)$
        end if
    end if
else
    if InputOutputRatio==1 **then**
        $n \leftarrow order(t_{out}, T)$
        $C \leftarrow inputStreamExtract(n)$
    end if
    $C \leftarrow inputStreamExtract(1 : end)$
end if

Algorithm 5 Retrieve contributing data through reduced streaming data for single stream, single step structure

Require: a data element d in the output stream in one window, time-based window sized W in time unit, samling interval t_s, a collection of FT records $inputFTrecordList$ after post-processing input streams

Ensure: set of contributing data elements C in the input stream

1: $t_{out} \leftarrow d.t$
2: $opID \leftarrow d.opID$
3: $opname \leftarrow table.opID.opname$
4: $InputOutputRatio \leftarrow table.opID.ratio$
5: $a \leftarrow table.opID.specialParameter$
6: $inputStreamID \leftarrow table.opID.inputStreamID$
7: $WindowNum \leftarrow 1 + floor(t_{out}/W)$
8: $inputFTrecord = inputFTrecordList.inputStreamID.WindowNum$
9: $starttime \leftarrow inputFTrecord.starttime$
10: $endtime \leftarrow inputFTrecord.endtime$
11: $sampling \leftarrow inputFTrecord.sampling$
12: $T \leftarrow reconstructT(starttime, sampling, endtime)$
13: $f \leftarrow inputFTrecord.frequency$
14: $m \leftarrow inputFTrecord.magnitude$
15: $p \leftarrow inputFTrecord.phrase$
16: $StreamOffset \leftarrow inputFTrecord.StreamOffset$
17: $inputStreamExtract \leftarrow reconstructInputStream(f, m, p, T, StreamOffset)$
18: **if** a ='null **then**
19: **if** opname=='scaling' **then**
20: $t_{in} \leftarrow a * t_{out}$
21: $n \leftarrow order(t_{in}, T)$
22: $C \leftarrow inputStreamExtract(n)$
23: **else**
24: **if** opname=='shift' **then**
25: $t_{in} \leftarrow t_{out} - a$
26: $n \leftarrow order(t_{in}, T)$
27: $C \leftarrow inputStreamExtract(n)$
28: **end if**
29: **end if**
30: **else**
31: **if** InputOutputRatio==1 **then**
32: $n \leftarrow order(t_{out}, T)$
33: $C \leftarrow inputStreamExtract(n)$
34: **end if**
35: $C \leftarrow inputStreamExtract(1 : end)$
36: **end if**

Chapter 8

Analysis of the model over the use case

In this chapter, we describe the underlying stream applications for the respiratory use case and different provenance approaches added upon them in Section 8.1. In Section 8.2, we evaluate different provenance approaches with a set of metrics.

8.1 Setup

The original respiration rate processor is a MATLAB App as described in Chapter 5. In order to make it easier to compare different stream provenance approaches added upon the application, I extracted codes from the MATLAB App in its code view and created two draft MATLAB M-files out of it. Prior to creating the actual scripts, I use the YesWorkflow tool to get some understanding of these workflows. Workflow graphs created through yesWorkflow tools are listed in Appendix E.

After getting more understanding of the workflow, I plot the topology of the stream processing chain and denote the stream and operator with the stream ID number and operator ID number respectively as shown in Figure 8.1. I then code the stream processing topology into the MATLAB scripts.

We now have a length of measurement of 31 minutes of respiration sensor data at hand. Our window for stream processing is the time-based, tumbling window using event time. The size of the window is 1 minute. As I experimented with these 31 windows, the stream segment from the second window appears to be good for demonstrating purpose and I use data from the second window for the following experiments.

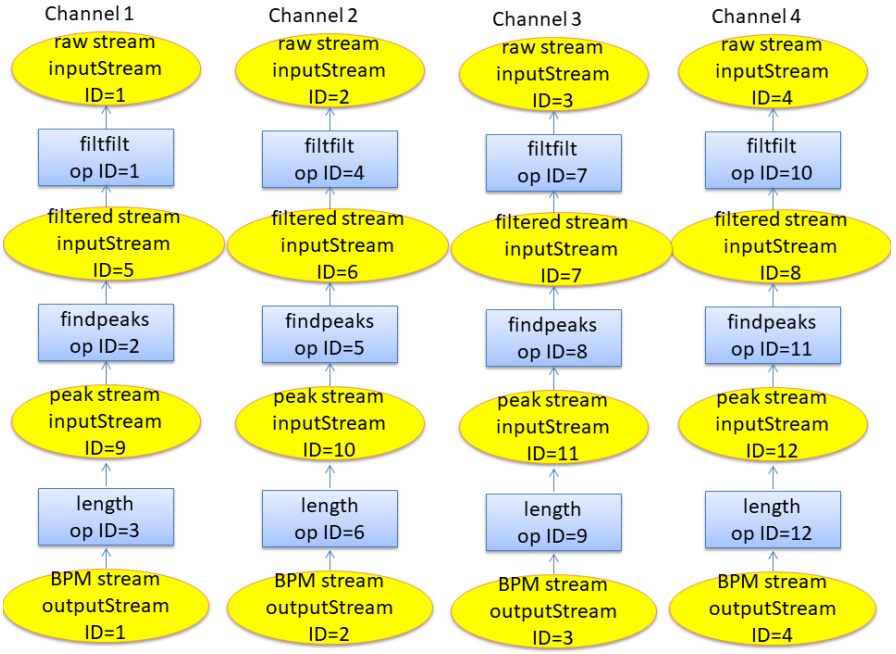

FIGURE 8.1: Topology of stream processing chain

The first script is for getting the respiration rate. It underpins the provenance need 1 and 2 from Section 6.3.1 and Section 6.3.2. The second script is for finding the best position. This script underpins the provenance need 3 from Section 6.3.3. The script to get the respiration rate and the script to find the best position are listed in Section D.2 and Section D.3 in Appendix D respectively. For the script to get the respiration rate, I use sensor data coming from the first channel. I get 9 bpm as output. For the script to find the best position, I get 9bpm, 10bpm, 10bpm, and 9bpm for 4 channels. More specifically as listed in Section D.1, to simulate stream processing over one window, I created a MATLAB script to read the text file that contains the raw sensor data and stored it in ”MATLAB structure” data type named ”inputStream” as if the data from the external source has been put in a log (which is ”MATLAB structure”) for later processing.

The operators used in this application are filtfilt, findpeaks, and length. I then documented the operator ID, operator name, the stream ID of the input stream that feeds into the operator, and special parameter (if there is any) as metadata of each operator beforehand in the excel table.

The descriptions of different stream provenance approaches are as follows.

	A	B	C	D
1	opID	name	stream id	special parameter
2	1	filtfilt	1	
3	2	findpeaks	5	
4	3	length	9	
5	4	filtfilt	2	
6	5	findpeaks	6	
7	6	length	10	
8	7	filtfilt	3	
9	8	findpeaks	7	
10	9	length	11	
11	10	filtfilt	4	
12	11	findpeaks	8	
13	12	length	12	
14				

FIGURE 8.2: A snapshot of the excel table used to store operator metadata

8.1.1 Fine-grained provenance approach using original streaming data over use case

As we do the operations using operators, We stored intermediate streams with the operator IDs they are marked with in the MATLAB structure named ”inputStream” together with the source stream. The output streams with the operator IDs they are marked with are stored in another MATLAB structure data type named ”outputStream” finally. The stream IDs and operator IDs stored are in accordance with the ID numbers displayed in Figure 8.1. A snapshot of the excel table used to store operator metadata is presented in Figure 8.2. The MATLAB script to get the respiration rate added with fine-grained provenance tracking component is listed in Section A.1 in Appendix A. The MATLAB script to find the best position added with fine-grained provenance tracking component is listed in Section A.2 also in Appendix A.

8.1.2 Provenance approach using reduced streaming data with FT over use case

For these streaming applications, we do have prior knowledge about the sampling interval and the type of source stream data that is to be expected. The sampling interval is constant and the source stream is numerical. The source stream can be applied with FFT. We do not need to check to see if it is eligible to be applied with FFT.

Looking at the topology shown in Figure 8.1, we analyze the input and output of the operators involved.

As analyzed earlier in Section 7.3.3, the **filtfilt** operator filters out the noise and smooths the signal. The output stream that comes out of it still maintains a constant sampling interval. It can also be applied with FFT. The **findpeaks** operator finds peaks of the signal. The output stream that comes out of it does not maintain a constant sampling interval. It cannot be applied with FFT.

We then apply FFT on the source stream data sequence and the intermediate stream data sequence that comes out of filtfilt. By convention, the X-axis of the spectrum figure is frequency. For better illustration, we time the X-axis by 60 so that the X-axis becomes cycles/min. As it is impossible to measure frequencies above the Nyquist limit, we reserve and double the spectral values below the Nyquist limit. After the generation of the spectrum figures, we observe the figures and store information about the major spectral lines. This information is stored in a newly created MATLAB structure called ”inputStreamFT”. The ID of the FT record corresponds to stream denotation in the topology described previously. The MATLAB script to post-process the streams for the stream application to get respiration rate is listed in Section B.1 in Appendix B while the MATLAB script to post-process the streams for the stream application to find the best position is listed in Section B.2 in Appendix B.

For the stream application to get respiration rate, this process gives back 2 spectrum figures for observation as shown in Figure 8.3 and Figure 8.4 (we depicted what the original source stream and the intermediate stream look like on the left of the spectrum figures generated).

By observing the above 2 Figures, we maintain bin 9 and 10 that have dominant spectral lines for both figures.

For the stream application to find the best position, this process gives back 8 spectrum figures: 4 figures for post processing 4 source streams and 4 figures for post processing 4 intermediate streams. We are not going to list them specifically here.

8.2 Analysis

8.2.1 Post-processing time

The post-processing of the stream is a semi-automatic process. We first apply FFT on the segment of streams and generate spectrum figures. Then we observe the figures and identify the dominant spectral lines. After that, we store information about these spectral lines which includes start time, sampling, and end time of the stream

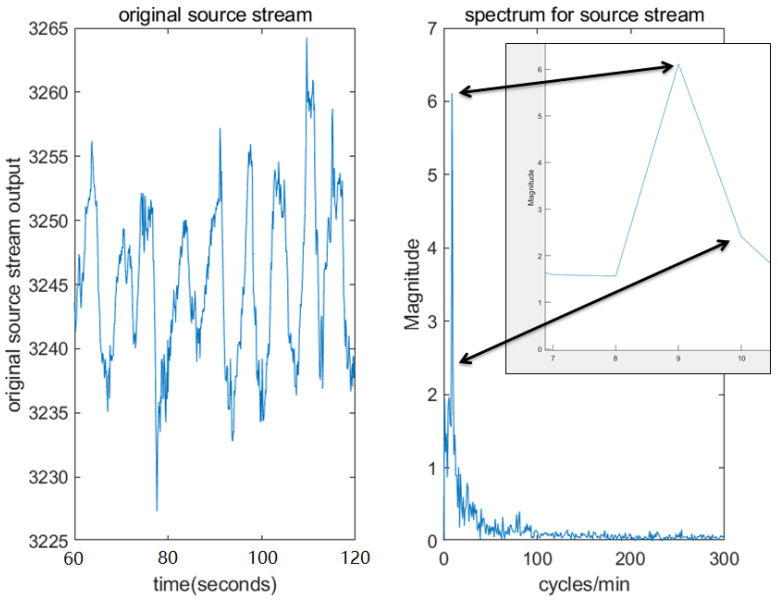

FIGURE 8.3: Spectrum for source stream

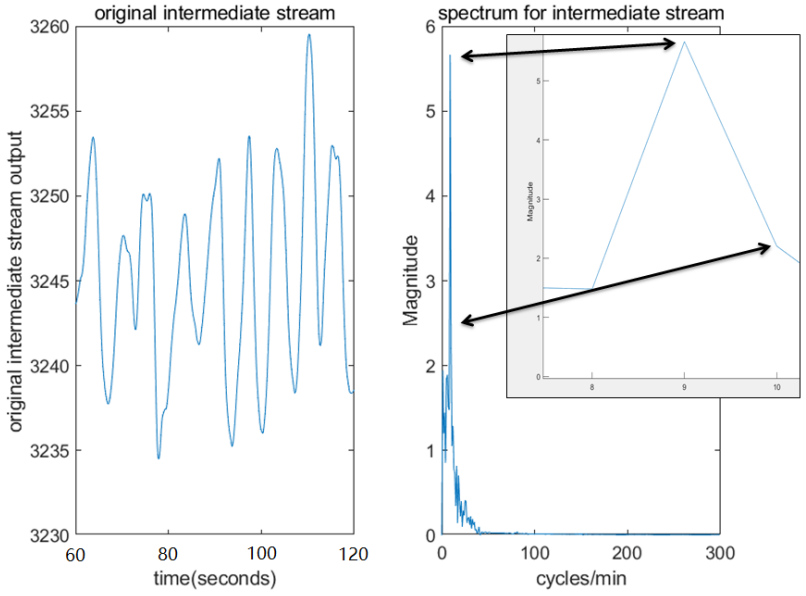

FIGURE 8.4: Spectrum for intermediate stream coming out filtfilt

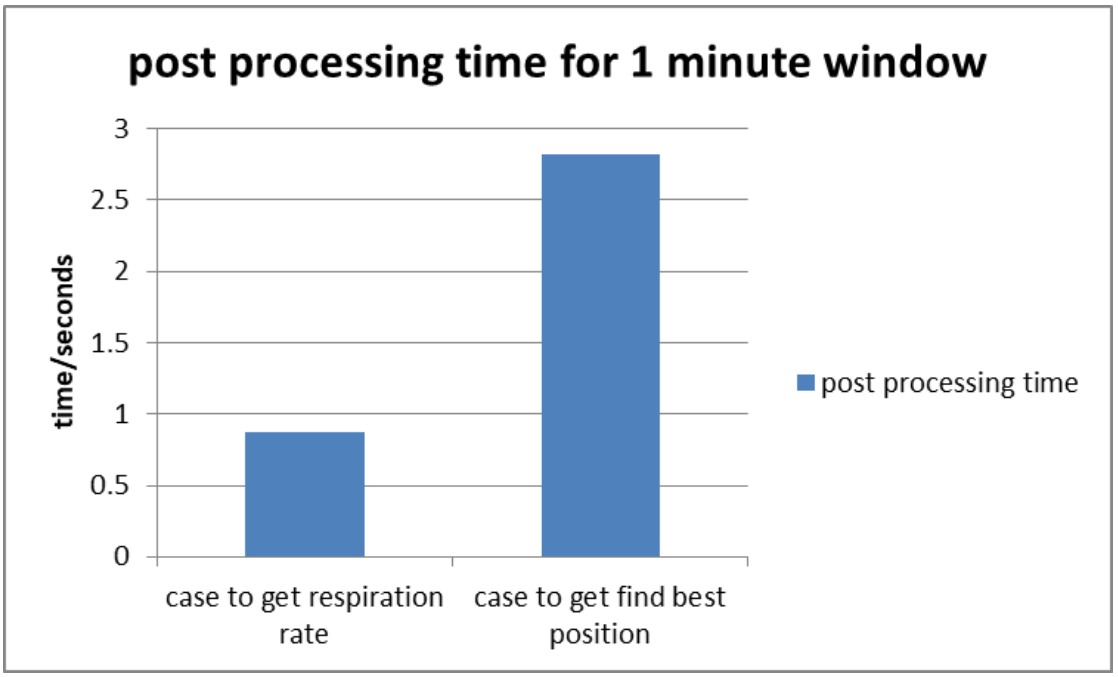

FIGURE 8.5: Post processing time analysis for the stream application to get respiration rate and that to find best position

segment, frequencies and corresponding magnitudes, phases, and offset (we get rid of the DC offset first so that there is no disturbing spectral line at bin 0) for later contributing data retrieval. The post-processing time we analyze here includes time to apply FFT on stream segment(s) and time to store information about reduced stream(s). The post-processing time for the stream application to get respiration rate and that to find the best position is shown in Figure 8.5. The average runtime for the stream application to get respiration rate is 0.008 seconds and 0.013 seconds for that to find the best position. It takes on average 0.873 seconds to post-process streaming data for the stream application to get respiration rate and 2.816 seconds to post-process streaming data for the stream application to find the best position.

This means that for the respiration sensor data which is being processed in near real time after the second one-minute window, only after a reasonable amount of time can the provenance query be initiated. This means that the implementation of the FT technique to post process respiration data in the second one-minute window can support reasonably reactive stream provenance retrieval.

8.2.2 Storage overhead

The storage for the original streaming data equals timestamp, data content, and operator ID attached (if there is any) of the source stream(s) and intermediate stream(s) coming out of filtfilt. The storage for reduced streaming data using FT equals the summation of the start time, sampling, end time, frequency bins and their

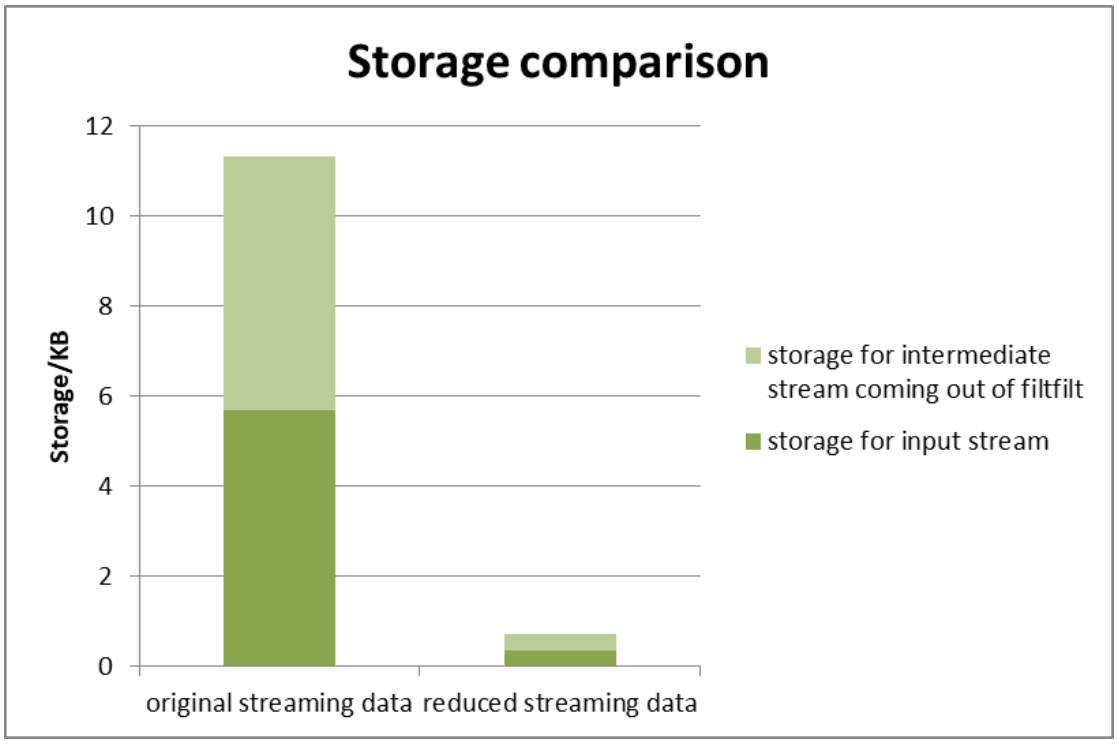

FIGURE 8.6: Storage comparison for the stream application to get respiration rate

corresponding magnitudes, phases, offsets, and operator ID attached (if there is any) of the source and intermediate stream(s). Figure 8.6 illustrates the storage comparison for the stream application to get the respiration rate while Figure 8.7 illustrates the storage comparison for the stream application to find the best position. As it is shown, there is quite a lot of reduction for both applications dropping from 11.33 KB to 0.72KB (the original streaming data is approximately 15.7 times the size of the reduced streaming data) and from 43.5 KB to 1.19 KB (the original streaming data is approximately 36.6 times the size of the reduced streaming data) respectively.

This means that the application of the FT technique over streaming data in the second one-minute window has led to successful storage reduction for both the stream application to get respiration rate and the stream application to find the best position.

8.2.3 Provenance questions answerability

We list in Table 8.1 as follows to compare the utility of provenance retrieval using original streaming data and provenance retrieval using reduced streaming data.

PQ1. PQ1 can be resolved using original streaming data. This is done by looking up the operator ID in the respiration rate, looking up the stream ID in the table containing parameters of the operator, looking up the operator ID in the input stream log and so on and so forth to the source stream. The MATLAB script in Section C.1 in Appendix

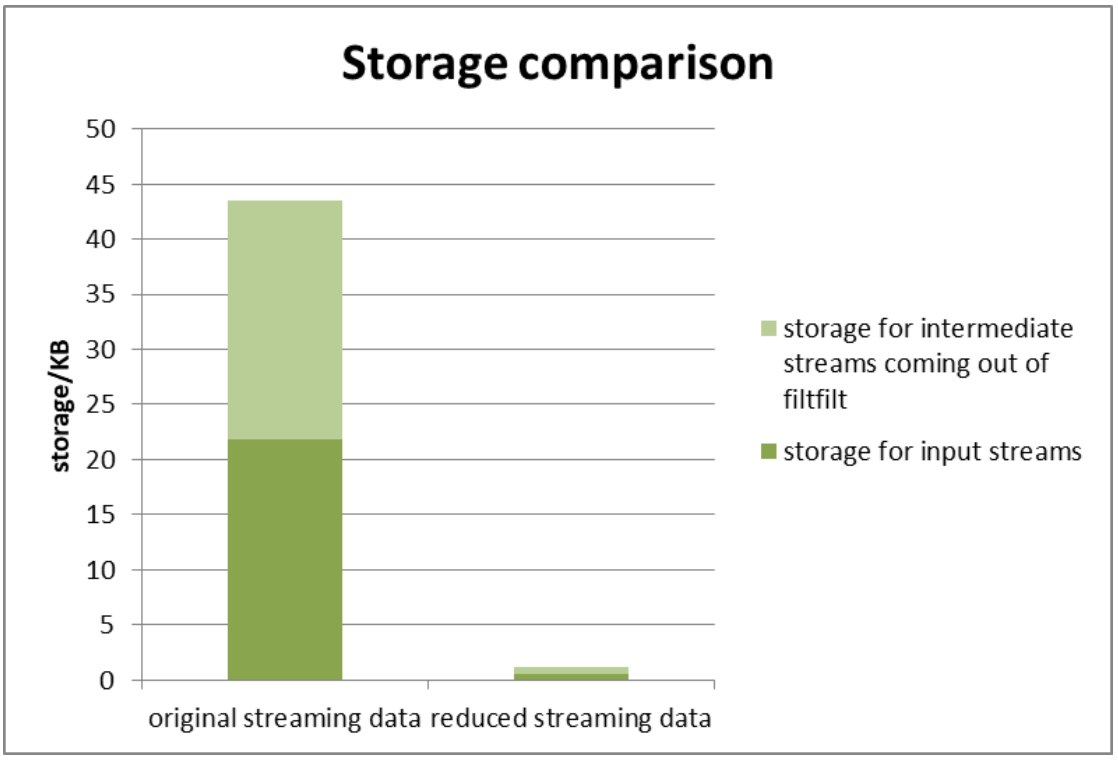

FIGURE 8.7: Storage comparison for the stream application to find best position

method	PQ1	PQ2	PQ3	PQ4	PQ5
provenance retrieval using original streaming data	Yes	Yes	Yes	Yes	Yes
provenance retrieval using reduced streaming data	Yes	Yes	Yes	Yes	Yes

TABLE 8.1: Provenance questions answerability using original streaming data or reduced streaming data over the second window

C shows how the contributing operators and streams are returned. PQ1 can be resolved using reduced streaming data. This is done by looking up the operator ID in the respiration rate, looking up the stream ID in the table containing parameters of the operators, looking up the operator ID in the FT records or input stream log and so on and so forth to the FT record for the source stream. The MATLAB script in Section C.2 in Appendix C shows how the contributing operators and input stream log or FT records are returned. Figure 8.8 shows the result being returned by both scripts. By getting information about the operator such as using a simple ”help” command, the doctors will be able to know the high-level description of the operators. This means that both the original streaming data and reduced streaming data can be used to return back the streams and processes constituting the respiration rate over the second one-minute window on a high level with low granularity.

PQ2. PQ2 can be resolved using original streaming data. According to Algorithm 4 in Section 7.3.4, all elements from one window of the upstream of ”length”, “findpeaks”, and ”filtfilt” contribute to its output. As a result, all data elements of one window from all source or intermediate streams shall be returned. This process doesn’t differ

```
str1 =

    "the operator from upstream to downstream are: filtfilt, findpeaks, length"


str2 =

    "the stream ID from upstream to downstream are: 1, 5, 9"
```

FIGURE 8.8: Result returned for PQ1 using either original streaming data or reduced streaming data

```
str2 =

    "the stream ID from upstream to downstream are: 1, 5, 9"
```

FIGURE 8.9: Result returned for PQ2 using either original streaming data or reduced streaming data

much from the above MATLAB script to answer PQ1 except that it only needs to return the contributing stream ID. The MATLAB script to answer PQ2 is listed in Section C.3 in Appendix C. PQ2 can be resolved using reduced streaming data. This resembles how reduced streaming data was used to answer PQ1. But additionally, the FT records denoted with Stream ID need to be reconstructed and stored in a "MATLAB structure" called "inputStreamReconstructed" should the technician want to look at the streaming data closely. The MATLAB script to answer PQ2 is listed in Section C.4 in Appendix C. Figure 8.9 shows the result being returned by both scripts. This means that both the original streaming data and reduced streaming data can be used to return the data constituting the respiration rate over the second one-minute window with high granularity.

PQ3. PQ3 can be resolved using original streaming data. The operators that touched the upstream are returned by finding the indice of the stream ID in the table containing operator metadata. Unlike PQ1, we return back the specific operator ID besides the operator name. The MATLAB script to answer PQ3 using original streaming data is listed in Section C.5 in Appendix C. PQ3 can be resolved using reduced streaming data. Since the stream ID is the same from either original input stream or FT record, the MATLAB script to answer PQ3 using reduced streaming data is the same as what is listed in Section C.5 in Appendix C. Figure 8.10 shows the result being returned by using either the original streaming data or reduced streaming data. This means that both the original streaming data and reduced streaming data can be equally used to return back the processes that touched the contributing data over the second one-minute window with high granularity.

```
str1 =

    "The stream 1,  5,  9 are touched by operator filtfilt, findpeaks, length whose operator id are 1, 2, 3"
```

FIGURE 8.10: Result returned for PQ3 using either original streaming data or reduced streaming data

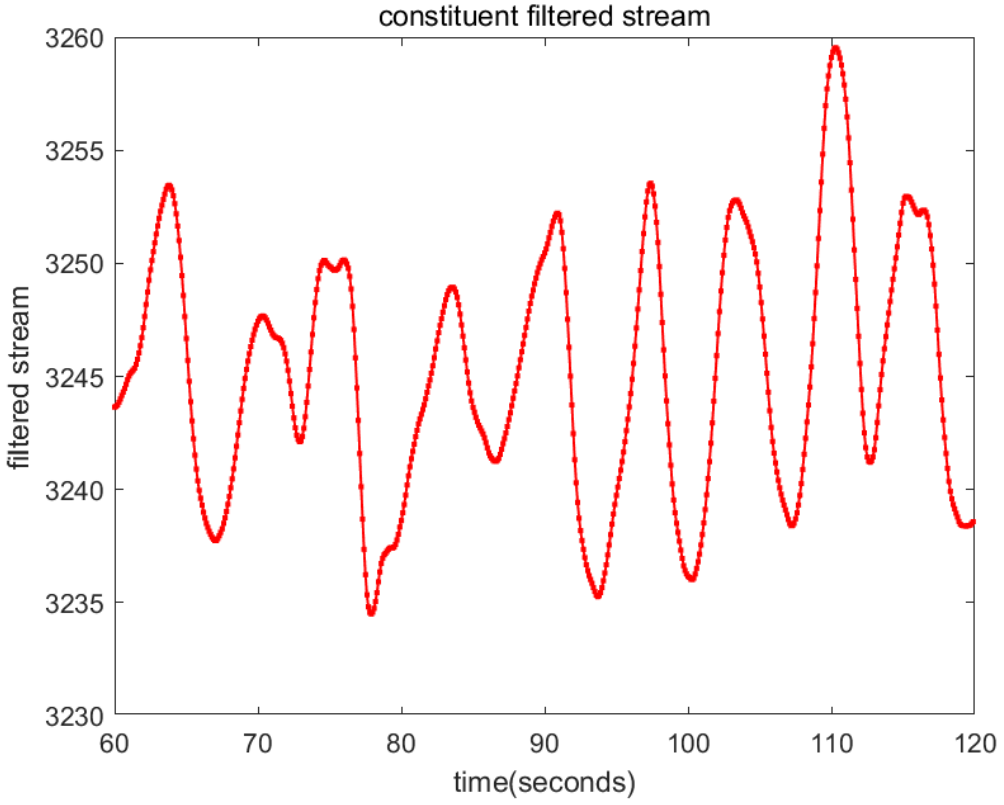

FIGURE 8.11: Result returned for PQ4 using original streaming data

PQ4. PQ4 can be resolved using original streaming data. This is achieved by plotting the stream using the filtered contributing data. The MATLAB script using original streaming data to answer PQ4 is listed in Section C.6 in Appendix C. Figure 8.11 is returned as the query result. PQ4 can be resolved using reduced streaming data. This is achieved by reconstructing the stream using the stored start time, sampling, end time, frequency bins and their corresponding magnitudes, phases, and offsets. The MATLAB script using reduced streaming data to answer PQ4 is listed in Section C.7 in Appendix C. Figure 8.12 is returned as the query result. A figure comparing the reconstructed stream with the original stream is presented in Figure 8.13. The mean absolute percentage error of the reconstructed stream is 0.0865%. This means that the reconstructed stream is very accurate compared with the original stream. Also, we can see in Figure 8.13, both the original stream and reconstructed stream have 9 visual peaks as calculated by the application to get the respiration rate. The reconstructed stream preserves the main stream pattern.

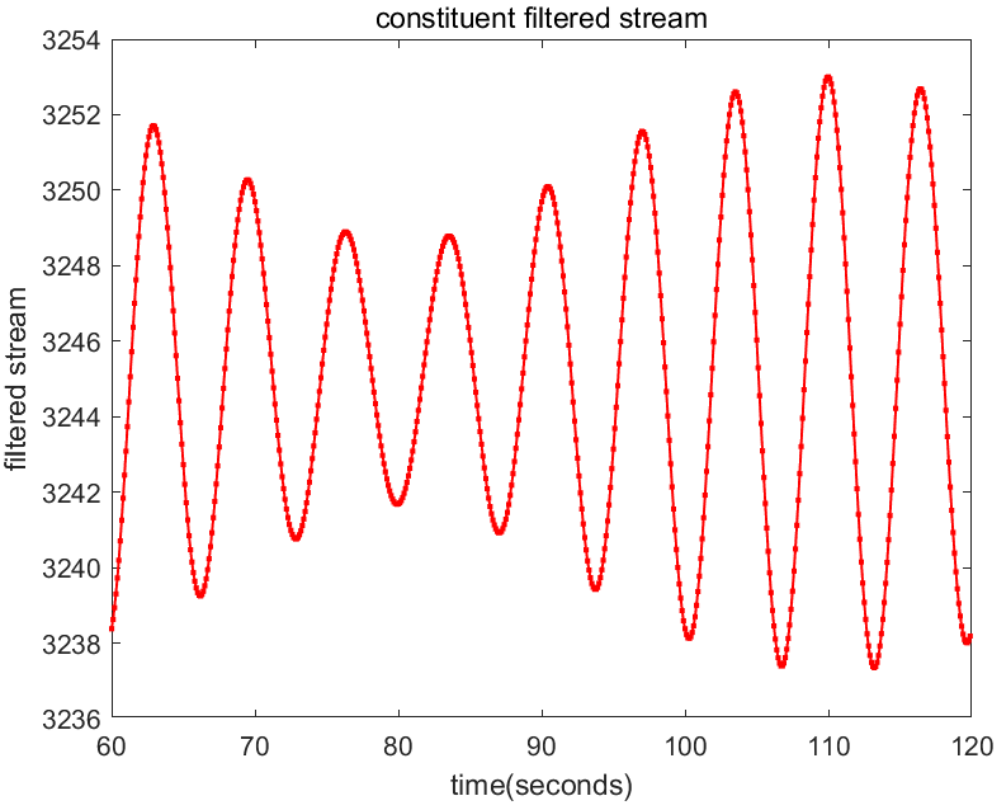

FIGURE 8.12: Result returned for PQ4 using reduced streaming data

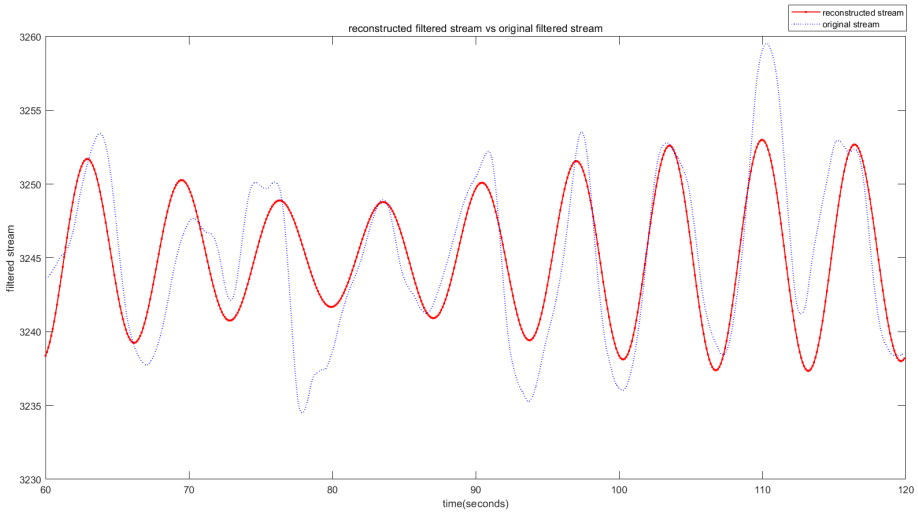

FIGURE 8.13: Comparison of reconstructed stream and original stream

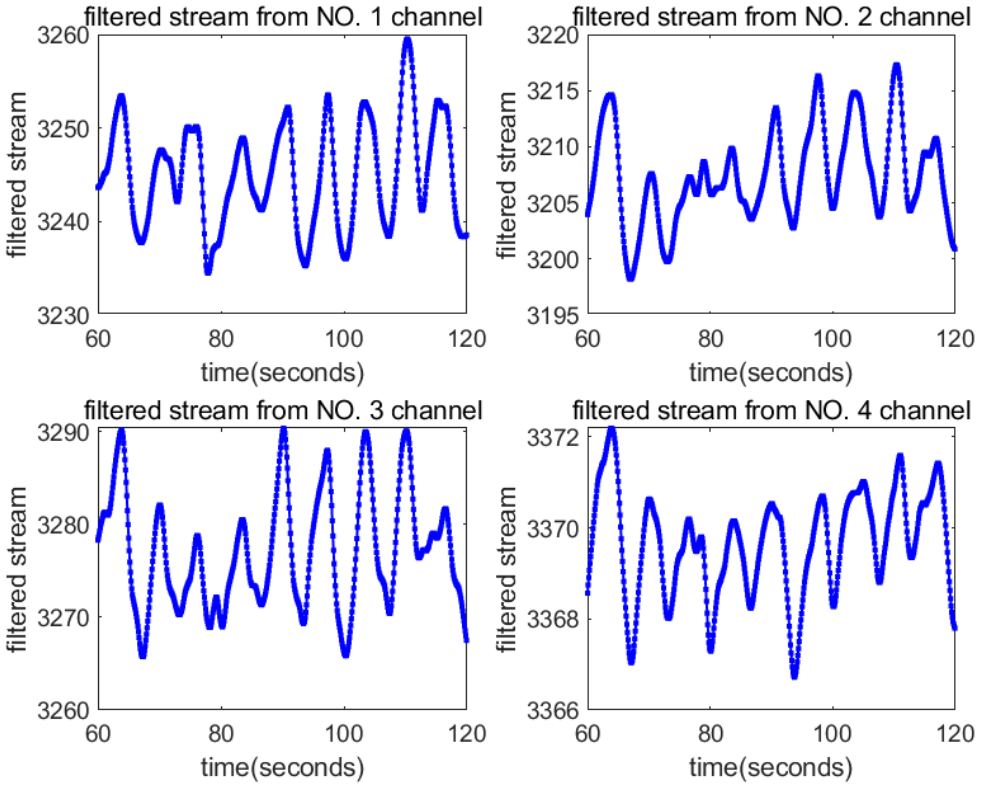

FIGURE 8.14: Result returned for PQ5 using original streaming data

PQ5. PQ5 can be resolved using original streaming data. This is achieved by plotting the filtered streams from the 4 channels using the contributing data. The MATLAB script using original streaming data to answer PQ5 is listed in Section C.8 in Appendix C. Figure 8.14 is returned as the query result. PQ5 can be resolved using reduced streaming data. This is achieved by reconstructing the filtered streams using the stored start time, sampling, end time, frequency bins and their corresponding magnitudes, phases, and offsets. The MATLAB script using reduced streaming data to answer PQ5 is listed in Section C.9 in Appendix C. Figure 8.15 is returned as the query result. A figure comparing reconstructed streams with original streams from 4 channels is presented in Figure 8.16. The mean absolute percentage errors of the reconstructed streams from 4 channels are 0.0865%, 0.0577%, 0.0774%, and 0.0119% respectively. This means that the 4 reconstructed streams are very accurate compared with the original streams. We can see in Figure 8.16, both the original stream and reconstructed stream from channel 1 and 4 have 9 visual peaks as calculated by the application to find the best position. The reconstructed stream preserves the main stream pattern. But for channel 2 and 3, while there are 9 visual peaks for reconstructed streams, the calculated respiration rates are 10. There are some patterns that are missing from the reconstructed streams.

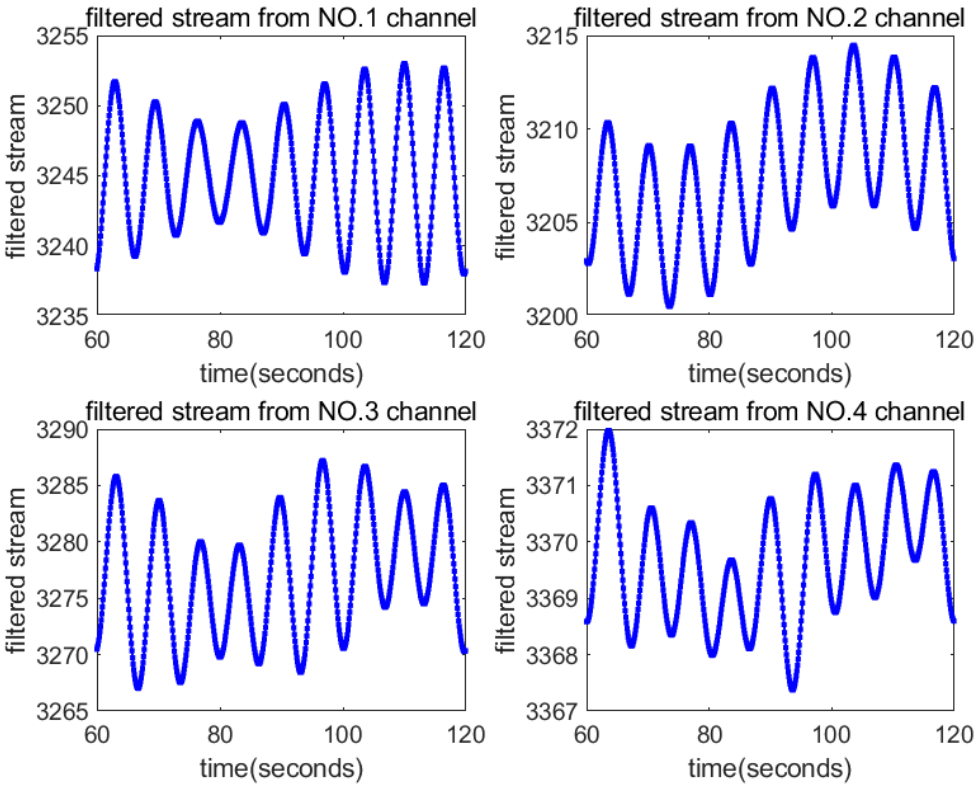

FIGURE 8.15: Result returned for PQ5 using reduced streaming data

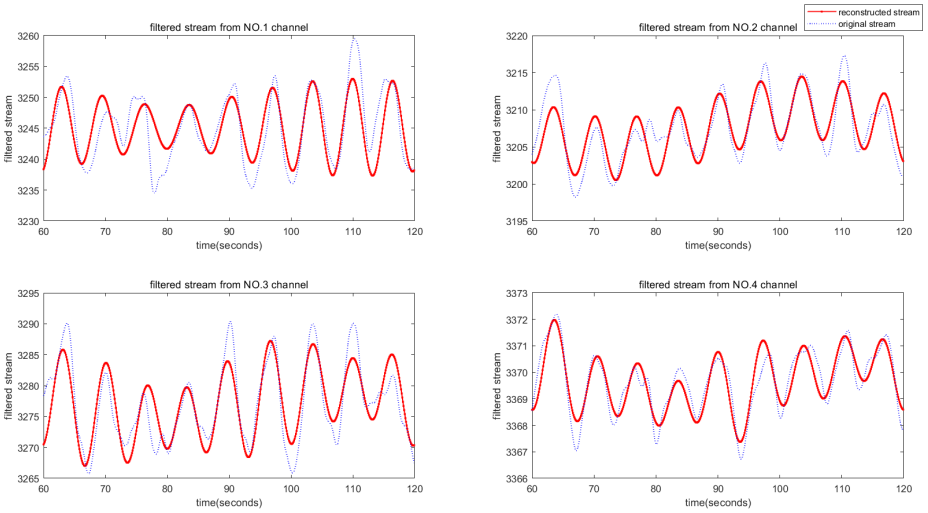

FIGURE 8.16: Comparison of reconstructed streams and original streams

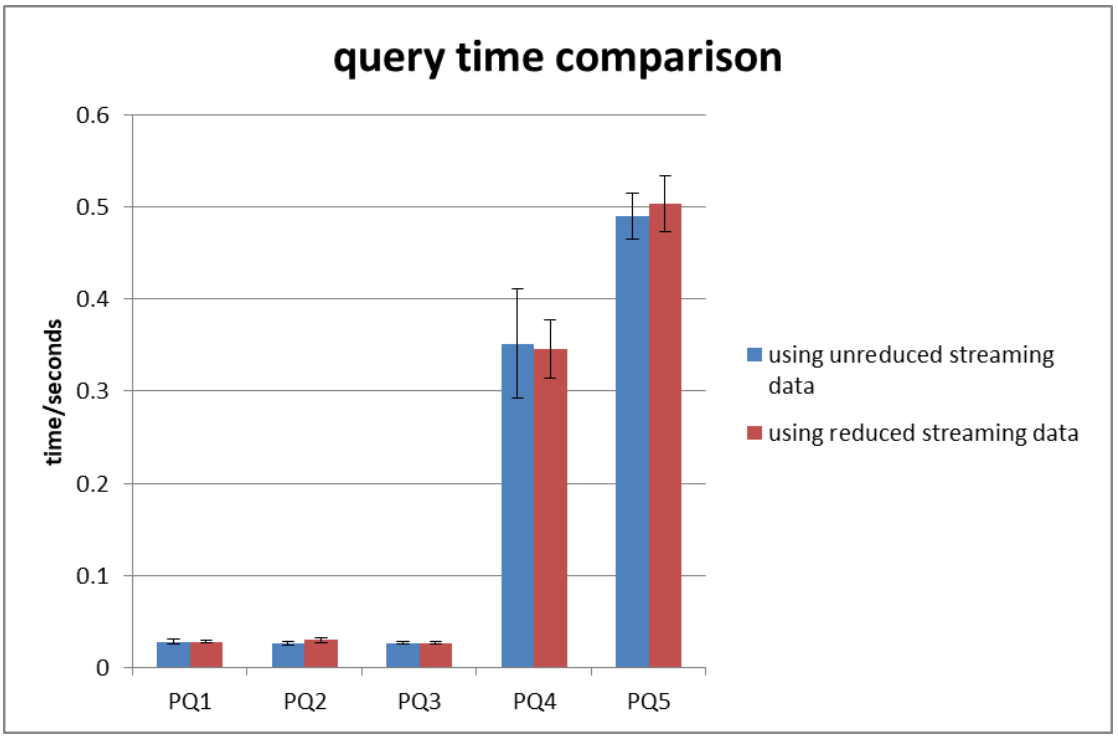

FIGURE 8.17: Query time comparison

8.2.4 Query time

The average query time comparison between provenance retrieval using original streaming data and provenance retrieval using reduced streaming data over 5 provenance questions is shown in Figure 8.17.

For PQ1, it takes on average 0.028 seconds to retrieve contributing data and processes using original streaming data and 0.028 seconds to retrieve contributing data and processes using reduced streaming data. The Wilcoxon Rank Sum Test over these query times shows that there is not enough evidence to reject the null hypothesis of equal medians of query time at the 5% significance level. This is because PQ1 is a high-level provenance question for justification that requires low granularity. The query requires the contributing stream IDs and operator names but not the specific data.

For PQ2, it takes on average 0.026 seconds to retrieve contributing data using original streaming data and 0.029 seconds to retrieve contributing data using reduced streaming data. The Wilcoxon Rank Sum Test over these query times shows the existence of a positive shift in the median of query time from the query using original data to the query using reduced data at the 5% significance level. This is because PQ2 is a provenance question for debugging that requires specific data. The FT records need to be reconstructed for examining. Thus the query using reduced stream takes more time.

For PQ3, as stated earlier in Section 8.2.3, as the operator retrieval using original streaming data and operator retrieval using reduced streaming data share the same MATLAB script, there is no time difference between these 2 queries. It takes 0.026 seconds on average to run the query to answer PQ3.

For PQ4, it takes on average 0.352 seconds to return the figure of the filtered stream using original streaming data and 0.346 seconds using reduced streaming data. The Wilcoxon Rank Sum Test over these query times shows the existence of a positive shift in the median of query time from the query using original data to the query using reduced data at the 5% significance level. This is because PQ4 is a provenance question for debugging that requires specific data for plotting stream figure. The FT record needs to be reconstructed for plotting. Thus the query using reduced stream takes more time.

For PQ5, it takes on average 0.490 seconds to return 4 figures of filtered streams using original streaming data and 0.504 seconds using reduced streaming data. The Wilcoxon Rank Sum Test over these query times shows the existence of a positive shift in the median of query time from the query using original data to the query using reduced data at the 5% significance level. This is because PQ5 is a provenance question for justification that requires specific data for plotting. The FT records need to be reconstructed for plotting. Thus the query using reduced streams takes more time.

8.2.5 Discussion

In summary, in our experimentation of post-processing streaming data with FFT from the respiratory use case over the second window, we can observe the strengths of the approach as follows.

1)The post-processing time over streaming data in the second one-minute window is reasonable (0.873 seconds for the application to get the respiration rate and 2.816 seconds for the application to find the best position) to support reactive stream provenance retrieval.

2)The deployment of the technique over one window led to successful storage reduction making streaming data storage in discussion drop from 11.33KB to 0.72KB (15.7 times reduction effect) for the case to get respiration rate and 43.5KB to 1.19KB (36.6 times reduction effect) for the case to find the best position compared with when there is no reduction technique imposing on the streaming data.

3)The reduced streaming data preserves the ability to answer PQ1 which requires the return of the IDs of contributing stream or FT records and processes on a high level for understanding for doctors which is no different from the query using unreduced streams as the IDs of FT records and those of unreduced streams are the same. This is

also reflected in the query time that there is no significant difference between the medians of the 2 corresponding query times at the 5 % significance level (both the query time using original streaming data and query time using reduced streaming data takes 0.028 seconds on average).

It also preserves the ability to answer PQ2 which requires the return of the detailed contributing streaming data for the technicians by reconstructing detailed streams using the stored FT records.

The reduced streaming data has the ability to answer PQ4 which requires returning the figure of the stream for technician. In our experimentation of the filtered stream over the second window coming from the first channel, the reconstructed stream succeeded in preserving the main pattern of the original stream with a low mean absolute percentage error of 0.0865 %.

The limitation of the approach can be observed as follows.

1)While the 4 reconstructed streams to answer PQ5 has minor mean absolute percentage errors, there are some observable pattern losses for streams coming from channel 2 and 3.

2)There exists a positive shift in the median of query time from the query using original data to the query using reduced data at the 5% significance level for both PQ2, PQ4, and PQ5.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

The overall aim of the thesis is to explore the applicability of the signal processing technique - Fourier transform in reducing streaming data storage while retaining its utility for provenance retrieval.

This is broken down into 3 research questions for exploration.

RQ1. An initial observation of work on provenance of stream in Section 2.3 shows that there is no major difference between the use of provenance in a transactional system and that in a stream processing system.

Thus in Chapter 4, we summarize and propose a classification of provenance usage for the transactional system based on past literature and the provenance use cases proposed by the W3C provenance incubator group which is more mature that a stream application can draw from. We propose provenance questions concerning justification, compliance, trust, debugging, responsibility, taint analysis, and reproducibility. We map the use of provenance for streams extracted from work in Section 2.3 to the classification and find that they sit well within this classification.

We've also come up with 3 provenance needs that can be further broken down into 5 provenance questions for a real respiratory streaming use case in Chapter 5 through 3 methods: literature review around provenance need for health care domain, interviewing and seeking expert opinion, and studying the applicability of the provenance question classification to the provenance need of the streaming application. We find that the provenance questions derived for the respiratory streaming application can all be mapped to the provenance question classification. There comes the limitation for the expert review. Although having senior experts who work closely with the use case will provide valuable insight, having a group of junior

academics in related areas (Stewart and Shamdasani (2014)) to interview will complement the expert review to make the provenance needs for the use case more validated.

With the proposal of a more general provenance usage classification and a specific analysis of provenance needs for a real respiratory streaming use case, RQ1 is answered, which forms the basis for the latter analysis.

RQ2. To answer RQ2, a model for streaming data reduction using Fourier transform along with its analysis is presented in Chapter 7. They show that there is the possibility that FT can reduce the size of streaming data. The retrieval algorithm using FT record from Section 7.3.4 shows the utility of the reduced streaming data for provenance retrieval.

Apart from that, we also create two stream applications in Chapter 8. One stream application is for getting the respiration rate. The other stream application is for finding the best position. The application of the technique over the source and intermediate streams stored during processing of the second window of the respiratory sensor streaming data shows that there is quite a lot of reduction for the source and intermediate streams dropping from 11.33KB to 0.72KB (15.7 times reduction effect) for the case to get respiration rate and 43.5KB to 1.19KB (36.6 times reduction effect) for the case to find the best position. Yet for utility, it comes with some limitations. While using the FT technique doesn't affect the answerability of the query that requires stream ID but not specific data such as PQ1, the query that requires examining data content such as PQ2, and the query that requires contributing operators such as PQ3, for the query that requires returning the figure of the stream such as PQ4 and PQ5, it's possible that there can be some pattern losses.

RQ3. To answer RQ3, metrics such as post-processing time and query time are explored in Chapter 8. The analysis shows that the post-processing time is reasonable for the second one-minute time window and can support more reactive stream provenance retrieval (on average 0.873 seconds for the stream application to get the respiration rate and 2.816 seconds for the stream application to find the best position). The analysis shows that while there is no significant difference between the medians of the query time using unreduced streaming data and reduced streaming data for PQ1 which doesn't require specific data at a 5% significance level, there exists a positive shift in the median of query time from the query using original data to query using reduced data at the 5 % significance level for PQ2, PQ4, and PQ5 which requires specific contributing data. The use of the FT technique doesn't affect the query time for PQ3 as the contributing operators can be retrieved from the table storing the metadata.

9.2 Future work

A number of future works for exploration can be extended from this work.

9.2.1 Analysis of the technique over an aggregation of windows

Applying FT on each individual window of streaming data has the advantage of supporting more reactive stream provenance retrieval. But it comes with certain limitation - even though each window of streaming data is being reduced in FT records, the summation of these FT records can increase the storage.

An extended analysis can be carried out to attempt to apply FT on an aggregation of the windows - for example from window No.1 to window No.5 (Attempt for MATLAB script to simulate processing respiration sensor data for a period of 5 minutes is presented in Section A.3 in Appendix A). There are the following things to look for which could be interesting.

1) the storage comparison of the FT record for the aggregation of windows vs the summation of 5 FT records implementing over 5 individual windows

2) how does applying FT over an aggregation of windows affect reactive stream provenance?

3) how does applying FT over an aggregation of windows affect query answerability?

4) how does applying FT over an aggregation of windows affect query time?

9.2.2 Exploring other techniques for streaming data reduction for provenance retrieval

A potential future work direction is to explore the applicability of other signal processing techniques such as wavelet transform and compressed sensing to reduce storage for the stream segment.

9.2.3 Applying Fourier transform on streaming data from other domains

The idea of using the signal processing technique - Fourier transform on streaming data can be applied to stream processing from other domains.

Appendix A

MATLAB scripts added with provenance tracking components

```
tic
% fetch time and raw sensor data
load('inputStream.mat')
t=inputStream(1).timestamp;%time data
x=inputStream(1).data;%raw sensor data

% filter raw sensor data
windowSize = 10;
b = (1/windowSize)*ones(1,windowSize); %initialize filter window
a = 1;
y = filtfilt(b,a,x);

% storing stream and op id
inputStream(5).timestamp=t;
inputStream(5).data=y;
inputStream(5).opID=1;

% find peaks using MinPeakProminence
prom = 1; %initialize_prom_value
[peakHeight,peakTime]=findpeaks(y,t, 'MinPeakProminence', prom);

% storing stream and op id
inputStream(9).timestamp=peakTime;
inputStream(9).data=peakHeight;
inputStream(9).opID=2;
save 'inputStream' inputStream;

% calculate respiration rate
bpm=length(peakTime);

% storing stream and op id
outputStream(1).timestamp=t(1);
outputStream(1).data=bpm;
outputStream(1).opID=3;

% show in real time
save 'outputStream' outputStream;
```

```
toc
```

LISTING A.1: MATLAB script for the case to get respiration rate added with provenance tracking component

```
tic
% fetch time and raw sensor data for 4 channels
load('inputStream.mat');
t=inputStream(1).timestamp;%time data
x1=inputStream(1).data;%raw sensor data
x2=inputStream(2).data;%raw sensor data
x3=inputStream(3).data;%raw sensor data
x4=inputStream(4).data;%raw sensor data

% filter raw sensor data
windowSize = 10;
b = (1/windowSize)*ones(1,windowSize);
a = 1;
y1 = filtfilt(b,a,x1);
y2 = filtfilt(b,a,x2);
y3 = filtfilt(b,a,x3);
y4 = filtfilt(b,a,x4);

% storing stream and op id
inputStream(5).opID=1;
inputStream(5).timestamp=t;
inputStream(5).data=y1;
inputStream(6).opID=4;
inputStream(6).timestamp=t;
inputStream(6).data=y2;
inputStream(7).opID=7;
inputStream(7).timestamp=t;
inputStream(7).data=y3;
inputStream(8).opID=10;
inputStream(8).timestamp=t;
inputStream(8).data=y4;

% find peaks using MinPeakProminence for filtered sensor data
prom = 1;
[peakHeight1,peakTime1]=findpeaks(y1,t, 'MinPeakProminence', prom);
[peakHeight2,peakTime2]=findpeaks(y2,t, 'MinPeakProminence', prom);
[peakHeight3,peakTime3]=findpeaks(y3,t, 'MinPeakProminence', prom);
[peakHeight4,peakTime4]=findpeaks(y4,t, 'MinPeakProminence', prom);

% storing stream and op id
inputStream(9).opID=2;
inputStream(9).timestamp=peakTime1;
inputStream(9).data=peakHeight1;
inputStream(10).opID=5;
inputStream(10).timestamp=peakTime2;
inputStream(10).data=peakHeight2;
inputStream(11).opID=8;
inputStream(11).timestamp=peakTime3;
inputStream(11).data=peakHeight3;
inputStream(12).opID=11;
inputStream(12).timestamp=peakTime4;
inputStream(12).data=peakHeight4;
save 'inputStream' inputStream;
```

```
% calculate respiration rate
bpm1=length(peakTime1);
bpm2=length(peakTime2);
bpm3=length(peakTime3);
bpm4=length(peakTime4);

% storing stream and op id
outputStream(1).opID=3;
outputStream(1).timestamp=t(1);
outputStream(1).data=bpm1;
outputStream(2).opID=6;
outputStream(2).timestamp=t(1);
outputStream(2).data=bpm2;
outputStream(3).opID=9;
outputStream(3).timestamp=t(1);
outputStream(3).data=bpm3;
outputStream(4).opID=12;
outputStream(4).timestamp=t(1);
outputStream(4).data=bpm4;

save 'outputStream' outputStream;
toc
```

LISTING A.2: MATLAB script for the case to find best position added with provenance tracking component

```
t1=[];x1=[]; t5=[];x5=[];t9=[];x9=[];bpm=[];tout=[];xout=[];

for num=1:5
% fetch time and raw sensor data
load('inputStream.mat')
t=inputStream(1).timestamp;%time data
t=t((num-1)*600+1:num*600);
t1=[t1, t];
x=inputStream(1).data;%raw sensor data
x=x((num-1)*600+1:num*600);
x1=[x1, x];

% filter raw sensor data
windowSize = 10;
b = (1/windowSize)*ones(1,windowSize); %initialize filter window
a = 1;
y = filtfilt(b,a,x);

% storing stream
t5((num-1)*600+1:num*600)=t;
x5((num-1)*600+1:num*600)=y;

% find peaks using MinPeakProminence
prom = 1; %initialize_prom_value
[peakHeight,peakTime]=findpeaks(y,t, 'MinPeakProminence', prom);

% storing stream and op id
t9=[t9;peakTime];
x9=[x9;peakHeight];

% calculate respiration rate
```

```
bpm=[bpm,length(peakTime)];

% storing stream and op id
tout=[tout,num*600];
xout=bpm;

% show in real time
formatSpec1 = "time:%d min; rr: %d \n";
str1 = sprintf(formatSpec1,0.1*tout(end)/60,xout(end))

pause(60)

end

inputStream(5).timestamp=t5;
inputStream(5).data=x5;
inputStream(5).opID=1;
inputStream(9).timestamp=t9;
inputStream(9).data=x9;
inputStream(9).opID=2;
save 'inputStream' inputStream;

outputStream(1).timestamp=tout;
outputStream(1).data=bpm;
outputStream(1).opID=3;
save 'outputStream' outputStream;
```

LISTING A.3: MATLAB script to simulate processing respiration sensor data for a period of 5 minutes

Appendix B

MATLAB scripts to post-process the streams

```
tic
%% postprocess source stream
load('inputStream.mat')
t=inputStream(1).timestamp;%time data
x=inputStream(1).data;%raw sensor data of 1 min time window

N=length(x);
Fs=10*60;

x_new=x-mean(x);%remove DC

X_new=fft(x_new);%fft

f1=figure(1);
P2=abs(X_new/N);
P1 = P2(1:N/2+1);
P1(2:end-1) = 2*P1(2:end-1);
plot(0:(Fs/N):(Fs/2-Fs/N),P1(1:N/2))
ylabel('Magnitude');
xlabel('cycles/min');
title('spectrum for source stream');
saveas(f1,'spectrum for source stream.png');

% store FT records
inputStreamFT(1).starttime=t(1);
inputStreamFT(1).sampling=t(2)-t(1);
inputStreamFT(1).endtime=t(end);
inputStreamFT(1).frequency=9:10;
inputStreamFT(1).magnitude=P1(10:11);
inputStreamFT(1).phrase=angle(X_new(10:11));
inputStreamFT(1).offset=mean(x);


%% post process intermediate stream
t2=inputStream(5).timestamp;%time data
x2=inputStream(5).data;%raw sensor data
```

```
x2_new=x2-mean(x2);%remove DC

X2_new=fft(x2_new);%fft

f2=figure(2);
P2=abs(X2_new/N);
P1 = P2(1:N/2+1);
P1(2:end-1) = 2*P1(2:end-1);
plot(0:(Fs/N):(Fs/2-Fs/N),P1(1:N/2))
ylabel('Magnitude');
xlabel('cycles/min');
title('spectrum for intermediate stream');
saveas(f2,'spectrum for intermediate stream.png');

% store FT records
inputStreamFT(5).starttime=t2(1);
inputStreamFT(5).sampling=t2(2)-t2(1);
inputStreamFT(5).endtime=t2(end);
inputStreamFT(5).frequency=9:10;
inputStreamFT(5).magnitude=P1(10:11);
inputStreamFT(5).phrase=angle(X2_new(10:11));
inputStreamFT(5).offset=mean(x2);
inputStreamFT(5).opID=inputStream(5).opID;


save 'inputStreamFT' inputStreamFT;
toc
```

LISTING B.1: MATLAB script to post-process the stream for the case to get respiration rate

```
tic
%% postprocess source stream
load('inputStream.mat');
t=inputStream(1).timestamp;%time data

N=length(t);
Fs=10*60;

%----observe spectrum for source stream for left chest
x=inputStream(1).data;
x_new=x-mean(x);%remove DC

X_new=fft(x_new);%fft

f1=figure(1);
P2=abs(X_new/N);
P1 = P2(1:N/2+1);
P1(2:end-1) = 2*P1(2:end-1);
plot(0:(Fs/N):(Fs/2-Fs/N),P1(1:N/2))
ylabel('Magnitude');
xlabel('cycles/min');
title('spectrum for source stream for left chest');
saveas(f1,'spectrum for source stream for left chest.png');

% FT record for left chest
inputStreamFT(1).starttime=t(1);
inputStreamFT(1).sampling=t(2)-t(1);
```

```
inputStreamFT(1).endtime=t(end);
inputStreamFT(1).frequency=9:10;
inputStreamFT(1).magnitude=P1(10:11);
inputStreamFT(1).phrase=angle(X_new(10:11));
inputStreamFT(1).offset=mean(x);

% ----observe spectrum for source stream for right chest
x=inputStream(2).data;
x_new=x-mean(x);%remove DC

X_new=fft(x_new);%fft

f2=figure(2);
P2=abs(X_new/N);
P1 = P2(1:N/2+1);
P1(2:end-1) = 2*P1(2:end-1);
plot(0:(Fs/N):(Fs/2-Fs/N),P1(1:N/2))
ylabel('Magnitude');
xlabel('cycles/min');
title('spectrum for source stream for right chest');
saveas(f2,'spectrum for source stream for right chest.png');

% store FT records for right chest
inputStreamFT(2).starttime=t(1);
inputStreamFT(2).sampling=t(2)-t(1);
inputStreamFT(2).endtime=t(end);
inputStreamFT(2).frequency=[1 9];
inputStreamFT(2).magnitude=P1([2,10]);
inputStreamFT(2).phrase=angle(X_new([2,10]));
inputStreamFT(2).offset=mean(x);

%----observe spectrum for source stream for left side
x=inputStream(3).data;
x_new=x-mean(x);%remove DC

X_new=fft(x_new);%fft

f3=figure(3);
P2=abs(X_new/N);
P1 = P2(1:N/2+1);
P1(2:end-1) = 2*P1(2:end-1);
plot(0:(Fs/N):(Fs/2-Fs/N),P1(1:N/2))
ylabel('Magnitude');
xlabel('cycles/min');
title('spectrum for source stream for left side');
saveas(f1,'spectrum for source stream for left side.png');

% FT record for left side
inputStreamFT(3).starttime=t(1);
inputStreamFT(3).sampling=t(2)-t(1);
inputStreamFT(3).endtime=t(end);
inputStreamFT(3).frequency=[1 7 9];
inputStreamFT(3).magnitude=P1([2,8,10]);
inputStreamFT(3).phrase=angle(X_new([2,8,10]));
inputStreamFT(3).offset=mean(x);

%----observe spectrum for source stream for right side
x=inputStream(4).data;
x_new=x-mean(x);%remove DC
```

```
X_new=fft(x_new);%fft

f4=figure(4);
P2=abs(X_new/N);
P1 = P2(1:N/2+1);
P1(2:end-1) = 2*P1(2:end-1);
plot(0:(Fs/N):(Fs/2-Fs/N),P1(1:N/2))
ylabel('Magnitude');
xlabel('cycles/min');
title('spectrum for source stream for right side');
saveas(f1,'spectrum for source stream for right side.png');

% FT record for right side
inputStreamFT(4).starttime=t(1);
inputStreamFT(4).sampling=t(2)-t(1);
inputStreamFT(4).endtime=t(end);
inputStreamFT(4).frequency=[1 5 7 9];
inputStreamFT(4).magnitude=P1([2,6,8,10]);
inputStreamFT(4).phrase=angle(X_new([2,6,8,10]));
inputStreamFT(4).offset=mean(x);

%% post process intermediate stream
%----observe spectrum for intermediate stream for left chest
x=inputStream(5).data; %intermediate stream coming out of filtfilt

x_new=x-mean(x);%remove DC

X_new=fft(x_new);%fft

f5=figure(5);
P2=abs(X_new/N);
P1 = P2(1:N/2+1);
P1(2:end-1) = 2*P1(2:end-1);
plot(0:(Fs/N):(Fs/2-Fs/N),P1(1:N/2))
ylabel('Magnitude');
xlabel('cycles/min');
title('spectrum for intermediate stream for left chest');
saveas(f5,'spectrum for intermediate stream for left chest.png');

inputStreamFT(5).starttime=t(1);
inputStreamFT(5).sampling=t(2)-t(1);
inputStreamFT(5).endtime=t(end);
inputStreamFT(5).frequency=9:10;
inputStreamFT(5).magnitude=P1(10:11);
inputStreamFT(5).phrase=angle(X_new(10:11));
inputStreamFT(5).offset=mean(x);

%----observe spectrum for intermediate stream for right chest
x=inputStream(6).data; %intermediate stream coming out of filtfilt

x_new=x-mean(x);%remove DC

X_new=fft(x_new);%fft

f6=figure(6);
P2=abs(X_new/N);
P1 = P2(1:N/2+1);
P1(2:end-1) = 2*P1(2:end-1);
```

```
plot(0:(Fs/N):(Fs/2-Fs/N),P1(1:N/2))
ylabel('Magnitude');
xlabel('cycles/min');
title('spectrum for intermediate stream for right chest');
saveas(f6,'spectrum for intermediate stream for right chest.png');

inputStreamFT(6).starttime=t(1);
inputStreamFT(6).sampling=t(2)-t(1);
inputStreamFT(6).endtime=t(end);
inputStreamFT(6).frequency=[1 9];
inputStreamFT(6).magnitude=P1([2,10]);
inputStreamFT(6).phrase=angle(X_new([2,10]));
inputStreamFT(6).offset=mean(x);


%----observe spectrum for intermediate stream for left side
x=inputStream(7).data; %intermediate stream coming out of filtfilt

x_new=x-mean(x);%remove DC

X_new=fft(x_new);%fft

f7=figure(7);
P2=abs(X_new/N);
P1 = P2(1:N/2+1);
P1(2:end-1) = 2*P1(2:end-1);
plot(0:(Fs/N):(Fs/2-Fs/N),P1(1:N/2))
ylabel('Magnitude');
xlabel('cycles/min');
title('spectrum for intermediate stream for left side');
saveas(f7,'spectrum for intermediate stream for left side.png');

inputStreamFT(7).starttime=t(1);
inputStreamFT(7).sampling=t(2)-t(1);
inputStreamFT(7).endtime=t(end);
inputStreamFT(7).frequency=[1 7 9];
inputStreamFT(7).magnitude=P1([2,8,10]);
inputStreamFT(7).phrase=angle(X_new([2,8,10]));
inputStreamFT(7).offset=mean(x);


%----observe spectrum for intermediate stream for right side
x=inputStream(8).data; %intermediate stream coming out of filtfilt

x_new=x-mean(x);%remove DC

X_new=fft(x_new);%fft

f8=figure(8);
P2=abs(X_new/N);
P1 = P2(1:N/2+1);
P1(2:end-1) = 2*P1(2:end-1);
plot(0:(Fs/N):(Fs/2-Fs/N),P1(1:N/2))
ylabel('Magnitude');
xlabel('cycles/min');
title('spectrum for intermediate stream for right side');
saveas(f8,'spectrum for intermediate stream for right side.png');

inputStreamFT(8).starttime=t(1);
```

```
inputStreamFT(8).sampling=t(2)-t(1);
inputStreamFT(8).endtime=t(end);
inputStreamFT(8).frequency=[1 5 7 9];
inputStreamFT(8).magnitude=P1([2,6,8,10]);
inputStreamFT(8).phrase=angle(X_new([2,6,8,10]));
inputStreamFT(8).offset=mean(x);

save 'inputStreamFT' inputStreamFT;

toc
```

LISTING B.2: MATLAB script to post-process the stream for the case to find best position

Appendix C

MATLAB scripts to answer provenance questions

```
tic
load('outputStream.mat') %input
load('inputStream.mat')
opPara = readcell('op_parameter.xlsx');

opID1=outputStream.opID;
opName1=opPara{opID1+1,2};

inputStream1ID=opPara{opID1+1,3};

opID2=inputStream(inputStream1ID).opID;
opName2=opPara{opID2+1,2};

inputStream2ID=opPara{opID2+1,3};

opID3=inputStream(inputStream2ID).opID;
opName3=opPara{opID3+1,2};

inputStream3ID=opPara{opID3+1,3};

formatSpec1 = "the operator from downstream to upstream are: %s, %s, %s";
str1 = sprintf(formatSpec1,opName1,opName2,opName3)
formatSpec2 = "the stream ID (starting from the first non-output stream) from downstream to
...upstream are: %d, %d, %d";
str2 = sprintf(formatSpec2,inputStream1ID,inputStream2ID,inputStream3ID)

toc
```

LISTING C.1: MATLAB script to answer PQ1 using original streaming data

```
tic
load('outputStream.mat')
load('inputStream.mat')
load('inputStreamFT.mat')
opPara = readcell('op_parameter.xlsx');
```

```
opID1=outputStream.opID;
opName1=opPara{opID1+1,2};

inputStream1ID=opPara{opID1+1,3};

opID2=inputStream(inputStream1ID).opID;
opName2=opPara{opID2+1,2};

inputStream2ID=opPara{opID2+1,3};

opID3=inputStreamFT(inputStream2ID).opID;
opName3=opPara{opID3+1,2};

inputStream3ID=opPara{opID3+1,3};

formatSpec1 = "the operator from upstream to downstream are: %s, %s, %s";
str1 = sprintf(formatSpec1,opName3,opName2,opName1)
formatSpec2 = "the stream ID from upstream to downstream are: %d, %d, %d";
str2 = sprintf(formatSpec2,inputStream3ID,inputStream2ID,inputStream1ID)

toc
```

LISTING C.2: MATLAB script to answer PQ1 using reduced streaming data

```
tic
load('outputStream.mat') %input
load('inputStream.mat')
opPara = readcell('op_parameter.xlsx');

opID1=outputStream.opID;

inputStream1ID=opPara{opID1+1,3};

opID2=inputStream(inputStream1ID).opID;

inputStream2ID=opPara{opID2+1,3};

opID3=inputStream(inputStream2ID).opID;

inputStream3ID=opPara{opID3+1,3};

formatSpec2 = "the stream ID (starting from the first non-output stream) from downstream to
...upstream are: %d, %d, %d";
str2 = sprintf(formatSpec2,inputStream1ID,inputStream2ID,inputStream3ID)

toc
```

LISTING C.3: MATLAB script to answer PQ2 using original streaming data

```
tic
load('outputStream.mat')
load('inputStream.mat')
load('inputStreamFT.mat')
opPara = readcell('op_parameter.xlsx');

opID1=outputStream.opID;
```

```
inputStream1ID=opPara{opID1+1,3};

opID2=inputStream(inputStream1ID).opID;

inputStream2ID=opPara{opID2+1,3};
%reconstruct stream using inputStream2ID
inputStreamReconstructed(inputStream2ID).timestamp=inputStreamFT(inputStream2ID).starttime:
...inputStreamFT(inputStream2ID).sampling:inputStreamFT(inputStream2ID).endtime+
...inputStreamFT(inputStream2ID).sampling;
t2start=0;
t2sampling=inputStreamFT(inputStream2ID).sampling;
t2end= inputStreamFT(inputStream2ID).endtime-inputStreamFT(inputStream2ID).starttime;
t2=t2start:t2sampling:t2end+t2sampling;
offset=inputStreamFT(inputStream2ID).offset;
f=inputStreamFT(inputStream2ID).frequency;
m=inputStreamFT(inputStream2ID).magnitude;
p=inputStreamFT(inputStream2ID).phrase;

af=2*pi/60*f';
aft=af*t2;
aftp=p*ones(1,length(t2))+aft;
y= cos(aftp);
M=m*ones(1,length(t2));
y=M.*y;
y=cumsum(y,1);
y=y(end,:);
y=y+offset;

inputStreamReconstructed(inputStream2ID).data=y;

inputStreamReconstructed(inputStream2ID).opID=inputStreamFT(inputStream2ID).opID;

opID3=inputStreamFT(inputStream2ID).opID;

inputStream3ID=opPara{opID3+1,3};
%reconstruct stream using inputStream3ID
inputStreamReconstructed(inputStream3ID).timestamp=inputStreamFT(inputStream3ID).starttime:
...inputStreamFT(inputStream3ID).sampling:inputStreamFT(inputStream3ID).endtime+
...inputStreamFT(inputStream3ID).sampling;
t2start=0;
t2sampling=inputStreamFT(inputStream3ID).sampling;
t2end=inputStreamFT(inputStream3ID).endtime-inputStreamFT(inputStream3ID).starttime;
t2=t2start:t2sampling:t2end+t2sampling;
offset=inputStreamFT(inputStream3ID).offset;
f=inputStreamFT(inputStream3ID).frequency;
m=inputStreamFT(inputStream3ID).magnitude;
p=inputStreamFT(inputStream3ID).phrase;

af=2*pi/60*f';
aft=af*t2;
aftp=p*ones(1,length(t2))+aft;
y= cos(aftp);
M=m*ones(1,length(t2));
y=M.*y;
y=cumsum(y,1);
y=y(end,:);
y=y+offset;
```

```
inputStreamReconstructed(inputStream3ID).data=y;

save 'inputStreamReconstructed' inputStreamReconstructed;

formatSpec2 = "the stream ID from upstream to downstream are: %d, %d, %d";
str2 = sprintf(formatSpec2,inputStream3ID,inputStream2ID,inputStream1ID)

toc
```

LISTING C.4: MATLAB script to answer PQ2 using reduced streaming data

```
tic
load('outputStream.mat')
opPara = readcell('op_parameter.xlsx');

inputStream1ID=9;%input
inputStream2ID=5;
inputStream3ID=1;

m=cell2mat(opPara(2:13,3)); % extract colum denoted stream ID

opid3=find(m==inputStream3ID);
opName3=opPara{opid3+1,2};

opid2=find(m==inputStream2ID);
opName2=opPara{opid2+1,2};

opid1=find(m==inputStream1ID);
opName1=opPara{opid1+1,2};

formatSpec1 = "The stream %d, %d, %d are touched by operator %s,%s,%s whose operator id are
...%d,%d,%d";
str1 = sprintf(formatSpec1,inputStream3ID,inputStream2ID,inputStream1ID,opName3, opName2,
...opName1, opid3,opid2,opid1)

toc
```

LISTING C.5: MATLAB script to answer PQ3 using original streaming data or reduced streaming data

```
tic
load('inputStream.mat')

t2=inputStream(5).timestamp;%input
y=inputStream(5).data;

f1=figure(1);
plot(t2,y,'r.-', 'LineWidth', 1);
xlabel('time(seconds)');
ylabel('filtered stream');
title('constituent filtered stream');
saveas(f1,'constituent_stream.png');
toc
```

LISTING C.6: MATLAB script to answer PQ4 using original streaming data

```
tic
load('inputStreamFT.mat')

t2start=0;
t2sampling=inputStreamFT(5).sampling;
t2end= inputStreamFT(5).endtime-inputStreamFT(5).starttime+inputStreamFT(5).sampling;
t2=t2start:t2sampling:t2end;
offset=inputStreamFT(5).offset;
f=inputStreamFT(5).frequency;
m=inputStreamFT(5).magnitude;
p=inputStreamFT(5).phrase;

af=2*pi/60*f';
aft=af*t2;
aftp=p*ones(1,length(t2))+aft;
y= cos(aftp);
M=m*ones(1,length(t2));
y=M.*y;
y=cumsum(y,1);
y=y(end,:);
y=y+offset;

f1=figure(1);
t=inputStreamFT(5).starttime:inputStreamFT(5).sampling:inputStreamFT(5).endtime+
...inputStreamFT(5).sampling;
plot(t,y,'r.-', 'LineWidth', 1);
xlabel('time(seconds)');
ylabel('filtered stream');
title('constituent filtered stream');
saveas(f1,'constituent_stream2.png');
toc
```

LISTING C.7: MATLAB script to answer PQ4 using reduced streaming data

```
tic
load('inputStream.mat')

f1=figure(1);
for i=5:8
    t=inputStream(i).timestamp;
    x=inputStream(i).data;
    subplot(2,2,i-4);
    plot(t,x,'b.-', 'LineWidth', 1);
    xlabel('time(seconds)');
    ylabel('filtered stream');
    title(sprintf('filtered stream from NO. %d channel',i-4))
    t=[];x=[];
end
saveas(f1,'4 filtered stream.png');
toc
```

LISTING C.8: MATLAB script to answer PQ5 using original streaming data

```
tic
load('inputStreamFT.mat')
```

```
f1=figure(1);

for i=5:8
time=inputStreamFT(i).starttime:inputStreamFT(i).sampling:inputStreamFT(i).endtime+
...inputStreamFT(i).sampling;
tstart=0;
tsampling=inputStreamFT(i).sampling;
tend= inputStreamFT(i).endtime-inputStreamFT(i).starttime+inputStreamFT(i).sampling;
t=tstart:tsampling:tend;
offset=inputStreamFT(i).offset;
f=inputStreamFT(i).frequency;
m=inputStreamFT(i).magnitude;
p=inputStreamFT(i).phrase;

af=2*pi/60*f';
aft=af*t;
aftp=p*ones(1,length(t))+aft;
y= cos(aftp);
M=m*ones(1,length(t));
y=M.*y;
y=cumsum(y,1);
y=y(end,:);
y=y+offset;

subplot(2,2,i-4);
plot(time,y,'b.-', 'LineWidth', 1);
xlabel('time(seconds)');
ylabel('filtered stream');
title(sprintf('filtered stream from NO.%d channel',i-4))
end

saveas(f1,'filtered stream FT.png');


toc
```

LISTING C.9: MATLAB script to answer PQ5 using reduced streaming data

Appendix D

MATLAB scripts to get respiration rate and to find best position

```
clear

fid = fopen('res_data.txt');
delimiter = '\t';
startRow = 23;
formatSpec = '%f%f%f%f%f%f%[^\n\r]';
dataArray = textscan(fid, formatSpec, 'Delimiter', delimiter, 'TextType', 'string',...
    'EmptyValue', NaN,'HeaderLines' ,startRow-1, 'ReturnOnError', false, 'EndOfLine', '\r\n');

t=dataArray{1};%time data
num=2; %which number of window
t=t((num-1)*600+1:num*600);
x1=flip(dataArray{2});%raw sensor data no.1 channel
x1=x1((num-1)*600+1:num*600);
x2=flip(dataArray{3});%raw sensor data no.2 channel
x2=x2((num-1)*600+1:num*600);
x3=flip(dataArray{4});%raw sensor data no.3 channel
x3=x3((num-1)*600+1:num*600);
x4=flip(dataArray{5});%raw sensor data no.4 channel
x4=x4((num-1)*600+1:num*600);

inputStream(1).timestamp=t;
inputStream(1).data=x1;
inputStream(2).timestamp=t;
inputStream(2).data=x2;
inputStream(3).timestamp=t;
inputStream(3).data=x3;
inputStream(4).timestamp=t;
inputStream(4).data=x4;

save 'inputStream' inputStream;
```

LISTING D.1: MATLAB script to grasp sensor data

```
% fetch time and raw sensor data
```

```
load('inputStream.mat')
t=inputStream(1).timestamp;%time data
x=inputStream(1).data;%raw sensor data

% filter raw sensor data
windowSize = 10;
b = (1/windowSize)*ones(1,windowSize); %initialize filter window
a = 1;
y = filtfilt(b,a,x);

% find peaks using MinPeakProminence
prom = 1; %initialize prom value
[peakHeight,peakTime]=findpeaks(y,t, 'MinPeakProminence', prom);

% calculate respiration rate
bpm=length(peakTime);

outputStream(1).timestamp=t(1);
outputStream(1).data=bpm;

save 'outputStream' outputStream;
```

LISTING D.2: MATLAB script to get respiration rate

```
% fetch time and raw sensor data for 4 channels
load('inputStream.mat');
t=inputStream(1).timestamp;%time data
x1=inputStream(1).data;%raw sensor data
x2=inputStream(2).data;%raw sensor data
x3=inputStream(3).data;%raw sensor data
x4=inputStream(4).data;%raw sensor data

% filter raw sensor data
windowSize = 10;
b = (1/windowSize)*ones(1,windowSize);
a = 1;
y1 = filtfilt(b,a,x1);
y2 = filtfilt(b,a,x2);
y3 = filtfilt(b,a,x3);
y4 = filtfilt(b,a,x4);

% find peaks using MinPeakProminence for filtered sensor data
prom = 1;
[peakHeight1,peakTime1]=findpeaks(y1,t, 'MinPeakProminence', prom);
[peakHeight2,peakTime2]=findpeaks(y2,t, 'MinPeakProminence', prom);
[peakHeight3,peakTime3]=findpeaks(y3,t, 'MinPeakProminence', prom);
[peakHeight4,peakTime4]=findpeaks(y4,t, 'MinPeakProminence', prom);

% calculate respiration rate
bpm1=length(peakTime1);
bpm2=length(peakTime2);
bpm3=length(peakTime3);
bpm4=length(peakTime4);

outputStream(1).timestamp=t(1);
outputStream(1).data=bpm1;
outputStream(2).timestamp=t(1);
```

```
outputStream(2).data=bpm2;
outputStream(3).timestamp=t(1);
outputStream(3).data=bpm3;
outputStream(4).timestamp=t(1);
outputStream(4).data=bpm4;

save 'outputStream' outputStream;
```

LISTING D.3: MATLAB script to find best position

Appendix E

Workflow graphs created through yesWorkflow tools

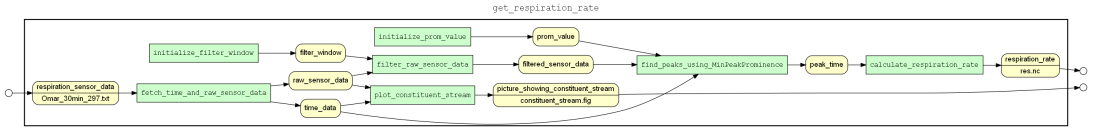

FIGURE E.1: Workflow graph to get respiration rate created through yesWorkflow

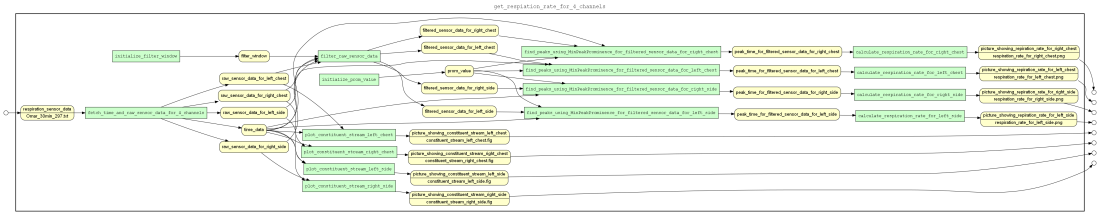

FIGURE E.2: Workflow graph to find the best position created through yesWorkflow

References

Ponomarev Alexey and Ponomareva Olga. Development of the theory of discrete fourier transform for solving problems of functional diagnostic of mechanical objects. In *2020 International Conference on Dynamics and Vibroacoustics of Machines (DVM)*, pages 1–7. IEEE, 2020.

M David Allen, Adriane Chapman, Barbara Blaustein, and Len Seligman. Capturing provenance in the wild. In *International Provenance and Annotation Workshop*, pages 98–101. Springer, 2010a.

M David Allen, Len Seligman, Barbara Blaustein, and Adriane Chapman. Provenance capture and use: A practical guide. *The MITRE Corporation*, 2010b.

M David Allen, Adriane Chapman, Len Seligman, and Barbara Blaustein. Provenance for collaboration: Detecting suspicious behaviors and assessing trust in information. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011 7th International Conference on*, pages 342–351. IEEE, 2011.

M David Allen, Adriane Chapman, and Barbara Blaustein. Provenance: Information for shared understanding, 2012.

Nicole Bidoit, Melanie Herschel, and Katerina Tzompanaki. Query-based why-not provenance with nedexplain. In *Extending database technology (EDBT)*, 2014.

Peter Buneman and Wang-Chiew Tan. Data provenance: What next? *ACM SIGMOD Record*, 47(3):5–16, 2019.

Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. Why and where: A characterization of data provenance. In *International conference on database theory*, pages 316–330. Springer, 2001.

Adriane Chapman and HV Jagadish. Why not? In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 523–534, 2009.

Adriane Chapman, Paolo Missier, Giulia Simonelli, and Riccardo Torlone. Capturing and querying fine-grained provenance of preprocessing pipelines in data science. *Proceedings of the VLDB Endowment*, 14(4):507–520, 2020.

Adriane Chapman, Abhirami Sasikant, Giulia Simonelli, Paolo Missier, and Riccardo Torlone. The right (provenance) hammer for the job: A comparison of data provenance instrumentation. In *Provenance in Data Science*, pages 25–45. Springer, 2021.

Adriane P Chapman, Hosagrahar V Jagadish, and Prakash Ramanan. Efficient provenance storage. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 993–1006, 2008.

Peng Chen and Beth A Plale. Big data provenance analysis and visualization. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 797–800. IEEE, 2015.

James Cheney, Adriane Chapman, and Joy Davidson. National physical laboratory scoping study on provenance, curation and data quality. 2018.

James Cheney, Adriane Chapman, Joy Davidson, and Alistair B Forbes. Data provenance, curation and quality in metrology. In *Advanced Mathematical and Computational Tools in Metrology and Testing XII*, pages 167–187. World Scientific, 2021.

Atanu Roy Chowdhury, Ben Falchuk, and Archan Misra. Medially: A provenance-aware remote health monitoring middleware. In *2010 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 125–134. IEEE, 2010.

Yingwei Cui and Jennifer Widom. Practical lineage tracing in data warehouses. In *Proceedings of 16th International Conference on Data Engineering (Cat. No. 00CB37073)*, pages 367–378. IEEE, 2000.

Yingwei Cui, Jennifer Widom, and Janet L Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Transactions on Database Systems (TODS)*, 25(2): 179–227, 2000.

Tom De Nies, Sara Magliacane, Ruben Verborgh, Sam Coppens, Paul Groth, Erik Mannens, and Rik Van de Walle. Git2prov: Exposing version control system content as w3c prov. In *International Semantic Web Conference (Posters & Demos)*, pages 125–128, 2013.

Wim De Pauw, Mihai Leţia, Buğra Gedik, Henrique Andrade, Andy Frenkiel, Michael Pfeifer, and Daby Sow. Visual debugging for stream processing applications. In *International Conference on Runtime Verification*, pages 18–35. Springer, 2010.

Vikas Deora, Arnaud Contes, Omer F Rana, Shrija Rajbhandari, Ian Wootten, Kifor Tamas, and Laszlo Z Varga. Navigating provenance information for distributed healthcare management. In *2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06)*, pages 859–865. IEEE, 2006.

Gulustan Dogan. *Protru: Leveraging provenance to enhance network trust in a wireless sensor network*. City University of New York, 2013.

Andréas Erlandsson and Mikael Gordani Shahri. Interactive fine-grained provenance for streaming-based analysis applications. 2021.

Felix CA Fernandes, Rutger LC van Spaendonck, and C Sidney Burrus. A new framework for complex wavelet transforms. *IEEE Transactions on signal processing*, 51(7):1825–1837, 2003.

Boris Glavic, Kyumars Sheykh Esmaili, Peter M Fischer, and Nesime Tatbul. Efficient stream provenance via operator instrumentation. *ACM Transactions on Internet Technology (TOIT)*, 14(1):7, 2014.

Lukasz Golab and M Tamer Özsu. Issues in data stream management. *ACM Sigmod Record*, 32(2):5–14, 2003.

Mikael Gordani Shahri, Andréas Erlandsson, Dimitris Palyvos-Giannas, and Vincenzo Gulisano. Poster: Twins, a middleware for adaptive streaming provenance at the edge. In *International Conference on Distributed Computing and Networking 2021*, pages 235–236, 2021.

Paul Groth. Use case creative commons. https://www.w3.org/2005/Incubator/prov/wiki/Use_Case_Creative_Commons, 2010. [Online; accessed 20-Nov-2021].

Paul Groth and Luc Moreau. Prov-overview. https://www.w3.org/TR/prov-overview/, 2013. [Online; accessed 20-May-2018].

Gongde Guo, Hui Wang, and David Bell. Data reduction and noise filtering for predicting times series. In *International Conference on Web-Age Information Management*, pages 421–429. Springer, 2002.

Olaf Hartig. Use case simple trustworthiness assessment. https://www.w3.org/2005/Incubator/prov/wiki/Use_Case_Simple_Trustworthiness_Assessment, 2010. [Online; accessed 20-Nov-2021].

Nick Hayward, Mahdi Shaban, James Badger, Isobel Jones, Yang Wei, Daniel Spencer, Stefania Isichei, Martin Knight, James Otto, Gurinder Rayat, et al. A capaciflector provides continuous and accurate respiratory rate monitoring for patients at rest and during exercise. *Journal of Clinical Monitoring and Computing*, pages 1–12, 2022.

David A Holland, Uri Jacob Braun, Diana Maclean, Kiran-Kumar Muniswamy-Reddy, and Margo I Seltzer. Choosing a data model and query language for provenance. In *Proceedings of the 2nd International Provenance and Annotation Workshop (IPAW'08)*. Springer, 2008.

Mohammad Rezwanul Huq. *An inference-based framework for managing data provenance*. PhD thesis, Ph. D. Dissertation. University of Twente, 2013.

Mohammad Rezwanul Huq, Andreas Wombacher, and Peter MG Apers. Inferring fine-grained data provenance in stream data processing: reduced storage cost, high accuracy. In *International Conference on Database and Expert Systems Applications*, pages 118–127. Springer, 2011.

HV Jagadish and Frank Olken. Database management for life sciences research. *ACM SIGMOD Record*, 33(2):15–20, 2004.

Oren N Jaspan, Roman Fleysher, and Michael L Lipton. Compressed sensing mri: a review of the clinical literature. *The British journal of radiology*, 88(1056):20150487, 2015.

Tamás Kifor, László Z Varga, Javier Vázquez-Salceda, Sergio Alvarez, Steven Willmott, Simon Miles, and Luc Moreau. Provenance in agent-mediated healthcare systems. *IEEE Intelligent Systems*, 21(6):38–46, 2006.

E Lai. Converting analog to digital signals and vice versa. *Practical Digital Signal Processing*, pages 14–49, 2003.

Barbara S Lerner and Emery R Boose. Collecting provenance in an interactive scripting environment. In *Workshop on the Theory and Practice of Provenance (TaPP), Cologne, Germany*, 2014.

Chiara Lindner, Sebastian Wolf, Jens Kiessling, and Frank Kühnemann. Fourier transform infrared spectroscopy with visible light. *Optics express*, 28(4):4426–4432, 2020.

Patrick C Loughlin, Frank Sebat, and John G Kellett. Respiratory rate: The forgotten vital sign—make it count! *Joint Commission journal on quality and patient safety*, 44(8): 494–499, 2018.

Milan Markovic, Peter Edwards, Martin Kollingbaum, and Alan Rowe. Modelling provenance of sensor data for food safety compliance checking. In *International Provenance and Annotation Workshop*, pages 134–145. Springer, 2016.

Jim McCusker. Experimental reproducability analysis. `https://www.w3.org/2005/Incubator/prov/wiki/Experimental_Reproducability_Analysis`, 2009. [Online; accessed 20-Nov-2021].

Timothy McPhillips, Shawn Bowers, Khalid Belhajjame, and Bertram Ludäscher. Retrospective provenance without a runtime provenance recorder. In *7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 15)*, 2015a.

Timothy McPhillips, Tianhong Song, Tyler Kolisnik, Steve Aulenbach, Khalid Belhajjame, Kyle Bocinsky, Yang Cao, Fernando Chirigati, Saumen Dey, Juliana Freire, et al. Yesworkflow: a user-oriented, language-independent tool for recovering workflow information from scripts. *arXiv preprint arXiv:1502.02403*, 2015b.

Simon Miles, Paul Groth, Miguel Branco, and Luc Moreau. The requirements of using provenance in e-science experiments. *Journal of Grid Computing*, 5(1):1–25, 2007.

Archan Misra, Marion Blount, Anastasios Kementsietsidis, Daby Sow, and Min Wang. Advances and challenges for scalable provenance in stream processing systems. In *International Provenance and Annotation Workshop*, pages 253–265. Springer, 2008.

Luc Moreau, Natalia Kwasnikowska, and Jan Van den Bussche. The foundations of the open provenance model. 2009.

Luc Moreau, Paolo Missier, Khalid Belhajjame, Reza B'Far, James Cheney, Sam Coppens, Stephen Cresswell, Yolanda Gil, Paul Groth, Graham Klyne, Timothy Lebo, Jim McCusker, Simon Miles, James Myers, Satya Sahoo, and Curt Tilmes. Prov-dm: The prov data model. https://www.w3.org/TR/prov-dm/#section-overview-components, 2013. [Online; accessed 20-May-2018].

Kiran-Kumar Muniswamy-Reddy, David A Holland, Uri Braun, and Margo I Seltzer. Provenance-aware storage systems. In *USENIX Annual Technical Conference, General Track*, pages 43–56, 2006.

Iman Naja and Luc Moreau. Use case provenance of decision making emergency response. https://www.w3.org/2005/Incubator/prov/wiki/Use_Case_Provenance_of_Decision_Making_Emergency_Response, 2010. [Online; accessed 20-Nov-2021].

Paula Olaya, Dominic Kennedy, Ricardo Llamas, Leobardo Valera, Rodrigo Vargas, Jay Lofstead, and Michela Taufer. Building trust in earth science findings through data traceability and results explainability. *IEEE Transactions on Parallel and Distributed Systems*, 2022.

Line Pouchard, Kevin Huck, Gyorgy Matyasfalvi, Dingwen Tao, Li Tang, Huub Van Dam, and Shinaje Yoo. Prescriptive provenance for streaming analysis of workflows at scale. In *2018 New York Scientific Data Summit (NYSDS)*, pages 1–6. IEEE, 2018.

Lukas Rupprecht, James C Davis, Constantine Arnold, Yaniv Gur, and Deepavali Bhagwat. Improving reproducibility of data science pipelines through transparent provenance capture. *Proceedings of the VLDB Endowment*, 13(12):3354–3368, 2020.

Watsawee Sansrimahachai. *Tracing fine-grained provenance in stream processing systems using a reverse mapping method*. PhD thesis, University of Southampton, 2012.

Watsawee Sansrimahachai, Mark J Weal, and Luc Moreau. Stream ancestor function: A mechanism for fine-grained provenance in stream processing systems. In *2012 Sixth International Conference on Research Challenges in Information Science (RCIS)*, pages 1–12. IEEE, 2012.

Watsawee Sansrimahachai, Luc Moreau, and Mark J Weal. An on-the-fly provenance tracking mechanism for stream processing systems. In *2013 IEEE/ACIS 12th International Conference on Computer and Information Science (ICIS)*, pages 475–481. IEEE, 2013.

Márcio José Sembay, Douglas Dyllon Jeronimo de Macedo, and Moisés Lima Dutra. A proposed approach for provenance data gathering. *Mobile Networks and Applications*, 26(1):304–318, 2021.

Julius Orion Smith. *Mathematics of the discrete Fourier transform (DFT): with audio applications*. Julius Smith, 2007.

Manolis Stamatogiannakis, Elias Athanasopoulos, Herbert Bos, and Paul Groth. Prov 2r: Practical provenance analysis of unstructured processes. *ACM Transactions on Internet Technology (TOIT)*, 17(4):37, 2017.

David W Stewart and Prem N Shamdasani. *Focus groups: Theory and practice*, volume 20. Sage publications, 2014.

Radovan Stojanović, Andrej Škraba, and Budimir Lutovac. A headset like wearable device to track covid-19 symptoms. In *2020 9th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–4. IEEE, 2020.

Isuru Suriarachchi, Sachith Withana, and Beth Plale. Big provenance stream processing for data intensive computations. In *2018 IEEE 14th International Conference on e-Science (e-Science)*, pages 245–255. IEEE, 2018.

Balder ten Cate, Cristina Civili, Evgeny Sherkhonov, and Wang-Chiew Tan. High-level why-not explanations using ontologies. In *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 31–43, 2015.

Nithya N Vijayakumar and Beth Plale. Towards low overhead provenance tracking in near real-time stream filtering. In *International Provenance and Annotation Workshop*, pages 46–54. Springer, 2006.

Min Wang, Marion Blount, John Davis, Archan Misra, and Daby Sow. A time-and-value centric provenance model and architecture for medical event streams. In *Proceedings of the 1st ACM SIGMOBILE international workshop on Systems*

and networking support for healthcare and assisted living environments, pages 95–100. ACM, 2007.

Neil M White and Omar Emara. personal communication, 2019.

Neil M White, Jordan Ash, Yang Wei, and Harry Akerman. A planar respiration sensor based on a capaciflector structure. *IEEE sensors letters*, 1(4):1–4, 2017.

Robin Whittemore, Ariana Chao, Myoungock Jang, Karl E Minges, and Chorong Park. Methods for knowledge synthesis: an overview. *Heart & Lung*, 43(5):453–461, 2014.

Paraskevi Zerva, Steffen Zschaler, and Simon Miles. Towards design support for provenance awareness: A classification of provenance questions. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 275–281. ACM, 2013.

Junli Zheng, Qiheng Ying, and Weili Yang. *Xinhao yu xitong (Signal and System)*. Higher Education Press Beijing, 2011.